American International University-Bangladesh (AIUB)

Department of Computer Science & Engineering
Faculty of Science & Technology (FST)

Course: Natural Language Processing

Course Instructor**: DR. ABDUS SALAM**

Submitted By:

Section: A		**Group No: 4**
SL	Student Name	Student ID
1	**MAHIN MONTASIR AFIF**	22-46573-1
2	**MD. RIFAT KHAN**	22-46060-1
3	**A.H.M TANVIR**	22-47034-1
4	**HASIN ALMAS SIFAT**	22-48679-3

Submission Date: 26 April 2025

Dataset Description

The dataset used in this project is titled "Judge Emotion About Brands and Products." It contains 9093 rows of tweets where contributors evaluated whether each tweet expresses a positive, negative, or no emotion towards a brand and/or product. If emotion is present, the annotators were also asked to identify the target brand or product. This dataset serves as a basis for developing a model capable of detecting emotional expressions directed at brands.

Dataset -> https://www.kaggle.com/datasets/drishtiagarwal20/brands-and-product-emotions

Importing Necessary Libraries

The project begins by importing essential Python libraries such as pandas, numpy, matplotlib, scikit-learn, and nltk. These libraries enable data loading, preprocessing, feature extraction, model training, evaluation, and visualization.

The code snippet used for importing is:

```
import pandas as pd
import string
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, ENGLISH_STOP_WORDS
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
```

Loading the Dataset

The dataset is loaded with a CSV file containing tweets and their associated sentiment labels. Since tweets may contain special characters, the dataset is read using 'ISO-8859-1' encoding to avoid errors.

The code for loading the dataset is:

```
file_path = '/kaggle/input/brands-and-product-emotions/judge-1377884607_tweet_product_company.csv'
data = pd.read_csv(file_path, encoding='ISO-8859-1')
data.head()
```

Output:

	tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand_or_product
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative emotion
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive emotion
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive emotion
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive emotion

Handling Missing Values

To ensure a clean dataset, missing values from critical columns like tweet_text, is_there_an_emotion_directed_at_a_brand_or_product, and emotion_in_tweet_is_directed_at are removed using the dropna() function. This prevents issues during model training.

The corresponding code is:

```
data_cleaned = data.dropna(subset=['tweet_text', 'is_there_an_emotion_directed_at_a_brand_or_product'])
data_cleaned = data_cleaned.dropna(subset=['emotion_in_tweet_is_directed_at'])
data_cleaned.isnull().sum()
```

Output:

	0
tweet_text	0
emotion_in_tweet_is_directed_at	0
is_there_an_emotion_directed_at_a_brand_or_product	0

dtype: int64

Counting Total Number of Sentiments

Next, the sentiment distribution is analyzed to understand class balance. The value_counts() method is used to count the number of positive, negative, neutral, and ambiguous sentiment labels.

The code snippet used for this section is:

```
sentiment_counts = data_cleaned['is_there_an_emotion_directed_at_a_brand_or_product'].value_counts()
print(sentiment_counts)
```

Output:

```
is_there_an_emotion_directed_at_a_brand_or_product
Positive emotion                       2672
Negative emotion                        519
No emotion toward brand or product       91
I can't tell                              9
Name: count, dtype: int64
```

Text Preprocessing – Lowercasing

Text preprocessing is performed to prepare the tweets for machine learning. First, the text is converted to lowercase to maintain uniformity.

The code snippet used for this section:

```
def to_lowercase(text):
    return text.lower()
data_cleaned['cleaned_tweet_text'] = data_cleaned['tweet_text'].apply(to_lowercase)
```

Text Preprocessing – Removing Punctuation

Then, punctuation is removed to eliminate noise from the text. Each character is checked against string.punctuation, and only non-punctuation characters are retained.

Code snippet:

```
def remove_punctuation(text):
    return ''.join([char for char in text if char not in string.punctuation])

data_cleaned['cleaned_tweet_text']=data_cleaned['cleaned_tweet_text'].apply(remove_punctuation)
```

Text Preprocessing – Removing Stop words

After removing punctuation, English stopwords are eliminated using ENGLISH_STOP_WORDS from sklearn, ensuring that only meaningful words are kept for model training.

Code:

```
def remove_stopwords(text):
    return ' '.join([word for word in text.split() if word not in ENGLISH_STOP_WORDS])
data_cleaned['cleaned_tweet_text']=data_cleaned['cleaned_tweet_text'].apply(remove_stopwords)
```

Text Preprocessing – Lemmatizations

Finally, lemmatization is applied to normalize words to their base forms. The text is tokenized and lemmatized using WordNetLemmatizer from nltk, improving feature consistency.

Code:

```
lemmatizer = WordNetLemmatizer()
def lemmatize_text(text):
    tokens = word_tokenize(text)
    lemmatized = [lemmatizer.lemmatize(token) for token in tokens]
```

```
    return ' '.join(lemmatized)
data_cleaned['cleaned_tweet_text']=data_cleaned['cleaned_tweet_text'].apply(lemmatize_text
)
```

Splitting Data into Features and Target

The cleaned data is then prepared for model training by splitting it into features and target variables. The preprocessed tweet texts are assigned to X, and sentiment labels are assigned to y.

Code:

```
X = data_cleaned['cleaned_tweet_text']
y = data_cleaned['is_there_an_emotion_directed_at_a_brand_or_product']
```

Train-Test Split

The dataset is divided into training and testing sets using an 80-20 split, with a fixed random state for reproducibility. This step allows the model to learn from training data and be evaluated separately on unseen test data.

Code:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Text Vectorization: Converting Text to Features

Before feeding the data into the machine learning model, text data is converted into numerical features using CountVectorizer, creating a bag-of-words representation.

Code:

```
vectorizer = CountVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)
```

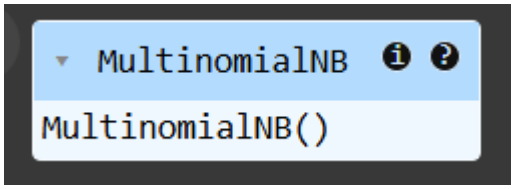
Model Training - Naive Bayes Classifier

For classification, a Multinomial Naive Bayes model is selected due to its effectiveness in text classification tasks. The model is trained on the vectorized training data.

Code:

```
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_vectorized, y_train)
```

Output:



Model Evaluation

The model's performance is evaluated by predicting sentiments on the test set and calculating the accuracy and a detailed classification report, including precision, recall, and F1-score for each sentiment category.

Code:

```
y_pred = nb_classifier.predict(X_test_vectorized)
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
print(accuracy)
print(classification_rep)
```

Output:

```
0.849772382397572
                                   precision    recall  f1-score   support

                      I can't tell       0.00      0.00      0.00         1
                  Negative emotion       0.79      0.30      0.43       105
No emotion toward brand or product       0.00      0.00      0.00        18
                  Positive emotion       0.85      0.99      0.92       535

                          accuracy                           0.85       659
                         macro avg       0.41      0.32      0.34       659
                      weighted avg       0.82      0.85      0.81       659
```

Confusion Matrix

Finally, a confusion matrix is created to visually assess the model’s predictions against true labels. The confusion matrix is plotted using matplotlib with annotated cell values, colored according to the number of samples, improving interpretability. Most accurate prediction: The model is most accurate in predicting "Positive emotion", with 529 samples correctly classified under this category. 31 instances of Negative emotion were incorrectly predicted as Negative emotion. 74 instances of Negative emotion were predicted as Positive emotion. There are a few misclassifications with other emotions, but overall, the majority of correct predictions are for Positive emotion.

Code:

```
cm = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize=(8, 6))
```

```
im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
ax.figure.colorbar(im, ax=ax)
ax.set(
    xticks=np.arange(cm.shape[1]),
    yticks=np.arange(cm.shape[0]),
    xticklabels=nb_classifier.classes_,
    yticklabels=nb_classifier.classes_,
    title='Confusion Matrix',
    ylabel='True Label',
    xlabel='Predicted'
)
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")
fmt = 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(
            j, i, format(cm[i, j], fmt),
            ha="center", va="center",
            color="white" if cm[i, j] > thresh else "black"
        )
fig.tight_layout()
plt.show()
```

Output: