

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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

import matplotlib.pyplot as plt
import seaborn as sns

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
import spacy

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

import sys
!{sys.executable} -m pip install nltk

Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-
packages (3.9.1)
Requirement already satisfied: click in
/usr/local/lib/python3.12/dist-packages (from nltk) (8.3.1)
Requirement already satisfied: joblib in
/usr/local/lib/python3.12/dist-packages (from nltk) (1.5.2)

```

```
Requirement already satisfied: regex>=2021.8.3 in  
/usr/local/lib/python3.12/dist-packages (from nltk) (2025.11.3)  
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-  
packages (from nltk) (4.67.1)
```

```
!pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-  
packages (3.9.1)  
Requirement already satisfied: click in  
/usr/local/lib/python3.12/dist-packages (from nltk) (8.3.1)  
Requirement already satisfied: joblib in  
/usr/local/lib/python3.12/dist-packages (from nltk) (1.5.2)  
Requirement already satisfied: regex>=2021.8.3 in  
/usr/local/lib/python3.12/dist-packages (from nltk) (2025.11.3)  
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-  
packages (from nltk) (4.67.1)
```

```
import nltk
```

```
nltk.download('punkt')  
nltk.download('wordnet')  
nltk.download('omw-1.4')  
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...  
[nltk_data]   Package punkt is already up-to-date!  
[nltk_data] Downloading package wordnet to /root/nltk_data...  
[nltk_data]   Package wordnet is already up-to-date!  
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...  
[nltk_data]   Package omw-1.4 is already up-to-date!  
[nltk_data] Downloading package stopwords to /root/nltk_data...  
[nltk_data]   Package stopwords is already up-to-date!
```

```
True
```

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

```
lemmatizer = WordNetLemmatizer()  
print("NLTK resources loaded successfully ☑")
```

```
NLTK resources loaded successfully ☑
```

```
from google.colab import files
```

```
import pandas as pd
```

```
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

Saving unbalanceddataset.xlsx to unbalanceddataset (1).xlsx

```
df = pd.read_excel('unbalanceddataset.xlsx')
df.head()

{"summary": "{\n  \"name\": \"df\",\n  \"rows\": 2600,\n  \"fields\": [\n    {\n      \"column\": \"text\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 2600,\n        \"samples\": [\n          \" Looking forward to following your journey on this endeavor - just subscribed to your blog\",\n          \"Frustrated with copying 13Gigs across USB 1.1.Stupid old servers\",\n          \" Thanks so much Jon.....same to your mom That is so sweet of you to think of all of us\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"sentiment\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 2,\n        \"samples\": [\n          \"negative\",\n          \"positive\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}", "type": "dataframe", "variable_name": "df"}

from sklearn.metrics import accuracy_score, classification_report

def evaluate_model(model_name, y_true, y_pred, df):
    print(f"\n--- Evaluation Report for {model_name} ---")

    accuracy = accuracy_score(y_true, y_pred)
    print(f"Accuracy: {accuracy:.4f}")

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

    df_errors = df.copy()
    df_errors['y_true'] = y_true
    df_errors['y_pred'] = y_pred
    misclassified = df_errors[df_errors['y_true'] != df_errors['y_pred']]

    if not misclassified.empty:
        print("\nFirst 5 Misclassified Samples:")
        print(misclassified.head())
    else:
        print("\nNo misclassified samples found!")

    print(f"--- End of Report for {model_name} ---")

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

```

def run_random_forest_gridsearch(df, target_column, param_grid=None,
test_size=0.2, random_state=42):
    X = df.drop(columns=[target_column])
    y = df[target_column]

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

    if param_grid is None:
        param_grid = {
            'n_estimators': [100, 200],
            'max_depth': [None, 10, 20],
            'min_samples_split': [2, 5],
            'min_samples_leaf': [1, 2],
            'max_features': ['auto', 'sqrt']
        }

    rf = RandomForestClassifier(random_state=random_state)
    grid_search = GridSearchCV(
        estimator=rf,
        param_grid=param_grid,
        cv=5,
        scoring='accuracy',
        n_jobs=-1
    )

    grid_search.fit(X_train, y_train)
    best_model = grid_search.best_estimator_
    print(f"\nBest Parameters: {grid_search.best_params_}")

    y_pred = best_model.predict(X_test)
    evaluate_model("Random Forest", y_test, y_pred,
df.iloc[y_test.index])

    return best_model

df = pd.read_excel('unbalanceddataset.xlsx')
df.head()

{"summary": "{\n  \"name\": \"df\",\n  \"rows\": 2600,\n  \"fields\": [\n    {\n      \"column\": \"text\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 2600,\n        \"samples\": [\n          \" Looking forward to following your journey on this endeavor - just subscribed to your blog\",\n          \"Frustrated with copying 13Gigs across USB 1.1.Stupid old servers\",\n          \" Thanks so much Jon.....same to your mom That is so sweet of you to think of all of us\"
        ],\n      }
    }
  ]\n}"}

```

