

```
In [1]: 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
)

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

import matplotlib.pyplot as plt
import seaborn as sns

import re
import nltk

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

from collections import Counter

from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE

from gensim.models import Word2Vec
```

```
In [2]: nltk.download('punkt')
nltk.download('stopwords')
nltk.download('punkt_tab')
```

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

Out[2]: True

```
In [3]: df = pd.read_csv("Arabic Sentiment Analysis Dataset - SS2030.csv")
df.head()
```

Out[3]:

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

```
In [4]: df.info()

<class 'pandas.DataFrame'>
RangeIndex: 4252 entries, 0 to 4251
Data columns (total 2 columns):
#   Column      Non-Null Count  Dtype  
---  -
0   text        4252 non-null   str
1   Sentiment   4252 non-null   int64  
dtypes: int64(1), str(1)
memory usage: 66.6 KB
```

```
In [5]: df['Sentiment'].value_counts()
```

Out[5]: Sentiment

1	2436
0	1816

Name: count, dtype: int64

```
In [6]: import pandas as pd
import numpy as np
import sklearn
import matplotlib
import seaborn
import nltk
import gensim
import imblearn

print("Pandas:", pd.__version__)
```

```

print("NumPy:", np.__version__)
print("Scikit-learn:", sklearn.__version__)
print("NLTK:", nltk.__version__)
print("Gensim:", gensim.__version__)
print("Imbalanced-learn:", imblearn.__version__)

print("\nAll libraries loaded successfully.")

```

Pandas: 3.0.5
 NumPy: 2.5.2
 Scikit-learn: 1.9.0
 NLTK: 3.10.3
 Gensim: 4.4.0
 Imbalanced-learn: 0.14.2

All libraries loaded successfully.

```

In [7]: from nltk.tokenize import word_tokenize
        from nltk.corpus import stopwords
        import re

```

```

In [8]: def preprocess_arabic_text(text):

        # Convert text to string
        text = str(text)

        # 1. Remove diacritics (Tashkeel)
        diacritics = re.compile(r'[\u0617-\u061A\u064B-\u0652]')
        text = re.sub(diacritics, '', text)

        # 2. Normalize Arabic Letters
        text = re.sub(r'[\u0600-\u0604]', '', text)
        text = re.sub(r'ي', 'ى', text)
        text = re.sub(r'و', 'ؤ', text)
        text = re.sub(r'ي', 'ئ', text)
        text = re.sub(r'ة', 'ه', text)

        # 3. Remove Tatweel
        text = re.sub(r'ـ+', '', text)

        # 4. Remove URLs
        text = re.sub(r'https?://\S+|www\.\S+', '', text)

        # 5. Remove English Letters and numbers
        text = re.sub(r'[A-Za-z0-9]', '', text)

        # 6. Remove non-Arabic characters and punctuation
        text = re.sub(r'^[\u0600-\u06FF\s]', '', text)

        # 7. Tokenize
        tokens = word_tokenize(text)

        # 8. Remove Arabic stopwords
        arabic_stopwords = set(stopwords.words("arabic"))

        tokens = [

```

```

    word for word in tokens
    if word not in arabic_stopwords
]

# 9. Reconstruct clean sentence
clean_text = ' '.join(tokens)

return clean_text

```

```

In [9]: df["clean_text"] = df["text"].apply(preprocess_arabic_text)

df = df.dropna()

df.head()

```

Out[9]:

	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	لجنة التنمية 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

```

In [10]: df[['text', 'clean_text', 'Sentiment']].head(10)

```

Out[10]:

	text	clean_text	Sentiment
0	حقوق المرأة ❤️❤️❤️ https://t.co/Mzf90Ta5g1	حقوق المراه	1
1	RT @__IHAVENOIDEA: حقوق المرأة في الإسلام. ht...	حقوق المراه الاسلام	1
2	RT @saud_talep: Retweeted لجنة التنمية بشبرا (...)	لجنه التنميه بشبرا زال التسجيل مستمر...دوره حقوق	1
3	RT @MojKsa: حقوق المرأة التي تضمنها لها وزارة ...	حقوق المراه تضمنها وزاره العدل	1
4	RT @abm112211: ولي امر الزوجة او ولي الزوجة او...	ولي امر الزوجه او ولي الزوجه او...المراه الا	1
5	RT @fatemaar7: انا كامرأة يوجعني كل حزن...نساء ا	انا كامراه يوجعني حزن نساء العالم،...سواء تعنفت	1
6	@Samar_K_ 🤗🤗🤗\n اومن حقوق المرأة...أنك تفتح	اومن حقوق المراه انك تفتحها الباب	1
7	@AliFCD @ShamsanM @AlyemenNor @space_dist @m2r...	غسق سيبس حلوه الحلوات اشخاص...هويتهم كيانهم المت	1
8	😊 مبدائيا انا مع حقوق المرأة وضد حقوق الرجل	مبدائيا انا حقوق المراه وضد حقوق الرجل	1
9	RT @Dresraamohammed: السؤال بقي للناس...الي شغال	السوال بقي للناس الي شغاله...حقوقالانسان حقوقالم	1

