Addis Ababa University
አዲስ፡አበባ፡ዩኒቨርሲቲ
SEEK WISDOM, ELEVATE YOUR INTELLECT AND SERVE HUMANITY !

NATURAL LANGUAGE DETECTION

Emotin Detection in Text

JUNE 12, 2025

Name ID Number

1. SOLOMON AWOKE-----------------------------ATE/ 8291/13
2. YONAS BAYSASAW-----------------------------ATE/9495/13
3. MICKIAS WORKU -------------------------------ATE/2508/13
4. NATNAEL HENOK--------------------------------ATE/6792/13
5. YABISRA HAILU-----------------------------------ATE/7665/13
6. YIDIDIYA TAMIRAT ---------------------------- ATE/6282/13

Contents

1 Introduction

Emotion is a fundamental aspect of human communication, expressing our internal states, reactions, and intentions. While often conveyed through facial expressions, tone of voice, or body language, a significant portion of our emotional output is also embedded within the written word. From casual social media posts to formal reviews, text carries subtle and overt cues about the sender's feelings. This document explores the fascinating field of emotion detection from text, a crucial subdomain within Natural Language Processing (NLP) and Artificial Intelligence. The core idea is to enable computers to automatically identify and categorize the emotions expressed in written content. This process involves a series of sophisticated steps, mirroring how a human might infer feelings from words:

- Data Preparation: Cleaning raw text by removing irrelevant characters, symbols, and common words (stopwords) that lack significant emotional weight.
- Feature Extraction: Converting the cleaned text into a numerical format that machine learning models can understand. Techniques like counting word frequencies (as seen with CountVectorizer) transform words into quantifiable features.
- Model Training: Using these numerical representations to train algorithms (such as Logistic Regression or Naive Bayes) to learn patterns and associations between specific words or phrases and particular emotions (e.g., joy, sadness, anger).
- Prediction and Interpretation: Once trained, the model can analyze new, unseen text, predict the most likely emotion, and even provide insights into which words most strongly influenced that prediction.

The ability to automatically detect emotion from text has wide-ranging applications, from enhancing customer service by prioritizing emotionally charged feedback to improving sentiment analysis in market research, and even aiding in mental health support by identifying distress signals in written communication. This powerful capability allows us to uncover emotional insights from vast amounts of textual data, offering a deeper understanding of human sentiment in the digital age.

2 The main.py Python Program

This program acts like a "feeling detector" for written words, essentially serving as a special assistant who reads what you type and tells you what emotion those words express. The process begins with **your input**, where you simply type any sentence or phrase into the program, much like typing a message into a chat application. For example, you might type, "I feel so happy today, the sun is shining!" or "This is frustrating, I can't believe it." To stop the program at any time, you just type the word exit.

Once you've entered your text, the program immediately moves into its **processing phase** behind the scenes. This involves two key steps. First, it "tidies up" your words through a cleaning process, much like decluttering a messy room to find important clues. It automatically removes common, unimportant words (like "the," "is," "a," "I am") that don't carry significant emotional weight, and it strips away punctuation marks such as! Or. So, a sentence like "I am so happy today, everything is going well!" might be streamlined to focus on words like "happy today, everything is going well." Second, because computers only understand numbers, the program "translates" these cleaned words into numerical codes or scores. It uses a special internal "dictionary" that it built by learning from countless texts previously, assigning unique numerical values to the important words in your input. This crucial step is how the computer "reads" and "understands" the meaning embedded in your text.

Finally, after processing, the program presents you with its **output**. Having transformed your words into numbers, it then uses its "trained brain"—a sophisticated machine learning model—which has studied vast examples of text to learn which numerical patterns correspond to emotions like "joy," "sadness," "anger," or "surprise." It compares the numerical codes from your words to what it has learned and makes its best guess about the primary emotion expressed in your text. Alongside this predicted emotion (which it displays in capital letters, for example, JOY), it also provides a "Confidence" score. This score, a number between 0 and 1 (or 0% and 100%), indicates how sure the program is about its prediction, allowing you to gauge the reliability of its analysis.

```
PS D:\project\Emotion-Detection-of-Text> python .\main.py
Emotion detection model loaded successfully.
CountVectorizer loaded successfully.

--- Emotion Detection from Text ---
Enter text to detect its emotion (type 'exit' to quit).

Enter your text: I am so happy today, everything is going well.
Predicted Emotion: JOY
Confidence: 0.82

Enter your text: I feel so empty and lost, today has been truly awful.
Predicted Emotion: SADNESS
Confidence: 0.51

Enter your text: I'm absolutely furious! This is unacceptable and makes me so mad.
Predicted Emotion: ANGER
Confidence: 0.92

Enter your text: Wow, I can't believe it! That's totally unexpected!
Predicted Emotion: SURPRISE
Confidence: 0.53

Enter your text:
```

3 Setup and Imports

This section imports all necessary libraries for data manipulation, visualization, text processing, machine learning, and model interpretation.

