AI in Natural Language Processing

Batch B Group 4

Group Members:

- Balasuriya R *[CB.EN.U4AIE2105]*
- Nattuva Bhavya Rupa *[CB.EN.U4AIE2140]*
- Navuduri Sameer *[CB.EN.U4AIE2141]*
- Yalavarthi Hima *[CB.EN.U4AIE2178]*

```
import pandas as pd
```

```
data = pd.read_csv('ner_datasetreference.csv',encoding = 'unicode_escape')
data.head()
```

	Sentence #	Word	POS	Tag
0	Sentence: 1	Thousands	NNS	O
1	NaN	of	IN	O
2	NaN	demonstrators	NNS	O
3	NaN	have	VBP	O
4	NaN	marched	VBN	O

```
print('number of tags: {}'.format(len(data.Tag.unique())))
freq = data.Tag.value_counts()
freq
```

```
number of tags: 17
Tag
O        61842
B-geo     2296
I-per     1337
B-gpe     1332
B-org     1327
B-tim     1292
B-per     1192
I-org     1000
I-geo      457
I-tim      370
B-art       62
```

```
B-art       82
B-eve       47
I-gpe       43
I-eve       40
I-art       35
B-nat       24
I-nat       10
Name: count, dtype: int64
```

```
data = data.fillna(method = 'ffill')
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data['Sentence #'] = LabelEncoder().fit_transform(data['Sentence #'])
data.head()
```

	Sentence #	Word	POS	Tag
0	0	Thousands	NNS	O
1	0	of	IN	O
2	0	demonstrators	NNS	O
3	0	have	VBP	O
4	0	marched	VBN	O

```
data.rename(columns = {'Sentence #':'sentence_id',
                       'Word':'words',
                       'Tag':'label'},inplace = True)

data['label'] = data['label'].str.upper()

import pandas as pd



# first few rows
print("First few rows:")
print(data.head())

# basic information
print("\nDataFrame Info:")
print(data.info())

# descriptive statistics
print("\nDescriptive Statistics:")
print(data.describe())
```

```
print(data.describe())

# frequency of each unique value
print("\nLabel Value Counts:")
print(data['label'].value_counts())

# missing values
print("\nMissing Values:")
print(data.isnull().sum())

# duplicated rows
print("\nDuplicated Rows:")
print(data[data.duplicated()])

# Check the shape of the DataFrame
print("\nDataFrame Shape:")
print(data.shape)
```

```
First few rows:
   sentence_id          words  POS label
0            0      Thousands  NNS     O
1            0             of   IN     O
2            0  demonstrators  NNS     O
3            0           have  VBP     O
4            0        marched  VBN     O

DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72706 entries, 0 to 72705
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   sentence_id  72706 non-null  int64
 1   words        72706 non-null  object
 2   POS          72706 non-null  object
 3   label        72706 non-null  object
dtypes: int64(1), object(3)
memory usage: 2.2+ MB
None

Descriptive Statistics:
        sentence_id
count  72706.000000
mean    1645.803400
std      956.813703
min        0.000000
25%      811.000000
50%     1647.000000
75%     2481.000000
max     3299.000000

Label Value Counts:
label
O        61842
```

```
O        61842
B-GEO     2296
I-PER     1337
B-GPE     1332
B-ORG     1327
B-TIM     1292
B-PER     1192
I-ORG     1000
I-GEO      457
I-TIM      370
B-ART       62
B-EVE       47
I-GPE       43
I-EVE       40
I-ART       35
B-NAT       24
I-NAT       10
Name: count, dtype: int64

Missing Values:
sentence_id    0
words          0
POS            0
```

N-grams

```
import pandas as pd
from collections import defaultdict

def generate_ngrams(tokens, n):
    ngrams = defaultdict(int)
    for i in range(len(tokens) - n + 1):
        ngram = ' '.join(tokens[i:i+n])
        ngrams[ngram] += 1
    return ngrams

# Extract words from the 'words' column of your DataFrame
words = data['words'].tolist()

# Generate unigrams
unigrams = generate_ngrams(words, 1)
print("Unigrams:", unigrams)

# Generate bigrams
bigrams = generate_ngrams(words, 2)
print("Bigrams:", bigrams)

# Generate trigrams
trigrams = generate_ngrams(words, 3)
print("Trigrams:", trigrams)
```

```
Unigrams: defaultdict(<class 'int'>, {'Thousands': 9, 'of': 1844, 'demonstrators': 9,
Bigrams: defaultdict(<class 'int'>, {'Thousands of': 9, 'of demonstrators': 2, 'demor
Trigrams: defaultdict(<class 'int'>, {'Thousands of demonstrators': 2, 'of demonstrat
```

BOW

```
import pandas as pd
from collections import defaultdict

# Function to generate bag of words representation
def generate_bow(tokens):
    bow = defaultdict(int)
    for token in tokens:
        bow[token] += 1
    return bow

# Extract words from the 'words' column of your DataFrame
words = data['words'].tolist()

# Generate bag of words
bag_of_words = generate_bow(words)
print("Bag of Words:", bag_of_words)
```

```
Bag of Words: defaultdict(<class 'int'>, {'Thousands': 9, 'of': 1844, 'demonstrators'
```

NLTK

```
import pandas as pd
from collections import defaultdict

# Function to generate bag of words representation
def generate_bow(tokens):
    bow = defaultdict(int)
    for token in tokens:
        bow[token] += 1
    return bow

# Function to generate NTK features
def generate_ntk_features(tokens):
    bow = generate_bow(tokens)
    total_tokens = sum(bow.values())
    ntk_features = {word: count / total_tokens for word, count in bow.items()}
    return ntk_features
```

```
# Extract words from the 'words' column of your DataFrame
words = data['words'].tolist()

# Generate NTK features
ntk_features = generate_ntk_features(words)
print("NTK Features:", ntk_features)
```

```
NTK Features: {'Thousands': 0.0001237862074656837, 'of': 0.025362418507413418, 'demor
```