```

{"semantic_type\\": "\\\"",
  "description\\": "\\\"",
  "column\\": "sentiment",
  "properties\\": {
    "dtype\\": "category",
    "num_unique_values\\": 2,
    "samples\\": [
      "negative",
      "positive"
    ]
  },
  "semantic_type\\": "\\\"",
  "description\\": "\\\"",
  "column\\": "sentiment",
  "dtype\\": "category",
  "num_unique_values\\": 2,
  "samples\\": [
    "negative",
    "positive"
  ]
},
{"type": "dataframe", "variable_name": "df"}

```

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

```

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

```

```
stemmer = PorterStemmer()
```

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

```

def preprocess_text(text):
    text = text.lower()

```

```
import nltk
```

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

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

```
[nltk_data] Downloading package punkt to /root/nltk_data...
```

```
[nltk_data] Package punkt is already up-to-date!
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
```

```
[nltk_data] Package stopwords is already up-to-date!
```

```
True
```

```
import nltk
```

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

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

```
nltk.data.path.append('/usr/share/nltk_data')
```

```
nltk.data.path.append('/usr/local/share/nltk_data')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
```

```
[nltk_data] Package punkt is already up-to-date!
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
```

```
[nltk_data] Package stopwords is already up-to-date!
```

```
import pandas as pd
```

```
import re
```

```
from nltk.corpus import stopwords
```

```
from nltk.stem import PorterStemmer
```

```
import nltk
```

```

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

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'http\S+|www.\S+', '', text)
    text = re.sub(r'<.*?>', '', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    tokens = text.split() # تقسيم النص بدون word_tokenize
    tokens = [word for word in tokens if word not in stop_words]
    stemmed_tokens = [stemmer.stem(word) for word in tokens]
    clean_text = ' '.join(stemmed_tokens)
    return clean_text

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

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!

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
Name: clean_text, dtype: object

import pandas as pd
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import nltk

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

df = pd.read_excel('unbalanceddataset.xlsx')

if 'clean_text' not in df.columns:
    def preprocess_text(text):
        text = str(text).lower()
        text = re.sub(r'http\S+|www.\S+', '', text)
        text = re.sub(r'<.*?>', '', text)
        text = re.sub(r'[^a-zA-Z\s]', '', text)
        tokens = text.split()
        tokens = [word for word in tokens if word not in stop_words]

```

```

        stemmed_tokens = [stemmer.stem(word) for word in tokens]
        return ' '.join(stemmed_tokens)

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

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

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

{"summary": "{\n  \"name\": \"df[['text', 'clean_text']]\", \n  \"rows\": 5,\n  \"fields\": [\n    {\n      \"column\": \"text\", \n      \"properties\": {\n        \"dtype\": \"string\", \n        \"num_unique_values\": 5, \n        \"samples\": [\n          \"haha aww hun i bet you are more creative than me\", \n          \" - just took over the #1 Most Endorsed spot on twindexx.com - thanks to the endorsement by \", \n          \"_pickle lol, thank you very much Hope you're having a great day!\", \n          ], \n          \"semantic_type\": \"\", \n          \"description\": \"\", \n          } \n        ], \n        \"column\": \"clean_text\", \n        \"properties\": {\n          \"dtype\": \"string\", \n          \"num_unique_values\": 5, \n          \"samples\": [\n            \"haha aww hun bet creativ\", \n            \"took endors spot twindexxcom thank endors\", \n            \"_pickl lol thank much hope your great day\", \n            ], \n            \"semantic_type\": \"\", \n            \"description\": \"\" \n          } \n        ] \n      ], \n      \"type\": \"dataframe\"}

import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from imblearn.over_sampling import SMOTE

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

df = pd.read_excel('unbalanceddataset.xlsx')

def preprocess_text(text):
    text = str(text).lower()
    text = re.sub(r'http\S+|www.\S+', '', text)
    text = re.sub(r'<.*?>', '', text)
    text = re.sub(r'^a-zA-Z\s', '', text)

```

```

    tokens = text.split()
    tokens = [stemmer.stem(word) for word in tokens if word not in
stop_words]
    return ' '.join(tokens)

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

print("=== Class Distribution ===")
print(df['sentiment'].value_counts())

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

vectorizer = TfidfVectorizer(max_features=5000)
X_vect = vectorizer.fit_transform(X)

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

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

print("\n=== Original (Imbalanced) Dataset ===")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)

rf_bal = RandomForestClassifier(random_state=42)
rf_bal.fit(X_train_bal, y_train_bal)
y_pred_bal = rf_bal.predict(X_test)

print("\n=== Balanced Dataset (SMOTE) ===")
print(classification_report(y_test, y_pred_bal))
print(confusion_matrix(y_test, y_pred_bal))

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!