Setup and Imports

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import neattext.functions as nfx
from textblob import TextBlob
from collections import Counter
from wordcloud import WordCloud
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, ConfusionMatrixDispla
from sklearn.model_selection import train_test_split
import joblib
import eli5
```

269] ✓ 0.0s Python

4 Data Loading and Initial Exploration

The dataset is loaded from a CSV file, and its basic characteristics are examined to understand the data structure and identify any initial issues.

4.1 Load Dataset

Loads the emotion_dataset_2.csv file into a pandas DataFrame.

```
df = pd.read_csv('data/emotion_dataset_2.csv')
```

4.2 Preview Data

Displays the first few rows of the DataFrame to get a glimpse of the data.

```
# preview
df.head()
```

✓ 0.1s Python

	Unnamed: 0	Emotion	Text	Clean_Text
0	0	neutral	Why ?	NaN
1	1	joy	Sage Act upgrade on my to do list for tommorow.	Sage Act upgrade list tommorow
2	2	sadness	ON THE WAY TO MY HOMEGIRL BABY FUNERAL!!! MAN ...	WAY HOMEGIRL BABY FUNERAL MAN HATE FUNERALS SH...
3	3	joy	Such an eye ! The true hazel eye-and so brill...	eye true hazel eyeand brilliant Regular feat...
4	4	joy	@lluvmiasantos ugh babe.. hugggzzz for u .! b...	ugh babe hugggzzz u babe naamazed nga ako e...

4.3 Dataset Shape

Shows the number of rows and columns in the dataset.

```
df.shape
```
✓ 0.0s Python

```
(34792, 4)
```

4.4 Data Types

Checks the data types of each column.

```
df.dtypes
```
✓ 0.0s Python

```
Unnamed: 0     int64
Emotion       object
Text          object
Clean_Text    object
dtype: object
```

4.5 Check for Missing Values

Identifies the count of missing values in each column.

```
df.isnull().sum()
```
✓ 0.0s Python

```
Unnamed: 0      0
Emotion         0
Text            0
Clean_Text    466
dtype: int64
```

4.6 Emotion Distribution

Calculates and displays the value counts for the 'Emotion' column, showing the distribution of different emotions in the dataset.

```
# value Counts of Emotions
df['Emotion'].value_counts()
```
✓ 0.0s Python

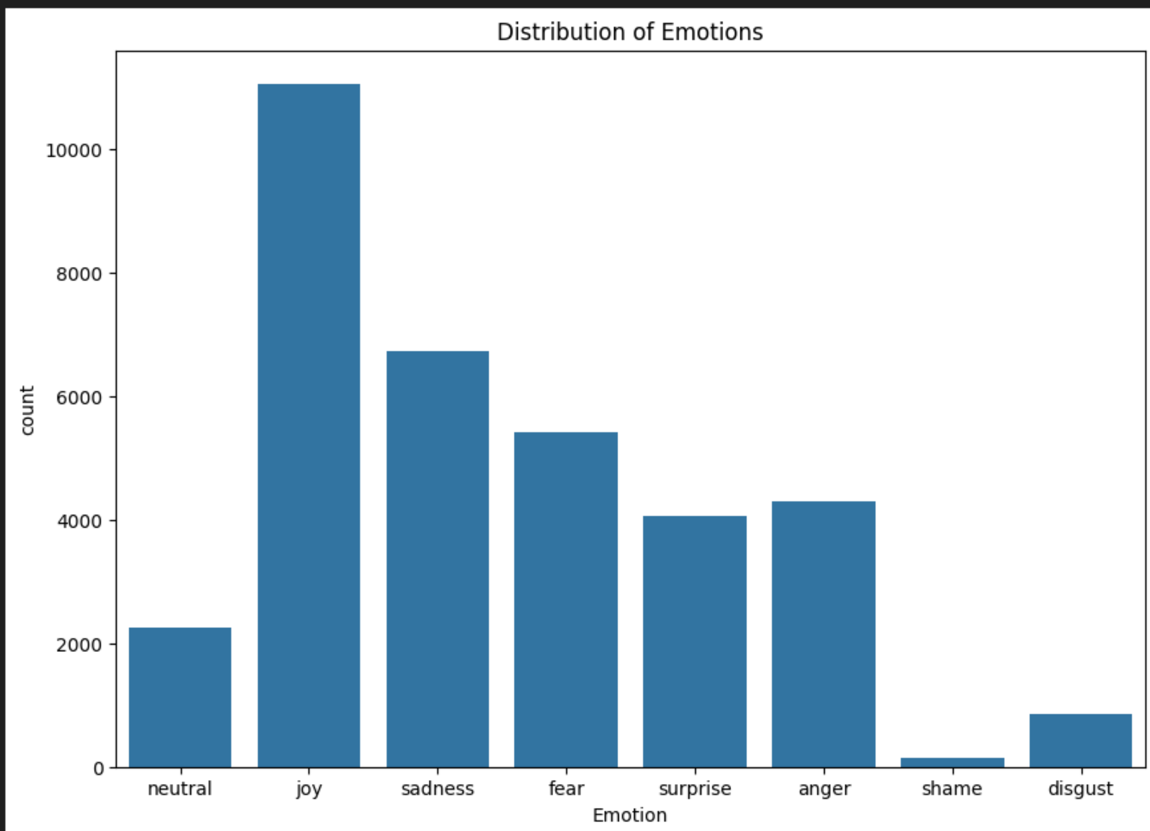
```
Emotion
joy          11045
sadness       6722
fear          5410
anger         4297
surprise      4062
neutral       2254
disgust        856
shame          146
Name: count, dtype: int64
```

4.7 Visualize Emotion Distribution

Creates a bar plot to represent the distribution of emotions visually.

