

Practical_Image_Processing_and_Natural_La

Project 2

Student: Sara Ismail



<https://www.kaggle.com/datasets/abdallahwagih/emotion-dataset>

```
In [4]: # Import necessary Libraries
import pandas as pd
import numpy as np
import re

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, reca

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

import matplotlib.pyplot as plt
import seaborn as sns

# Reading dataset
from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv("/content/drive/MyDrive/Emotion_classify_Data.csv")
df.head()
```

Mounted at /content/drive

```
Out[4]:
```

	Comment	Emotion
0	i seriously hate one subject to death but now ...	fear
1	im so full of life i feel appalled	anger
2	i sit here to write i start to dig out my feel...	fear
3	ive been really angry with r and i feel like a...	joy
4	i feel suspicious if there is no one outside l...	fear

```
In [5]: # Show columns names

df.columns
```

Out[5]: Index(['Comment', 'Emotion'], dtype='object')

In [6]: *# Select columns*

```
TEXT_COLUMN = "Comment"
LABEL_COLUMN = "Emotion"

df = df[[TEXT_COLUMN, LABEL_COLUMN]]

df.head()
```

Out[6]:

	Comment	Emotion
0	i seriously hate one subject to death but now ...	fear
1	im so full of life i feel appalled	anger
2	i sit here to write i start to dig out my feel...	fear
3	ive been really angry with r and i feel like a...	joy
4	i feel suspicious if there is no one outside l...	fear

In [7]: *# Function to evaluate model*

```
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 [8]: df.isnull().sum()
# Drop missing values
df = df.dropna()
# Check classes count
df["Emotion"].value_counts()

```

Out[8]:

	count
Emotion	
anger	2000
joy	2000
fear	1937

dtype: int64

```

In [9]: # Preprocessing text

def preprocess_text(text):

    # Convert text to lowercase
    text = text.lower()

    # Remove Links
    text = re.sub(r'http\S+|www\S+', '', text)

    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Remove punctuation and special characters
    text = re.sub(r'^\w\s', '', text)

    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()

    return text

```

```

In [10]: # Apply preprocessing

df["clean_text"] = df["Comment"].apply(preprocess_text)

df.head()

```

```
# Define X and y

X = df["clean_text"]
y = df["Emotion"]

# Split data into training and testing sets

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

# Feature representation
# Bag word

vectorizer1 = CountVectorizer()

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

# Random Forest Classifier

# 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_Bag_word, 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_Bag_word)

# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test, y_pred)
```

```

# 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{va

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

# LogisticRegression

# Define the Logistic Regression model
LR = LogisticRegression(random_state=42, max_iter=1000)

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

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

# Fit the grid search to the data
grid_search.fit(X_train_Bag_word, y_train)

# Get the best estimator from grid search
best_LR = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_LR.predict(X_test_Bag_word)

# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test, y_pred)

# 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{va

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

# Feature representation using TF-IDF with Unigrams, Bigrams, and Trigrams

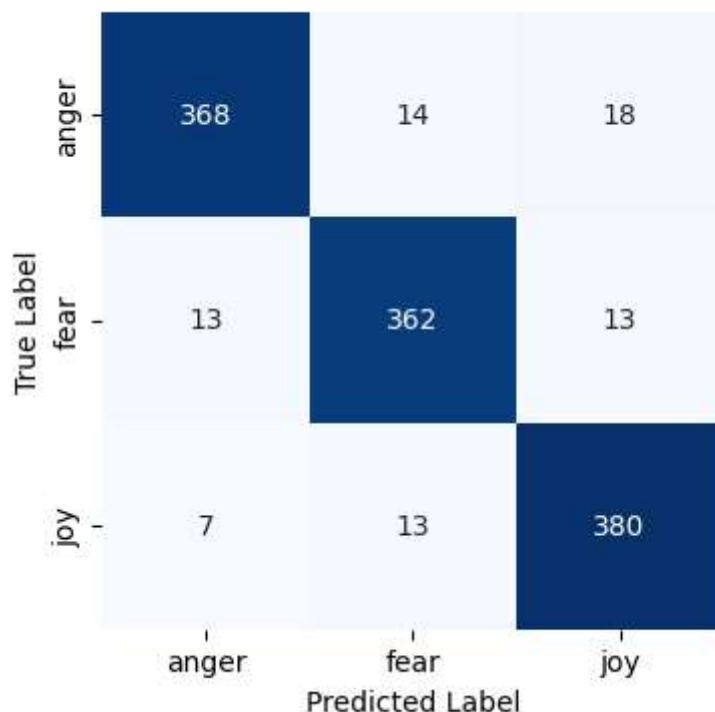
```

```
# Unigrams: "happy", "sad"
# Bigrams: "very happy", "not good"
# Trigrams: "i am happy"
# ngram_range=(1, 2) means Unigrams + Bigrams

vectorizer2 = TfidfVectorizer(ngram_range=(1, 2))

X_train_TFIDF = vectorizer2.fit_transform(X_train)
X_test_TFIDF = vectorizer2.transform(X_test)
```