In [11]: `df[df['clean_text'].str.strip() == '']`

Out[11]:

	text	Sentiment	clean_text
1198	#NAME?	0	

In [13]: `df = df[df['clean_text'].str.strip() != '']`In [14]: `df.shape`

Out[14]: (4251, 3)

In [15]: `X = df['clean_text']
y = df['Sentiment']`In [16]: `X_train, X_test, y_train, y_test = train_test_split(
X,
y,
test_size=0.20,
random_state=42,
stratify=y
)`

```
In [17]: print("Training samples:", len(X_train))
        print("Testing samples:", len(X_test))
```

Training samples: 3400

Testing samples: 851

```
In [18]: vectorizer1 = CountVectorizer()

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

```
In [19]: # 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],
        }
```

```
In [20]: # Initialize GridSearchCV
        grid_search = GridSearchCV(
            estimator=rf_classifier,
            param_grid=param_grid,
            cv=5,
            scoring='accuracy',
            n_jobs=-1
        )
```

```
In [21]: # Fit the grid search to the data
        grid_search.fit(X_train_Bag_word, y_train)
```

```
Out[21]:
```

‣ **GridSearchCV** ⓘ ?

‣ **best_estimator_: RandomForestClassifier**

‣ **RandomForestClassifier** ?

```
In [24]: # Get the best estimator from grid search
        best_rf_classifier = grid_search.best_estimator_
```

```
In [28]: # Make predictions using the best model
        y_pred = best_rf_classifier.predict(X_test_Bag_word)
```

```
In [30]: 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
    }

    # 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)
    )

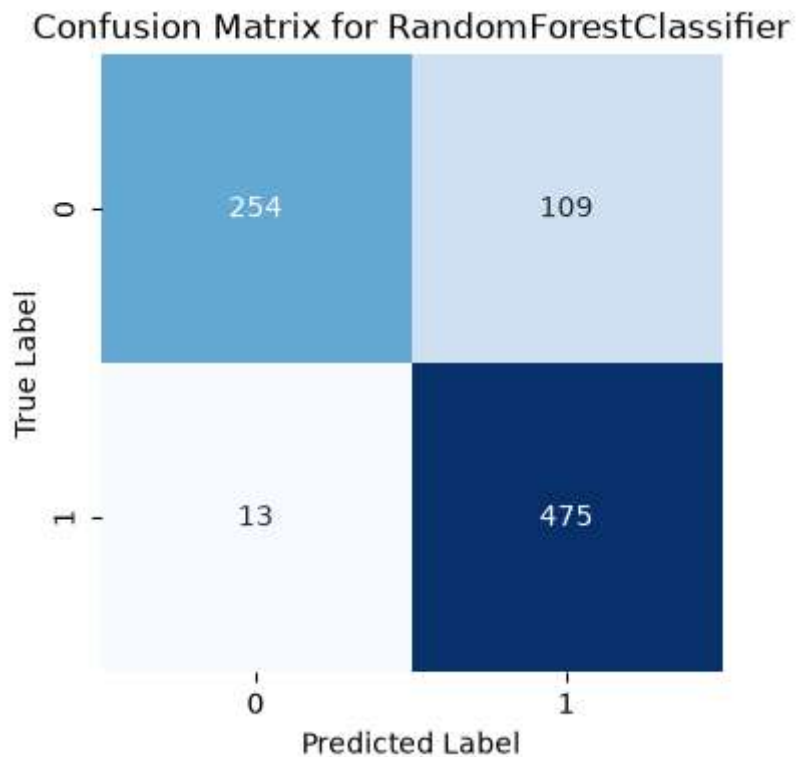
    plt.title(
        f'Confusion Matrix for {model_name}'
    )

    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
```

```
plt.show()

return metrics
```

```
In [31]: # Evaluate the model
evaluation_results = evaluate_model(
    'RandomForestClassifier',
    y_test,
    y_pred
)
```



```
In [32]: # Print the evaluation results
for key, value in evaluation_results.items():

    if key == 'Classification Report':
        print(value)

    else:
        print(
            f"{key}: {value:.4f}"
            if isinstance(value, float)
            else f"{key}: \n{value}"
        )
```

Model Name:

RandomForestClassifier

Accuracy: 0.8566

Precision: 0.8722

Recall: 0.8566

F1 Score: 0.8521

	precision	recall	f1-score	support
0	0.95	0.70	0.81	363
1	0.81	0.97	0.89	488
accuracy			0.86	851
macro avg	0.88	0.84	0.85	851
weighted avg	0.87	0.86	0.85	851

```
In [33]: # Print the best parameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

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

```
In [35]: best_rf_classifier = grid_search.best_estimator_

y_pred = best_rf_classifier.predict(
    X_test_Bag_word
)
```

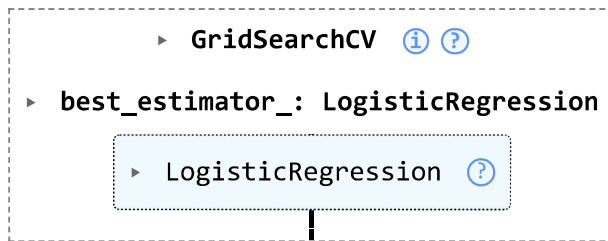
```
In [36]: # Define the Logistic Regression model
LR = LogisticRegression(random_state=42)
```

```
In [37]: # Define the hyperparameters for grid search
param_grid = {
    'solver': ['liblinear', 'lbfgs']
}
```

```
In [38]: # Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=LR,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)
```

```
In [39]: # Fit the grid search to the data
grid_search.fit(
    X_train_Bag_word,
    y_train
)
```

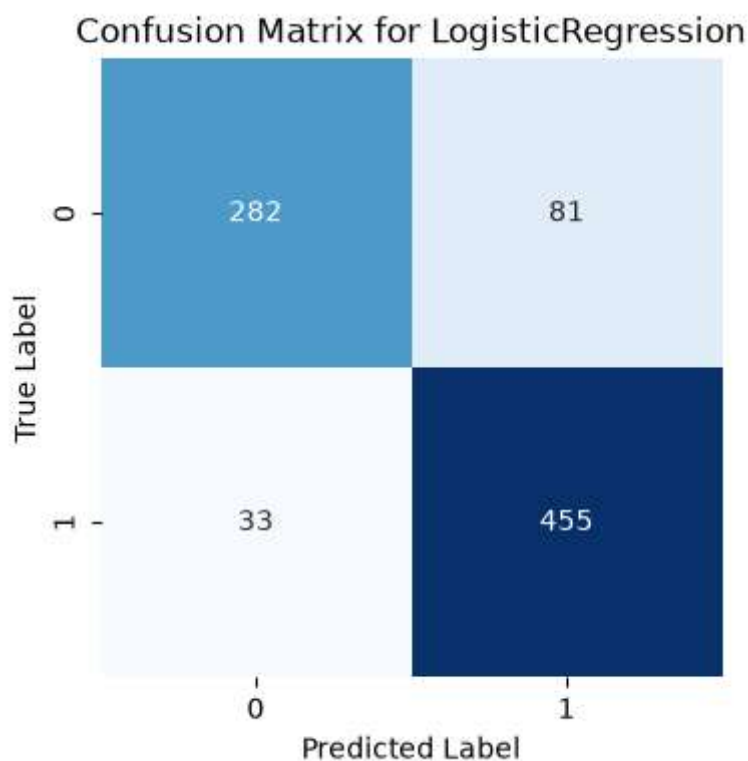
Out[39]:



```
In [41]: # Get the best estimator from grid search
best_LR = grid_search.best_estimator_
```

```
In [42]: # Make predictions using the best model
y_pred = best_LR.predict(
    X_test_Bag_word
)
```

```
In [43]: # Evaluate the model
evaluation_results = evaluate_model(
    'LogisticRegression',
    y_test,
    y_pred
)
```



```
In [44]: # Print the evaluation results
for key, value in evaluation_results.items():

    if key == 'Classification Report':
        print(value)

    else:
```

```

print(
    f"{key}: {value:.4f}"
    if isinstance(value, float)
    else f"{key}: \n{value}"
)

```

Model Name:

LogisticRegression

Accuracy: 0.8660

Precision: 0.8687

Recall: 0.8660

F1 Score: 0.8644

	precision	recall	f1-score	support
0	0.90	0.78	0.83	363
1	0.85	0.93	0.89	488
accuracy			0.87	851
macro avg	0.87	0.85	0.86	851
weighted avg	0.87	0.87	0.86	851

```

In [52]: # Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Best hyperparameters found by GridSearchCV:
{'solver': 'liblinear'}

```

In [56]: vectorizer2 = TfidfVectorizer(
          ngram_range=(1, 2)
        )

X_train_TFIDF = vectorizer2.fit_transform(X_train)

X_test_TFIDF = vectorizer2.transform(X_test)

```

```

In [57]: print("Training TF-IDF shape:", X_train_TFIDF.shape)
print("Testing TF-IDF shape:", X_test_TFIDF.shape)

```

Training TF-IDF shape: (3400, 66208)

Testing TF-IDF shape: (851, 66208)

```

In [58]: # Define the Decision Tree model
DT = DecisionTreeClassifier(random_state=42)

```

```

In [59]: # Define the hyperparameters for grid search
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30]
}

```

```

In [60]: # Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=DT,
    param_grid=param_grid,

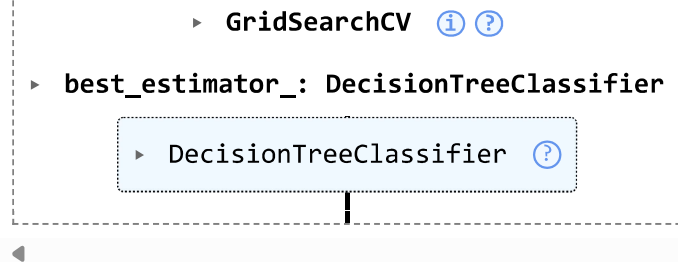
```

```
cv=10,  
scoring='accuracy',  
n_jobs=-1  
)
```

```
In [61]: # Fit the grid search to the data  
grid_search.fit(  
    X_train_TFIDF,  
    y_train  
)
```

Out[61]:

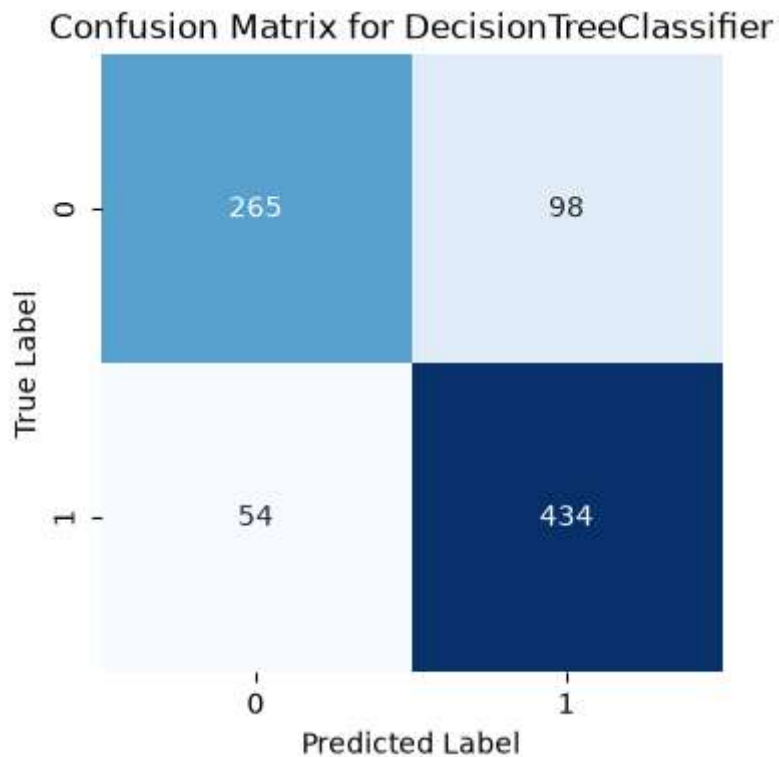
- GridSearchCV ⓘ ?
- best_estimator_: DecisionTreeClassifier
 - DecisionTreeClassifier ?