```python
# Vallue Counts of Emotions
plt.figure(figsize=(10, 7))
sns.countplot(x='Emotion', data=df)
plt.title('Distribution of Emotions')
plt.show()
```

0.1s



5 Text Preprocessing

This section focuses on cleaning the 'Text' data by removing unwanted elements such as stopwords, special characters, and punctuation. A new column 'Clean_Text' is created to store the preprocessed text.

5.1 Text Cleaning Functions

Defines helper functions to clean the text using the NeatText library.

```python
# Function to remove stopwords
def remove_stopwords(text):
    return nfx.remove_stopwords(text)

# Function to remove special characters
def remove_special_characters(text):
    return nfx.remove_special_characters(text)

# Function to remove punctuations
def remove_punctuations(text):
    return nfx.remove_punctuations(text)
```
✓ 0.0s Python

5.2 Apply Cleaning to Text Column

Applies the defined cleaning functions to the 'Text' column to create 'Clean_Text'.

```python
df['Clean_Text'] = df['Text'].apply(remove_stopwords)
df['Clean_Text'] = df['Clean_Text'].apply(remove_special_characters)
df['Clean_Text'] = df['Clean_Text'].apply(remove_punctuations)
```
✓ 0.2s Python

5.3 Verify Cleaned Text

Displays the Data Frame head with the new 'Clean Text' column.

```python
df.head()
```
✓ 0.0s Python

	Unnamed: 0	Emotion	Text	Clean_Text
0	0	neutral	Why ?	
1	1	joy	Sage Act upgrade on my to do list for tommorow.	Sage Act upgrade list tommorow
2	2	sadness	ON THE WAY TO MY HOMEGIRL BABY FUNERAL!!! MAN ...	WAY HOMEGIRL BABY FUNERAL MAN HATE FUNERALS SH...
3	3	joy	Such an eye ! The true hazel eye-and so brill...	eye true hazel eyeand brilliant Regular feat...
4	4	joy	@lluvmiasantos ugh babe.. hugggzzz for u .! b...	lluvmiasantos ugh babe hugggzzz u babe naamaz...

6 Feature Engineering

Text data needs to be converted into numerical features for machine learning models. This section demonstrates the use of CountVectorizer and TfidfVectorizer for this purpose.

6.1 Keyword Extraction Function

A utility function to extract keywords from a given document using Counter.

```python
def extract_keywords(text_data, num_keywords=50):
    total_keywords = Counter()
    for text in text_data:
        if isinstance(text, str):
            words = text.split()
            total_keywords.update(words)
    return total_keywords.most_common(num_keywords)
```
✓ 0.0s Python

6.2 Most Common Joy Keywords

Extracts and displays the most common keywords for the 'joy' emotion.

```python
joy_list = df[df['Emotion'] == 'joy']['Clean_Text'].tolist()
joy_docx = ' '.join([str(item) for item in joy_list])
keyword_joy = extract_keywords(joy_docx)
keyword_joy
```

✓ 0.3s Python

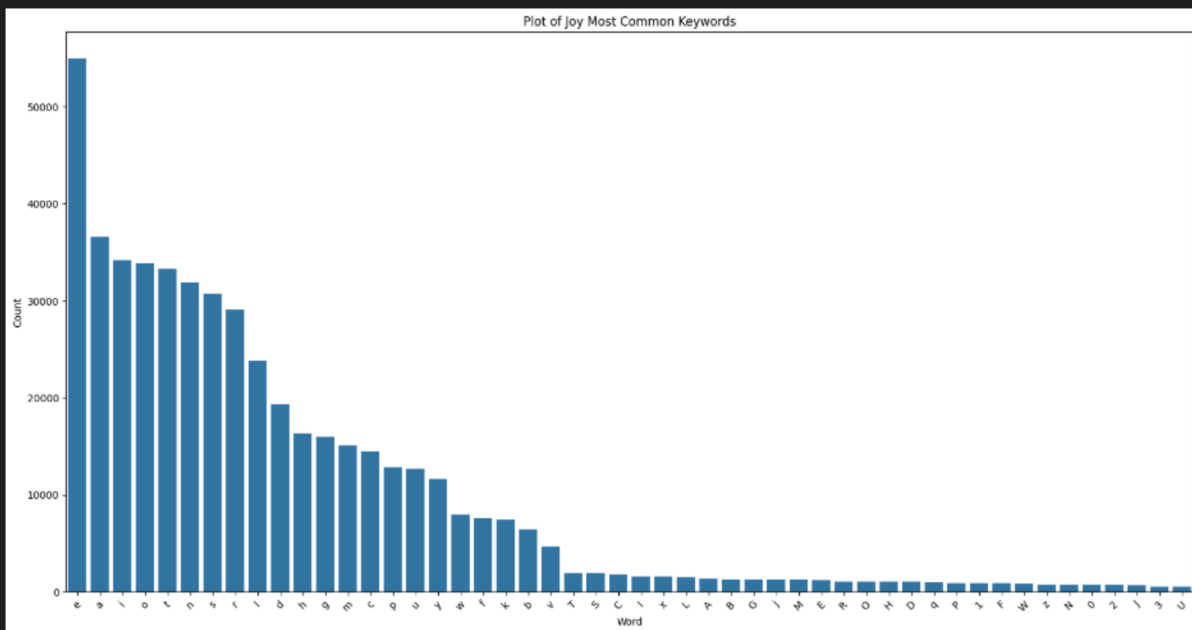
```
[('e', 54972),
 ('a', 36552),
 ('i', 34188),
 ('o', 33861),
 ('t', 33323),
 ('n', 31890),
 ('s', 30733),
 ('r', 29093),
 ('l', 23842),
 ('d', 19341),
 ('h', 16348),
 ('g', 15939),
 ('m', 15091),
 ('c', 14442),
 ('p', 12795),
 ('u', 12661),
 ('y', 11659),
 ('w', 7985),
 ('f', 7604),
 ('k', 7433),
 ('b', 6431),
 ('v', 4648),
 ('T', 1984),
 ('S', 1969),
 ('C', 1793),
...
 ('0', 759),
 ('2', 759),
 ('J', 673),
 ('3', 546),
 ('U', 540)]
```

6.3 Plot Most Common Words

A function to visualize the most common keywords for any given emotion.

```python
def plot_most_common_words(mydict, emotion_name):
    df_01 = pd.DataFrame(mydict, columns=['Word', 'Count'])
    plt.figure(figsize=(20, 10))
    plt.title("Plot of {} Most Common Keywords".format(emotion_name))
    sns.barplot(x='Word', y='Count', data=df_01)
    plt.xticks(rotation=45)
    plt.show()

# Example usage for joy keywords
plot_most_common_words(keyword_joy, 'Joy')
```

✓ 0.6s



6.4 Word Cloud Visualization

Generates and displays word clouds for different emotions to visually represent the most frequently used words.

```python
# For 'joy' emotion
plt.figure(figsize=(20, 10))
wordcloud = WordCloud(max_words=100, background_color="white").generate(joy_docx)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Joy Emotion Word Cloud")
plt.show()
```

✓ 0.7s Python

6.5 Prepare Data for Modeling

Splits the data into features (X) and target (y), and then into training and testing sets.

```python
Xfeatures = df['Clean_Text']
ylabels = df['Emotion']

# Split data
X_train, X_test, y_train, y_test = train_test_split(Xfeatures, ylabels, test_size=0.3, random_state=42)
```

✓ 0.0s Python

6.6 Feature Extraction using CountVectorizer

Converts the text data into a matrix of token counts.

```python
cv = CountVectorizer()
X_train_cv = cv.fit_transform(X_train.astype('U')) # Use .astype('U') to handle mixed types
X_test_cv = cv.transform(X_test.astype('U'))
```

✓ 0.3s Python

7 Model Training and Evaluation

This section trains two common machine learning models, Logistic Regression and Multinomial Naive Bayes, on the vectorized text data and evaluates their performance.

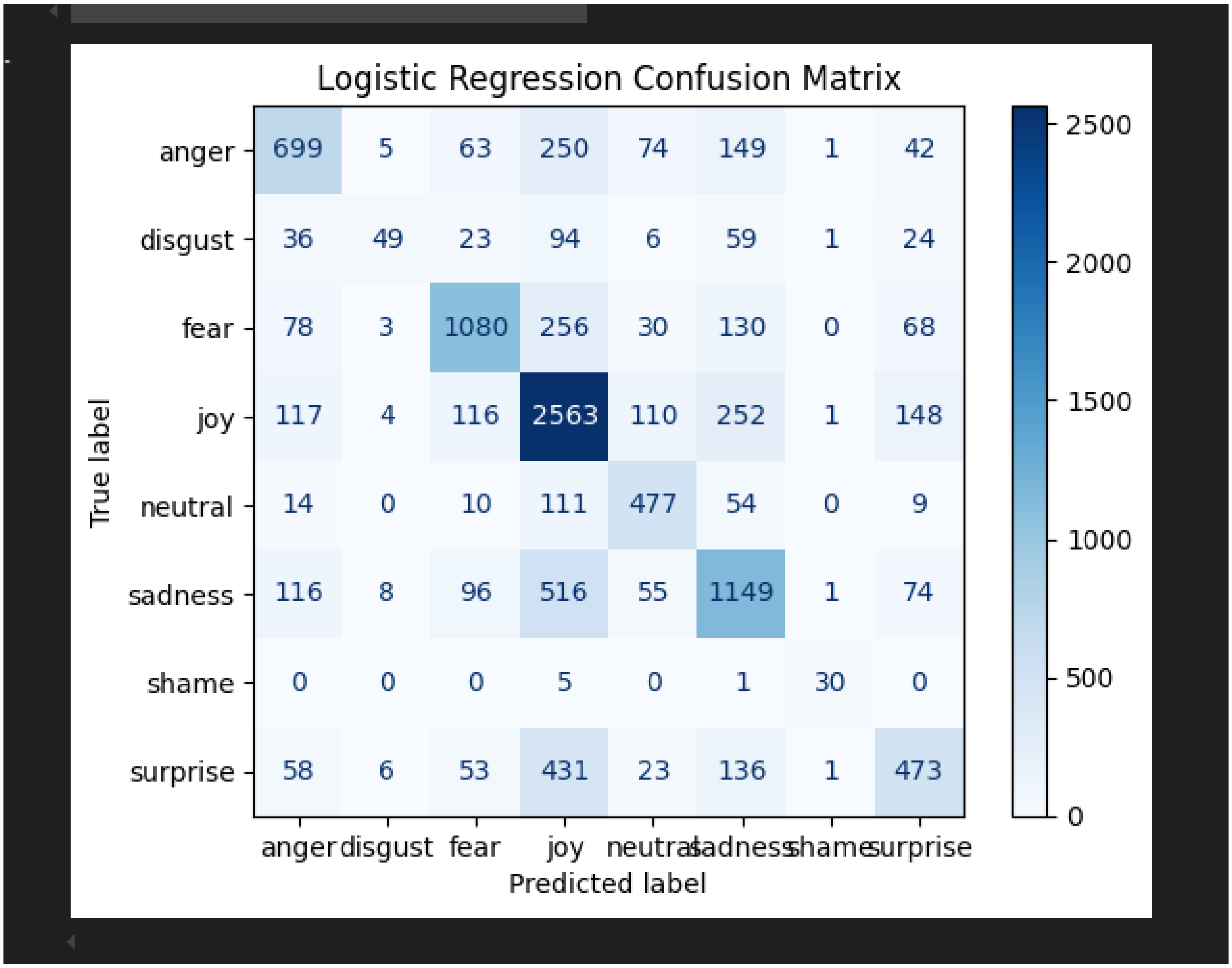
7.1 Logistic Regression Model

Trains a Logistic Regression model and evaluates its performance.

```python
lr_model = LogisticRegression(solver='liblinear', max_iter=1000)
lr_model.fit(X_train_cv, y_train)
lr_predictions = lr_model.predict(X_test_cv)

print("Logistic Regression Accuracy:", accuracy_score(y_test, lr_predictions))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, lr_predictions))
print("\nClassification Report:\n", classification_report(y_test, lr_predictions))

# Display Confusion Matrix
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, lr_predictions), display_labels=lr_model.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title('Logistic Regression Confusion Matrix')
plt.show()
```

✓ 1.2s Python

```
C:\Users\Yonas\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\si
  warnings.warn(
Logistic Regression Accuracy: 0.6246407357731366

Confusion Matrix:
 [[ 699    5   63  250   74  149    1   42]
 [  36   49   23   94    6   59    1   24]
 [  78    3 1080  256   30  130    0   68]
 [ 117    4  116 2563  110  252    1  148]
 [  14    0   10  111  477   54    0    9]
 [ 116    8   96  516   55 1149    1   74]
 [   0    0    0    5    0    1   30    0]
 [  58    6   53  431   23  136    1  473]]

Classification Report:
               precision    recall  f1-score   support

       anger       0.63      0.54      0.58      1283
     disgust       0.65      0.17      0.27       292
        fear       0.75      0.66      0.70      1645
         joy       0.61      0.77      0.68      3311
     neutral       0.62      0.71      0.66       675
     sadness       0.60      0.57      0.58      2015
       shame       0.86      0.83      0.85        36
    surprise       0.56      0.40      0.47      1181

    accuracy                           0.62     10438
   macro avg       0.66      0.58      0.60     10438
weighted avg       0.63      0.62      0.62     10438
```



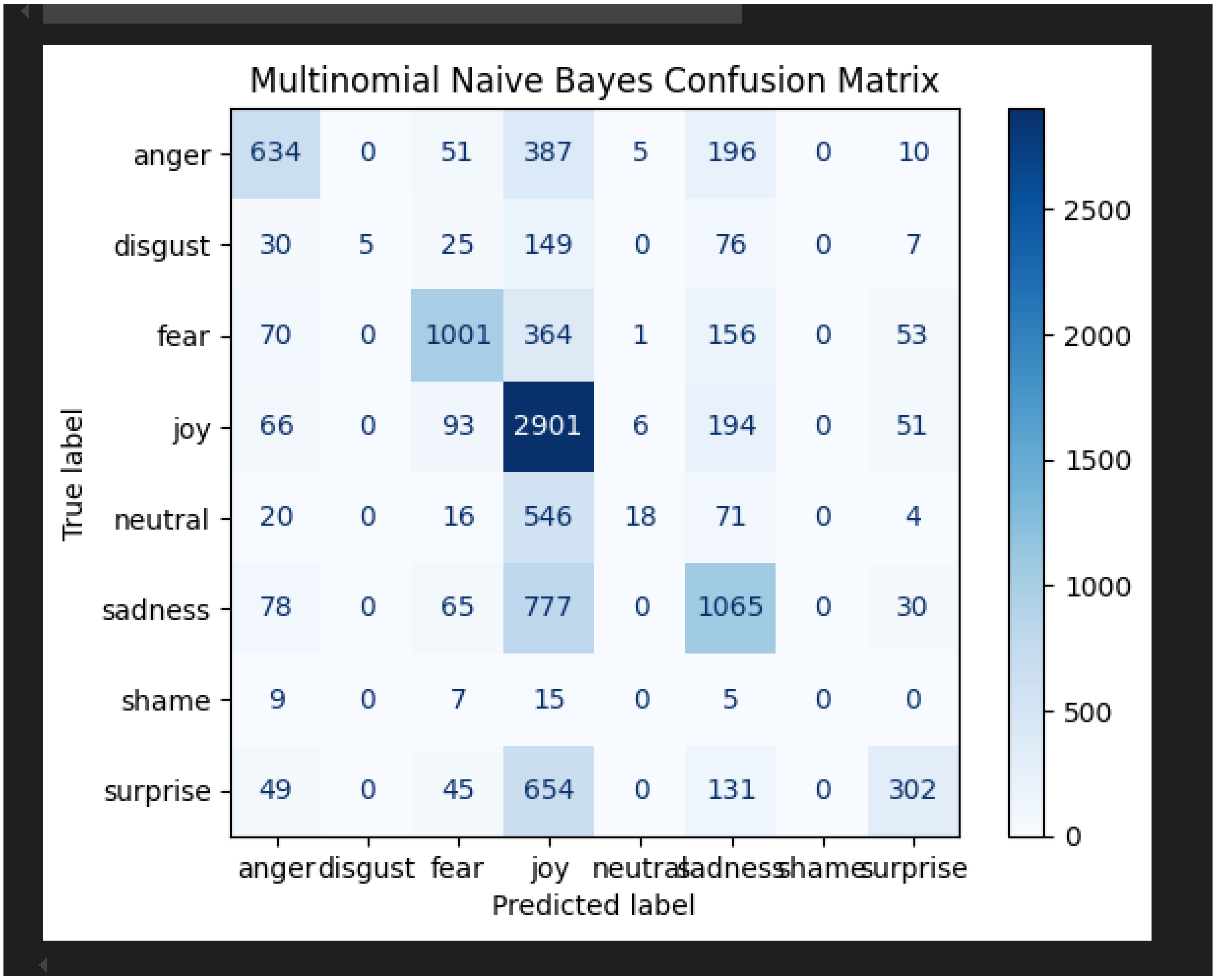
7.2 Multinomial Naive Bayes Model

Trains a Multinomial Naive Bayes model and evaluates its performance.