=== Class Distribution ===
sentiment
positive    2000
negative     600
Name: count, dtype: int64

=== Original (Imbalanced) Dataset ===

```

	precision	recall	f1-score	support
negative	0.00	0.00	0.00	600
positive	0.00	0.00	0.00	2000
micro avg	0.00	0.00	0.00	2600
macro avg	0.00	0.00	0.00	2600
weighted avg	0.00	0.00	0.00	2600

negative	0.93	0.45	0.61	120
positive	0.86	0.99	0.92	400
accuracy			0.87	520
macro avg	0.89	0.72	0.76	520
weighted avg	0.87	0.87	0.85	520

```
[[ 54  66]
 [  4 396]]
```

=== Balanced Dataset (SMOTE) ===

	precision	recall	f1-score	support
negative	0.80	0.63	0.71	120
positive	0.90	0.95	0.92	400
accuracy			0.88	520
macro avg	0.85	0.79	0.82	520
weighted avg	0.87	0.88	0.87	520

```
[[ 76  44]
 [ 19 381]]
```

```
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
```

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

```
df = pd.read_excel('unbalanceddataset.xlsx')
```

```
def preprocess_text(text):
    text = str(text).lower()
    text = re.sub(r'http\S+|www.\S+', '', text)
    text = re.sub(r'<.*?>', '', text)
    text = re.sub(r'^a-zA-Z\s', '', text)
    tokens = text.split()
    tokens = [stemmer.stem(word) for word in tokens if word not in
stop_words]
    return ' '.join(tokens)
```

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

```

df = df.dropna()

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

vectorizer = TfidfVectorizer(max_features=5000)
X_vect = vectorizer.fit_transform(X)

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

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

print("=== Random Forest Classifier Performance ===")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!

=== Random Forest Classifier Performance ===

```

	precision	recall	f1-score	support
negative	0.93	0.45	0.61	120
positive	0.86	0.99	0.92	400
accuracy			0.87	520
macro avg	0.89	0.72	0.76	520
weighted avg	0.87	0.87	0.85	520

```

[[ 54  66]
 [  4 396]]

import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

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

df = pd.read_excel('unbalanceddataset.xlsx')

def preprocess_text(text):

```

```

text = str(text).lower()
text = re.sub(r'http\S+|www.\S+', '', text)
text = re.sub(r'<.*?>', '', text)
text = re.sub(r'^a-zA-Z\s', '', text)
tokens = text.split()
tokens = [stemmer.stem(word) for word in tokens if word not in
stop_words]
return ' '.join(tokens)

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

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("X_train (first 5 samples):")
print(X_train[:5])
print("\nX_test (first 5 samples):")
print(X_test[:5])
print("\ny_train (first 5 samples):")
print(y_train[:5])
print("\ny_test (first 5 samples):")
print(y_test[:5])

print("\nTF-IDF Shapes:")
print("X_train_vec shape:", X_train_vec.shape)
print("X_test_vec shape:", X_test_vec.shape)

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!

X_train (first 5 samples):
1384      sound like challeng see would use abl explicit...
250                                           orral fun
2256      love fli live chat dont like sick
1228      ohh kayi fasho u go tonight fun
785      wish happi mother day
Name: clean_text, dtype: object

X_test (first 5 samples):
124      lolblow kiss u catch daddi neway im chillin h...
2229      may th happi star war day

```

```
2165      oh see like morn right happi breakfast thumb
195      haha phew ive never said
1037    constantli listen song fli mebc cutest songpet...
Name: clean_text, dtype: object
```

```
y_train (first 5 samples):
1384    positive
250     positive
2256    positive
1228    positive
785     positive
Name: sentiment, dtype: object
```

```
y_test (first 5 samples):
124     positive
2229    positive
2165    positive
195     negative
1037    positive
Name: sentiment, dtype: object
```

```
TF-IDF Shapes:
X_train_vec shape: (1820, 3657)
X_test_vec shape: (780, 3657)
```

```
from imblearn.over_sampling import RandomOverSampler
import pandas as pd
```

```
ros = RandomOverSampler(random_state=42)
X_train_res, y_train_res = ros.fit_resample(X_train_vec.toarray(),
y_train)
```

```
print("Original class distribution:")
print(y_train.value_counts())

print("\nResampled class distribution:")
print(pd.Series(y_train_res).value_counts())
```

```
Original class distribution:
sentiment
positive    1400
negative     420
Name: count, dtype: int64
```

```
Resampled class distribution:
sentiment
positive    1400
negative    1400
Name: count, dtype: int64
```

```
from imblearn.under_sampling import RandomUnderSampler
import pandas as pd

rus = RandomUnderSampler(random_state=42)
X_train_res, y_train_res = rus.fit_resample(X_train_vec.toarray(),
y_train)

print("Original class distribution:")
print(y_train.value_counts())

print("\nResampled class distribution:")
print(pd.Series(y_train_res).value_counts())
```

```
Original class distribution:
sentiment
positive    1400
negative     420
Name: count, dtype: int64
```

```
Resampled class distribution:
sentiment
negative     420
positive     420
Name: count, dtype: int64
```