Confusion Matrix for RandomForestClassifier



Model Name:

RandomForestClassifier

Accuracy: 0.9343

Precision: 0.9346

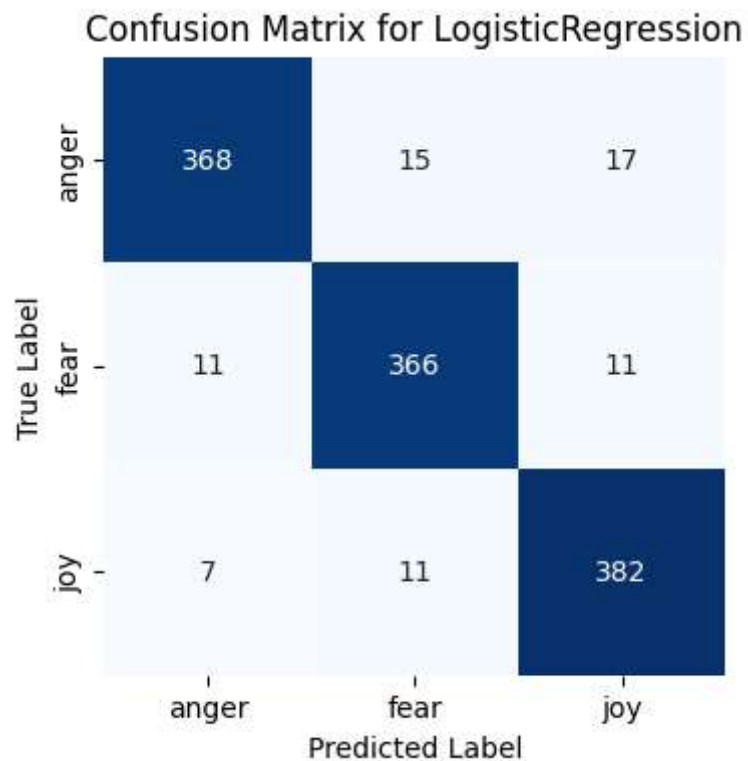
Recall: 0.9343

F1 Score: 0.9343

	precision	recall	f1-score	support
anger	0.95	0.92	0.93	400
fear	0.93	0.93	0.93	388
joy	0.92	0.95	0.94	400
accuracy			0.93	1188
macro avg	0.93	0.93	0.93	1188
weighted avg	0.93	0.93	0.93	1188

Best hyperparameters found by GridSearchCV:

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



Model Name:

LogisticRegression

Accuracy: 0.9394

Precision: 0.9396

Recall: 0.9394

F1 Score: 0.9394

	precision	recall	f1-score	support
anger	0.95	0.92	0.94	400
fear	0.93	0.94	0.94	388
joy	0.93	0.95	0.94	400
accuracy			0.94	1188
macro avg	0.94	0.94	0.94	1188
weighted avg	0.94	0.94	0.94	1188

Best hyperparameters found by GridSearchCV:

`{'solver': 'lbfgs'}`

```
In [11]: # DecisionTreeClassifier

# Define the Decision Tree model
DT = DecisionTreeClassifier(random_state=42)

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

# Initialize GridSearchCV
grid_search = GridSearchCV(
```

```

estimator=DT,
param_grid=param_grid,
cv=10,
scoring='accuracy',
n_jobs=-1
)

# Fit the grid search to the data
grid_search.fit(X_train_TFIDF, y_train)

# Get the best estimator from grid search
best_DT = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_DT.predict(X_test_TFIDF)

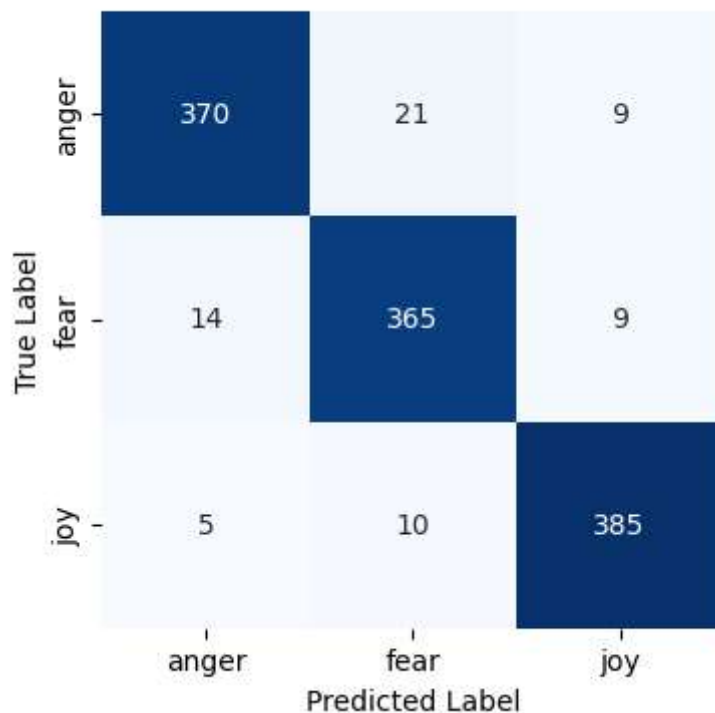
# Evaluate the model
evaluation_results = evaluate_model('DecisionTreeClassifier', y_test, y_pred)

# 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{va

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

```

Confusion Matrix for DecisionTreeClassifier



Model Name:

DecisionTreeClassifier

Accuracy: 0.9428

Precision: 0.9429

Recall: 0.9428

F1 Score: 0.9428

	precision	recall	f1-score	support
anger	0.95	0.93	0.94	400
fear	0.92	0.94	0.93	388
joy	0.96	0.96	0.96	400
accuracy			0.94	1188
macro avg	0.94	0.94	0.94	1188
weighted avg	0.94	0.94	0.94	1188

Best hyperparameters found by GridSearchCV:

{'criterion': 'gini', 'max_depth': None}

```
In [12]: # Support Vector Machine

SVM = SVC(random_state=42)

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

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

# Fit the grid search to the data
grid_search.fit(X_train_TFIDF, y_train)

# Get the best estimator from grid search
best_SVM = grid_search.best_estimator_

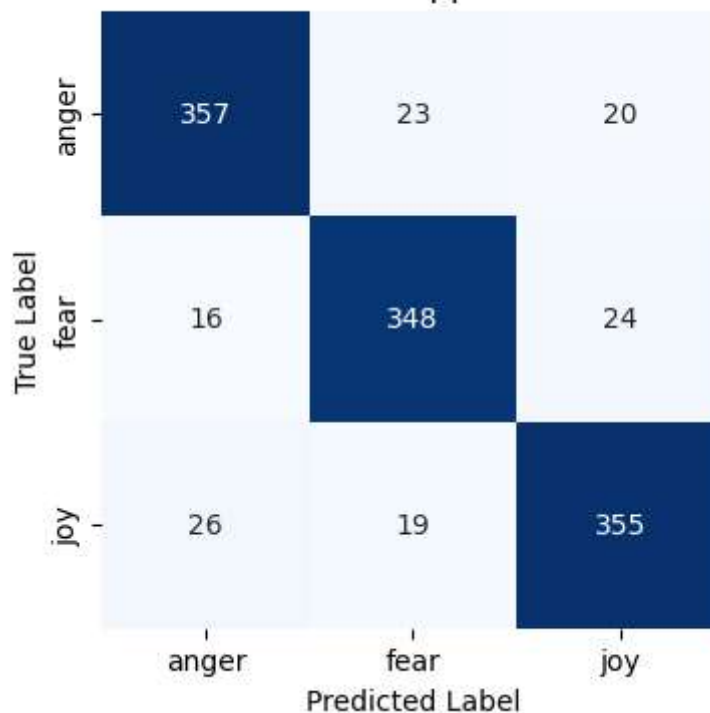
# Make predictions using the best model
y_pred = best_SVM.predict(X_test_TFIDF)

# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test, y_pred)

# 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{va
```

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

Confusion Matrix for Support Vector Machine



Model Name:

Support Vector Machine

Accuracy: 0.8923

Precision: 0.8923

Recall: 0.8923

F1 Score: 0.8923

	precision	recall	f1-score	support
anger	0.89	0.89	0.89	400
fear	0.89	0.90	0.89	388
joy	0.89	0.89	0.89	400
accuracy			0.89	1188
macro avg	0.89	0.89	0.89	1188
weighted avg	0.89	0.89	0.89	1188

Best hyperparameters found by GridSearchCV:

```
{'kernel': 'linear'}
```