

```

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_re

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

```

```

def evaluate_model(model_name, y_true, y_pred):

    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)

    report = classification_report(y_true, y_pred)

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

    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()

    return metrics

```

```

df = pd.read_csv("Arabic Sentiment Analysis Dataset - SS2030.csv")
print(df.head())

```

	text	Sentiment
0	حقوق المرأة ❤️❤️❤️ https://t.co/Mzf90Ta5g1	1
1	RT @___IHAVENOIDEA: حقوق المرأة في الإسلام. ht...	1
2	RT @saud_talep: Retweeted 1 ... لجنة التنمية بشيرا	
3	RT @MojKsa: 1 ... حقوق المرأة التي تضمنها لها وزارة	
4	RT @abm112211: 1 ... ولي امر الزوجة او ولي الزوجة او...	

```

import re
import nltk
nltk.download('punkt')
nltk.download('stopwords')

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

def preprocess_arabic_text(text):

    diacritics = re.compile(r'[\u0617-\u061A\u064B-\u0652]')

```

```
text = re.sub(diacritics, '', text)

text = re.sub(r'[إإإ]', 'ا', text)
text = re.sub(r'ى', 'ي', text)
text = re.sub(r'و', 'و', text)
text = re.sub(r'ئ', 'ي', text)
text = re.sub(r'ة', 'ه', text)

text = re.sub(r'_+', '', text)

text = re.sub(r'[A-Za-z0-9]', '', text)
text = re.sub(r'^\u0600-\u06FF\s]', '', text)

tokens = word_tokenize(text)

arabic_stopwords = set(stopwords.words("arabic"))
tokens = [word for word in tokens if word not in arabic_stopwords]

clean_text = ' '.join(tokens)
return clean_text
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
import nltk
nltk.download('punkt_tab')
df["clean_text"] = df["text"].apply(preprocess_arabic_text)
df = df.dropna()
df.head()
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

	text	Sentiment	clean_text	
0	حقوق المرأة ❤️❤️❤️ https://t.co/Mzf90Ta5g1	1	حقوق المراه	
1	RT @___IHAVENOIDEA: حقوق المرأة في الإسلام. ht...	1	حقوق المراه الاسلام	
2	RT @saud_talep: Retweeted لجنة التنمية بشبرا (...)	1	لجنه التنميه بشبرا زال التسجيل مستمر دوره حقوق	
3	RT @MojKsa: حقوق المرأة التي تضمنها لها وزارة ...	1	حقوق المراه تضمنها وزاره العدل	
4	RT @abm112211: ولي امر الزوجة او ولي الزوجة او ...	1	ولي امر الزوجه او ولي الزوجه او ولي المراه الا	

Next steps: [Generate code with df](#) [New interactive sheet](#)

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

	count
Sentiment	
1	2436
0	1816

dtype: int64

```
X = df["clean_text"]
y = df["Sentiment"]
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42, stratify=y)
```

```
vectorizer1 = CountVectorizer()
X_train_Bag_word = vectorizer1.fit_transform(X_train)
X_test_Bag_word = vectorizer1.transform(X_test)
```

```
rf_classifier = RandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
}

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

grid_search.fit(X_train_Bag_word, y_train)

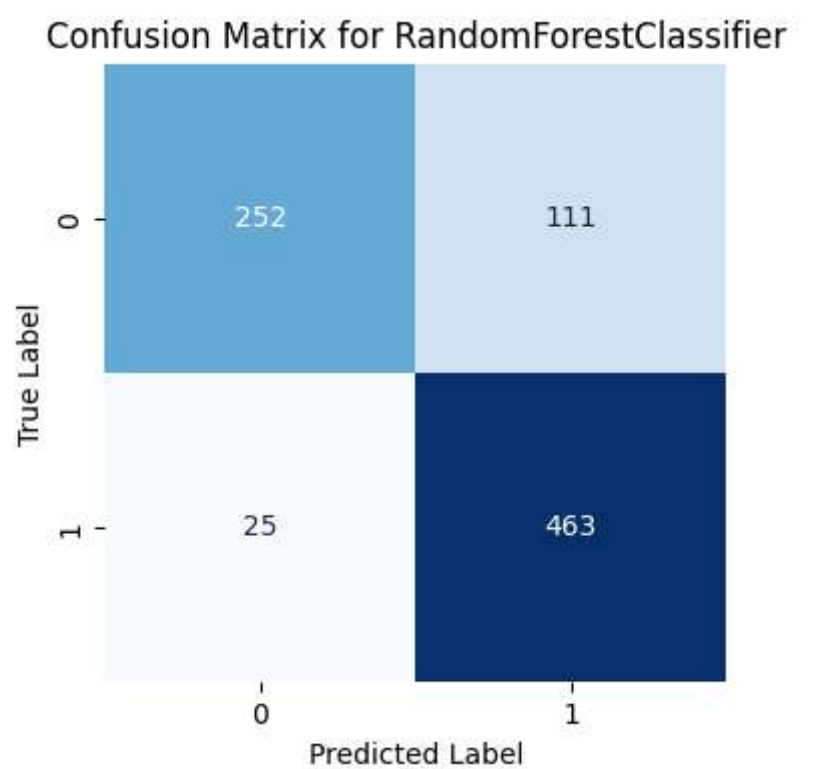
best_rf_classifier = grid_search.best_estimator_

y_pred = best_rf_classifier.predict(X_test_Bag_word)

evaluation_results = evaluate_model('RandomForestClassifier', y_test, y_pred)

for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}: {value}" if not isinstance(value, float)
              else f"{key}: {value:.4f}")

print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```



Model Name: RandomForestClassifier
Accuracy: 0.8402
Precision: 0.8506
Recall: 0.8402
F1 Score: 0.8359

	precision	recall	f1-score	support
0	0.91	0.69	0.79	363
1	0.81	0.95	0.87	488
accuracy			0.84	851
macro avg	0.86	0.82	0.83	851
weighted avg	0.85	0.84	0.84	851

Best hyperparameters found by GridSearchCV:
{ 'max_depth': None, 'n_estimators': 100 }

```
LR = LogisticRegression(random_state=42)

param_grid = {
    'solver': ['liblinear', 'lbfgs'],
}

grid_search = GridSearchCV(estimator=LR,
                           param_grid=param_grid,
```

```
cv=5,
scoring='accuracy',
n_jobs=-1)

grid_search.fit(X_train_Bag_word, y_train)

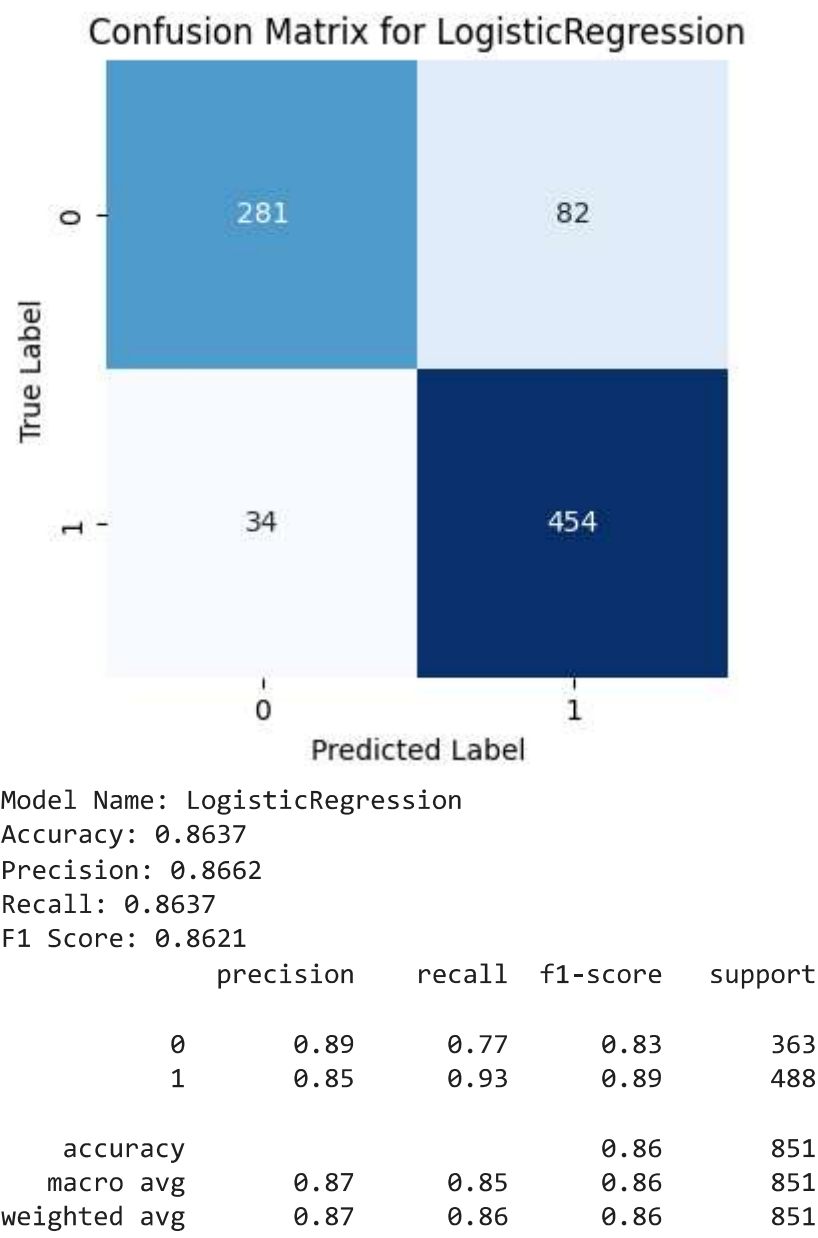
best_LR = grid_search.best_estimator_

y_pred = best_LR.predict(X_test_Bag_word)

evaluation_results = evaluate_model('LogisticRegression', y_test, y_pred)

for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}: {value}" if not isinstance(value, float)
              else f"{key}: {value:.4f}")

print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```



```
Best hyperparameters found by GridSearchCV:
{'solver': 'lbfgs'}
```

```
vectorizer2 = TfidfVectorizer(ngram_range=(1, 2))
X_train_TFIDF = vectorizer2.fit_transform(X_train)
X_test_TFIDF = vectorizer2.transform(X_test)
```

```
DT = DecisionTreeClassifier(random_state=42)

param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
}

grid_search = GridSearchCV(estimator=DT,
                           param_grid=param_grid,
                           cv=10,
                           scoring='accuracy',
                           n_jobs=-1)
```

```
grid_search.fit(X_train_TFIDF, y_train)

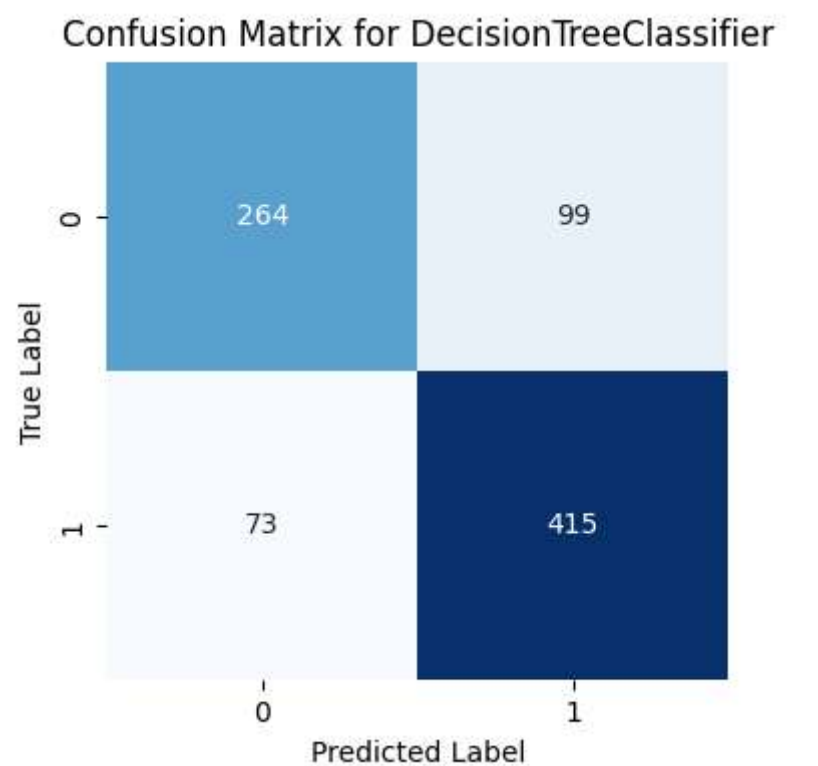
best_DT = grid_search.best_estimator_

y_pred = best_DT.predict(X_test_TFIDF)

evaluation_results = evaluate_model('DecisionTreeClassifier', y_test, y_pred)

for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}: {value}" if not isinstance(value, float)
              else f"{key}: {value:.4f}")

print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```



Model Name: DecisionTreeClassifier
Accuracy: 0.7979
Precision: 0.7972
Recall: 0.7979
F1 Score: 0.7968

	precision	recall	f1-score	support
0	0.78	0.73	0.75	363
1	0.81	0.85	0.83	488
accuracy			0.80	851
macro avg	0.80	0.79	0.79	851
weighted avg	0.80	0.80	0.80	851

Best hyperparameters found by GridSearchCV:
{'criterion': 'gini', 'max_depth': None}

```
SVM = SVC(random_state=42)

param_grid = {
    'kernel': ['linear', 'rbf', 'poly'],
}

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

grid_search.fit(X_train_TFIDF, y_train)

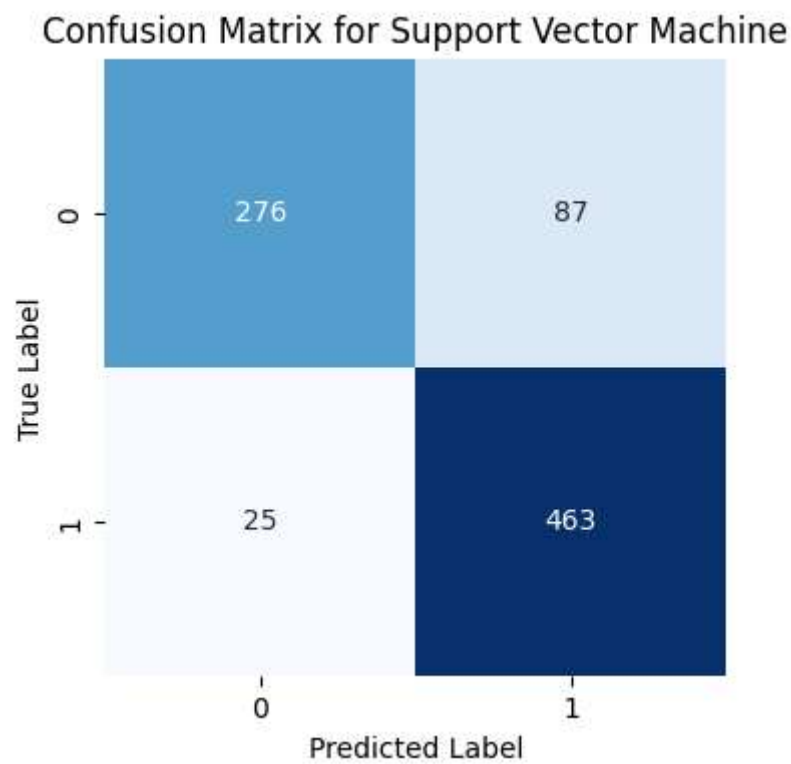
best_SVM = grid_search.best_estimator_

y_pred = best_SVM.predict(X_test_TFIDF)

evaluation_results = evaluate_model('Support Vector Machine', y_test, y_pred)
```

```
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}: {value}" if not isinstance(value, float)
              else f"{key}: {value:.4f}")

print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```



Model Name: Support Vector Machine
Accuracy: 0.8684
Precision: 0.8739
Recall: 0.8684
F1 Score: 0.8662

	precision	recall	f1-score	support
0	0.92	0.76	0.83	363
1	0.84	0.95	0.89	488
accuracy			0.87	851
macro avg	0.88	0.85	0.86	851
weighted avg	0.87	0.87	0.87	851

Best hyperparameters found by GridSearchCV:
{'kernel': 'linear'}