```python
nv_model = MultinomialNB()
nv_model.fit(X_train_cv, y_train)
nv_predictions = nv_model.predict(X_test_cv)

print("Naive Bayes Accuracy:", accuracy_score(y_test, nv_predictions))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, nv_predictions))
print("\nClassification Report:\n", classification_report(y_test, nv_predictions))

# Display Confusion Matrix
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, nv_predictions), display_labels=nv_model.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title('Multinomial Naive Bayes Confusion Matrix')
plt.show()
```

✓ 0.2s Python

```
Naive Bayes Accuracy: 0.5677332822379766

Confusion Matrix:
 [[ 634    0   51  387    5  196    0   10]
 [  30    5   25  149    0   76    0    7]
 [  70    0 1001  364    1  156    0   53]
 [  66    0   93 2901    6  194    0   51]
 [  20    0   16  546   18   71    0    4]
 [  78    0   65  777    0 1065    0   30]
 [   9    0    7   15    0    5    0    0]
 [  49    0   45  654    0  131    0  302]]

Classification Report:
               precision    recall  f1-score   support

       anger       0.66      0.49      0.57      1283
     disgust       1.00      0.02      0.03       292
        fear       0.77      0.61      0.68      1645
         joy       0.50      0.88      0.64      3311
     neutral       0.60      0.03      0.05       675
     sadness       0.56      0.53      0.54      2015
       shame       0.00      0.00      0.00        36
    surprise       0.66      0.26      0.37      1181

    accuracy                           0.57     10438
   macro avg       0.59      0.35      0.36     10438
weighted avg       0.61      0.57      0.53     10438

C:\Users\Yonas\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\si
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\Yonas\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\si
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\Yonas\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\si
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```



8 Emotion Prediction and Explainability

This section provides functions to predict emotion for new text and uses eli5 to explain model predictions.

8.1 Emotion Prediction Function

A function to preprocess new text and predict its emotion using the trained Logistic Regression model.

```python
def predict_emotion(text):
    clean_text = remove_stopwords(text)
    clean_text = remove_special_characters(clean_text)
    clean_text = remove_punctuations(clean_text)
    vectorized_text = cv.transform([clean_text])
    prediction = lr_model.predict(vectorized_text)[0]
    prediction_proba = lr_model.predict_proba(vectorized_text).max()
    return prediction, prediction_proba
```

✓ 0.0s Python

```python
text1 = "This is a good day"
emotion, prob = predict_emotion(text1)
print(f"Text: '{text1}' => Emotion: {emotion}, Probability: {prob:.2f}")
```

✓ 0.0s Python

```
Text: 'This is a good day' => Emotion: joy, Probability: 0.53
```

8.2 Model Explainability with ELI5

Uses ELI5 to show the weights of features for a given prediction, helping to understand which words contribute most to a particular emotion.

```python
# Assuming lr_model and cv (CountVectorizer) are already defined and fitted

# Example text for explanation
text_to_explain = "I am so happy today, everything is going well."

# Transform the text using the fitted CountVectorizer
vec_text = cv.transform([text_to_explain])

# Show weights for Logistic Regression model
eli5.show_weights(lr_model, vec=cv, top=(20, 5))
```

Python

y=anger top features		**y=disgust** top features	
Weight?	Feature	Weight?	Feature
+4.629	exasperation	+4.660	disgruntled
+4.350	indignant	+4.474	revulsion
+4.205	resentful	+2.680	fedup
+4.087	annoyance	+2.464	revolted
+3.967	anger	+2.284	nasty
+3.939	furious	+1.952	smell
+3.720	exasperated	+1.827	gross
+3.709	angry	+1.760	disgruntlement
+3.663	miffed	+1.738	ppl
+3.492	offended	+1.684	teeth
+3.481	infuriated	+1.648	ugh
+3.375	peeved	+1.623	smelling
+3.283	livid	+1.622	girls
+3.066	vexed	+1.596	public
+2.873	rage	+1.556	fat
+2.798	cross	+1.527	male
+2.721	mad	+1.503	nigga
+2.542	enraged	+1.501	respect
+2.468	insulted	+1.478	eww
+2.391	accused	+1.468	why
… 5631 more positive …		… 2363 more positive …	
… 26989 more negative …		… 30257 more negative …	
-1.823	frightened	-1.066	nice
-1.925	tomorrow	-1.144	scared
-2.109	scared	-1.201	thought
-2.407	<BIAS>	-1.636	tomorrow
-2.512	afraid	-4.233	<BIAS>

y=fear top features		**y=joy** top features	
Weight?	Feature	Weight?	Feature
+4.923	dismay	+3.940	delighted
+4.333	agitated	+3.743	delight
+4.162	nervous	+3.556	joy
+4.075	disquiet	+3.362	elation
+4.048	anxious	+3.261	ljb
+4.037	frightened	+3.155	pleased
+3.999	horrified	+3.006	amusement
+3.726	concerned	+3.001	ecstatic
+3.721	horror	+2.936	thebodyshopuk
+3.712	factor	+2.866	overjoyed
+3.693	scared	+2.856	elated
+3.439	fear	+2.671	steal
+3.237	officialpwg	+2.593	excitement
+3.232	alarmed	+2.566	thrilled
+3.215	afraid	+2.404	contented
+2.978	feared	+2.263	sarcasm
+2.936	terrified	+2.035	amused
+2.761	snakes	+2.019	homework
+2.689	threatened	+1.944	happiness
+2.629	drake	+1.853	joyful
… 7185 more positive …		… 11191 more positive …	
… 25435 more negative …		… 21429 more negative …	
-1.906	anger	-2.266	afraid
-1.917	delighted	-2.375	anger
-2.033	sad	-2.384	fear
-2.432	joy	-2.472	sad
-2.588	<BIAS>	-2.490	died

y=neutral top features		**y=sadness** top features	
Weight?	Feature	Weight?	Feature
+2.146	hi	+4.127	anguish
+1.754	veronica	+3.752	sad
+1.716	yes	+3.556	despondent
+1.561	hello	+3.482	grief
+1.487	ok	+3.471	miserable
+1.390	hobbies	+3.420	depressed
+1.337	teenagers	+3.327	heartbroken
+1.326	problem	+3.233	devastated
+1.293	stunts	+3.160	anguished
+1.279	excuse	+3.160	despair
+1.263	bars	+3.155	disconsolate
+1.235	sixties	+3.137	desolate
+1.234	sure	+3.114	mournful
+1.224	june	+3.105	downcast
+1.194	universe	+3.069	despondency
+1.181	yuan	+3.062	inconsolable
+1.172	whats	+2.899	died
+1.153	certainly	+2.836	sorrow
+1.136	suggest	+2.675	dejected
+1.129	convenient	+2.592	crestfallen
… 908 more positive …		… 7856 more positive …	
… 31712 more negative …		… 24764 more negative …	
-2.097	sad	-1.901	angry
-2.163	amp	-1.964	frightened
-2.231	christmas	-2.046	less
-2.232	me	-2.423	joy
-2.254	hate	-2.471	fear

y=shame top features		**y=surprise** top features	
Weight?	Feature	Weight?	Feature
+6.793	embarrassed	+4.149	astonishment
+6.038	ashamed	+4.083	bewildered
+5.555	embarrassment	+3.836	bewilderment
+3.069	mortified	+3.373	stunned
+2.799	humiliated	+3.203	startled
+2.322	discomfited	+3.134	puzzlement
+1.232	nt	+3.010	surprise
+0.948	real	+2.945	astonished
+0.928	expression	+2.920	astounded
+0.843	bishop	+2.321	nonplussed
+0.782	re	+2.142	van
+0.757	leave	+2.123	actually
+0.752	cushioned	+1.989	greysonchance
+0.747	proud	+1.899	surprised
+0.698	celia	+1.777	turns
+0.694	slightly	+1.720	gift
+0.674	took	+1.707	would
+0.663	scream	+1.636	mama
+0.642	hoot	+1.622	birthday
+0.630	drink	+1.603	flabbergasted
… 506 more positive …		… 8014 more positive …	
… 32114 more negative …		… 24606 more negative …	
-0.661	delighted	-1.749	sorry
-0.668	like	-1.879	sad
-0.766	amp	-1.994	kinder
-0.951	im	-2.514	afraid
-6.171	<BIAS>	-2.617	<BIAS>

9 Sentiment Analysis (TextBlob)

This section briefly demonstrates basic sentiment analysis using the TextBlob library.

9.1 TextBlob Sentiment Function

```python
def get_text_sentiment(text):
    return TextBlob(text).sentiment.polarity
```

Python

```python
# Example Usage
text = "This is a very good movie."
sentiment = get_text_sentiment(text)
print(f"Text: '{text}' => Sentiment: {sentiment}")
```

✓ 0.0s

Python

```
Text: 'This is a very good movie.' => Sentiment: 0.9099999999999999
```

10 Model Saving

This section shows how to save the trained model and vectorizer for future use.

```python
# Save the model
joblib.dump(lr_model, 'emotion_detector_model.joblib')

# Save the CountVectorizer
joblib.dump(cv, 'count_vectorizer.joblib')

# To load them later:
# loaded_model = joblib.load('emotion_detector_model.joblib')
# loaded_vectorizer = joblib.load('count_vectorizer.joblib')
```

Python

```
['count_vectorizer.joblib']
```