```
In [62]: # Get the best estimator from grid search  
best_DT = grid_search.best_estimator_
```

```
In [63]: # Make predictions using the best model  
y_pred = best_DT.predict(X_test_TFIDF)
```

```
In [64]: # Evaluate the model  
evaluation_results = evaluate_model(  
    'DecisionTreeClassifier',  
    y_test,  
    y_pred  
)
```



```
In [65]: # Print the evaluation results
for key, value in evaluation_results.items():

    if key == 'Classification Report':
        print(value)

    else:
        print(
            f"{key}: {value:.4f}"
            if isinstance(value, float)
            else f"{key}: \n{value}"
        )
```

Model Name:

DecisionTreeClassifier

Accuracy: 0.8214

Precision: 0.8222

Recall: 0.8214

F1 Score: 0.8195

	precision	recall	f1-score	support
0	0.83	0.73	0.78	363
1	0.82	0.89	0.85	488
accuracy			0.82	851
macro avg	0.82	0.81	0.81	851
weighted avg	0.82	0.82	0.82	851

```
In [66]: # Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

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

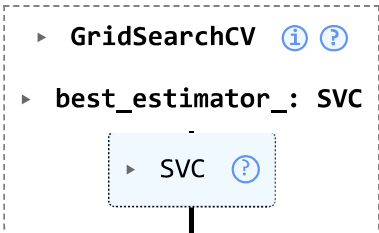
In [67]: `SVM = SVC(random_state=42)`

In [68]: `# Define the hyperparameters for grid search`
`param_grid = {`
 `'kernel': ['linear', 'rbf', 'poly']`
`}`

In [69]: `# Initialize GridSearchCV`
`grid_search = GridSearchCV(`
 `estimator=SVM,`
 `param_grid=param_grid,`
 `cv=5,`
 `scoring='accuracy',`
 `n_jobs=-1`
`)`

In [70]: `# Fit the grid search to the data`
`grid_search.fit(`
 `X_train_TFIDF,`
 `y_train`
`)`

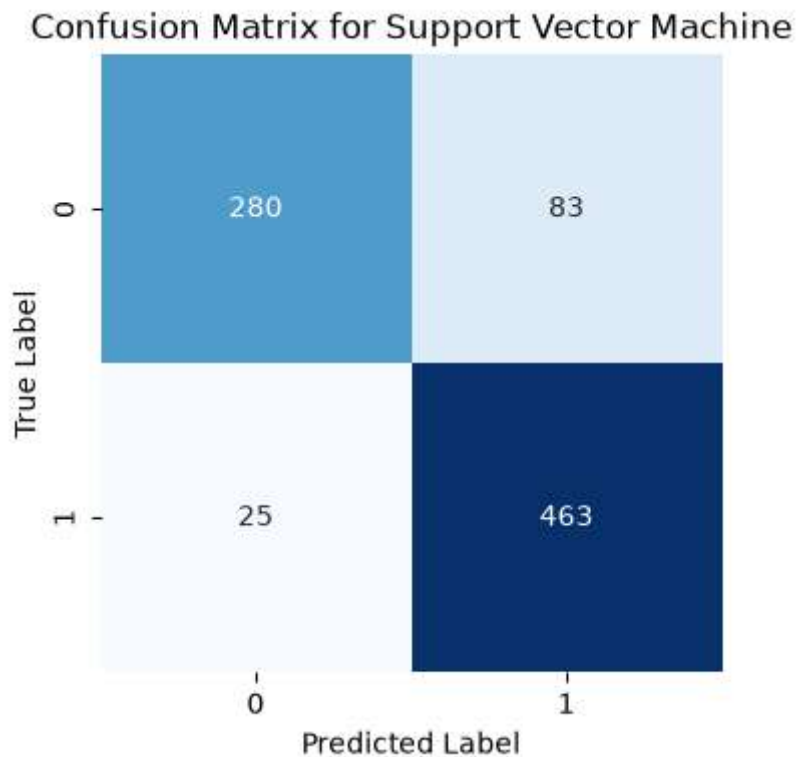
Out[70]:



In [71]: `# Get the best estimator from grid search`
`best_SVM = grid_search.best_estimator_`

In [72]: `# Make predictions using the best model`
`y_pred = best_SVM.predict(`
 `X_test_TFIDF`
`)`

In [73]: `# Evaluate the model`
`evaluation_results = evaluate_model(`
 `'Support Vector Machine',`
 `y_test,`
 `y_pred`
`)`



```
In [74]: # Print the evaluation results
for key, value in evaluation_results.items():

    if key == 'Classification Report':
        print(value)

    else:
        print(
            f"{key}: {value:.4f}"
            if isinstance(value, float)
            else f"{key}: \n{value}"
        )
```

Model Name:

Support Vector Machine

Accuracy: 0.8731

Precision: 0.8779

Recall: 0.8731

F1 Score: 0.8711

	precision	recall	f1-score	support
0	0.92	0.77	0.84	363
1	0.85	0.95	0.90	488
accuracy			0.87	851
macro avg	0.88	0.86	0.87	851
weighted avg	0.88	0.87	0.87	851

```
In [75]: # Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

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

In []: