TASK 4
Sentimation analysis

build a model that can analyze the sentiment of text data ,
such as customer reviews or socila media posts, use
techniques like bag-of-words,word embeddings ,or transformers to
classify text as positive , nagative mor natural sentimation

COUDE : GITHUB

IGNITE INTERN E MAIL:

LINKEDIN

Task 4 : ***Sentiment Analysis***

build a model that can analyze the sentiment of text data , such as customer reviews or socila media posts, use techniques like bag-of-words,word embeddings ,or transformers to classify text as positive , nagative mor natural sentimation

1. Bag-of-Words (BoW)

Concept:

- BoW is a simple and traditional method in Natural Language Processing (NLP). It represents text as a collection of words, disregarding grammar and word order but keeping multiplicity.

Process:

1. Text Preprocessing: Tokenize the text into words, remove stop words, and possibly apply stemming or lemmatization.

2. Vocabulary Creation: Create a vocabulary of all unique words in the corpus.

3. Vectorization: Convert each document into a vector where each element represents the frequency (or presence) of a word from the vocabulary.

Example:

- Text: "I love this movie. It is great."
- Vocabulary: ["I", "love", "this", "movie", "it", "is", "great"]
- Vector: [1, 1, 1, 1, 1, 1, 1]

2. Word Embeddings

Concept:

- Word embeddings map words to vectors of real numbers in a high-dimensional space, capturing semantic relationships between words.

Popular Models:

- Word2Vec: Generates embeddings by predicting context words given a target word (Skip-gram) or predicting a target word given context words (CBOW).
- GloVe: Generates embeddings by factorizing a word co-occurrence matrix.
- FastText: Extends Word2Vec by representing words as n-grams of characters, capturing subword information.

Process:

1. Text Preprocessing: Similar to BoW. 2. Embedding Lookup: Convert words into vectors using pre-trained embeddings or train embeddings on your own corpus.

3. Vector Representation: Represent each document as the average (or some other aggregate) of its word vectors.

Example:

- Text: "I love this movie."
- Embedding for "love": [0.2, 0.3, ..., 0.5]
- Document Vector: Average of vectors of "I", "love", "this", "movie".

3. Transformers

Concept:

- Transformers, especially models like BERT (Bidirectional Encoder Representations from Transformers), use self-attention mechanisms to capture the context of a word based on its surrounding words, handling long-range dependencies and polysemy effectively.

Popular Models:

- BERT: Pre-trained using masked language modeling and next sentence prediction tasks.

- GPT: Generative model pre-trained using autoregressive language modeling.

- RoBERTa, DistilBERT, T5, etc.: Variants and improvements over BERT.

Process:

1. Text Preprocessing: Tokenize text using the model’s tokenizer, adding special tokens (e.g., [CLS], [SEP] for BERT).

2. Input Representation: Convert tokens into embeddings, add positional embeddings, and create attention masks

3. Model Forward Pass: Pass the input through the transformer model to obtain contextual embeddings.

4. Classification: Use the [CLS] token embedding (for BERT-likemodels) or a pooled representation for classification tasks.

Example:

- Text: "I love this movie."

- Tokenized: ["[CLS]", "I", "love", "this", "movie", "

[SEP]"]

- Model Output: Contextual embeddings for each token.

- Use the embedding of "[CLS]" token for

classification.

1 simple program

In [2]:

```
# Import necessary libraries
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Example dataset
data = {
    'text': [
        'I  love  this  product!',  'This  is  the  worst
        thing  I  have  ever  bought.',  'It  is  okay,
        neither  good  nor  bad.',  'Absolutely  fantastic
        experience!',      'I      hate      it,      very
        disappointing.', 'Not great, not terrible.'

    ],
    'sentiment': ['positive', 'negative', 'neutral', 'positive', 'negative', 'ne
}

# Convert to DataFrame
df = pd.DataFrame(data)
df
```

Out[2]:

	text	sentiment
0	I love this product!	positive
1	This is the worst thing I have ever bought.	negative
2	It is okay, neither good nor bad.	neutral
3	Absolutely fantastic experience!	positive
4	I hate it, very disappointing.	negative
5	Not great, not terrible.	neutral

In [16]:

```
# Step 1: Text Preprocessing and Vectorization
# Create the Bag-of-Words model
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(df['text'])
print(X)
```

```
(0, 9)        1
(0, 11)       1
(1, 14)       1
(1, 13)       1
(1, 2)        1
(2, 10)       1
(2, 6)        1
(2, 1)        1
(3, 0)        1
(3, 5)        1
(3, 4)        1
(4, 8)        1
(4, 3)        1
(5, 7)        1
(5, 12)       1
```

In [17]:
```
# Encode the sentiment labels
y = df['sentiment'].map({'negative': 0, 'neutral': 1, 'positive': 2})
print(y)
```

```
0    2
1    0
2    1
3    2
4    0
5    1
Name: sentiment, dtype: int64
```

In [7]:
```
# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [18]:
```
print(X_train)
```

```
  (0, 7)	1
  (0, 12)	1
  (1, 10)	1
  (1, 6)	1
  (1, 1)	1
  (2, 8)	1
  (2, 3)	1
  (3, 0)	1
  (3, 5)	1
  (3, 4)	1
```

In [19]:
```
print(X_test)
```

```
  (0, 9)	1
  (0, 11)	1
  (1, 14)	1
  (1, 13)	1
  (1, 2)	1
```

In [20]:
```
print(y_train)
```

```
5    1
2    1
4    0
3    2
Name: sentiment, dtype: int64
```

In [21]:
```
print(y_test)
```

```
0    2
1    0
Name: sentiment, dtype: int64
```

In [26]:
```
# Step 3: Build and train the classifier (Logistic Regression)
classifier = LogisticRegression(max_iter=1000)
classifier.fit(X_train, y_train)
```

Out[26]:

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

In [27]:
```
# Step 4: Evaluate the model
y_pred = classifier.predict(X_test)
```

```
print(y_pred)
```

```
[1 1]
```

```
In [29]: print(classification_report(y_test, y_pred, target_names=['negative', 'neutral',
```

```
              precision    recall  f1-score   support

    negative       0.00      0.00      0.00       1.0
     neutral       0.00      0.00      0.00       0.0
    positive       0.00      0.00      0.00       1.0

    accuracy                           0.00       2.0
   macro avg       0.00      0.00      0.00       2.0
weighted avg       0.00      0.00      0.00       2.0
```

2nd Program

python necessary libraries

- pip install transformers
- pip install torch
- pip install datasets

```
In [ ]:
```

In [5]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
import gensim.downloader as api
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import torch
from transformers import BertTokenizer, BertForSequenceClassification, Trainer,
from datasets import Dataset
```

In [6]:
```
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Admin\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Admin\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[6]: True

In [9]:
```
data = {
    'text': [
        'I love this product!', 'This is the worst
        thing I have ever bought.', 'It is okay,
        neither good nor bad.', 'Absolutely fantastic
        experience!', 'I hate it, very
        disappointing.', 'Not great, not terrible.'

    ],
    'sentiment': ['positive', 'negative', 'neutral', 'positive', 'negative', 'ne
}

# Convert to DataFrame
df_data = pd.DataFrame(data)
df_data
```

Out[9]:

	text	sentiment
0	I love this product!	positive
1	This is the worst thing I have ever bought.	negative
2	It is okay, neither good nor bad.	neutral
3	Absolutely fantastic experience!	positive
4	I hate it, very disappointing.	negative
5	Not great, not terrible.	neutral

In [11]:
```
# Encode the sentiment labels
df_data['sentiment'] = df_data['sentiment'].map({'negative': 0, 'neutral': 1, 'p

df_data['sentiment']
```

Out[11]:
```
0    NaN
1    NaN
2    NaN
3    NaN
4    NaN
5    NaN
Name: sentiment, dtype: float64
```

In [12]:
```
df_data
```

```
    X_test = np.vstack(test_df['tokens'].apply(get_document_vector))
    y_train = train_df['sentiment']
    y_test = test_df['sentiment']

    classifier = LogisticRegression(max_iter=1000)
    classifier.fit(X_train, y_train)

    y_pred = classifier.predict(X_test)
    print("Word Embeddings Model")
    print(classification_report(y_test, y_pred, target_names=['negative', 'neutr
```

In [25]:
```
# 3. Transformer Model
def transformer_model(train_df, test_df):
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

    def tokenize_function(examples):
        return tokenizer(examples['text'], padding="max_length", truncation=True

    train_dataset = Dataset.from_pandas(train_df)
    test_dataset = Dataset.from_pandas(test_df)

    train_dataset = train_dataset.map(tokenize_function, batched=True)
    test_dataset = test_dataset.map(tokenize_function, batched=True)

    train_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask
    test_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask'

    model = BertForSequenceClassification.from_pretrained('bert-base-uncased', n

    training_args = TrainingArguments(
        output_dir='./results',
        num_train_epochs=2,
        per_device_train_batch_size=4,
        per_device_eval_batch_size=4,
        warmup_steps=500,
        weight_decay=0.01,
        logging_dir='./logs',
        logging_steps=10,
        evaluation_strategy="epoch"
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=test_dataset
    )

    trainer.train()

    predictions = trainer.predict(test_dataset)
    preds = np.argmax(predictions.predictions, axis=-1)
    y_test = test_df['sentiment'].values
    print("Transformer Model")
    print(classification_report(y_test, preds, target_names=['negative', 'neutra
```

In [ ]:
```
bow_model(train_df, test_df)
word_embeddings_model(train_df, test_df)
transformer_model(train_df, test_df)
```

```
              precision    recall  f1-score   support

    negative       0.00      0.00      0.00       1.0
     neutral       0.00      0.00      0.00       0.0
    positive       0.00      0.00      0.00       1.0

    accuracy                           0.00       2.0
   macro avg       0.00      0.00      0.00       2.0
weighted avg       0.00      0.00      0.00       2.0
```