Twitter Semantic Analysis

NLP - Second Semester 2024

- **Supervised by :**
  - Dr .Zahar Salah

- **Done by:**
  - Shahed Al Zu’bi - 2137097

Twitter sentiment analysis is a real-time automated machine-learning technique that determines and categorises the subjective context in tweets.

Sentiment analysis of Twitter data involves Opinion Mining to analyse the psychological intent in a tIet, positive, negative, or neutral. Then, based on the patterns identified during text mining, it predicts the subsequent thread of texts.

Using python, I wrote a code to sentiment analysis in Twitter and now let's break it down:

1. **Import needed libraries**: in the following cell, I import the libraries that I will be using in the project

```
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
import numpy as np
import pandas as pd
```

2. **Downloading NLTK** (Natural Language Toolkit) data, unzipping it and setting data path:
the following cell downloads and extracts the necessary NLTK data for the project to function properly. NLTK data includes wordnet and omw-1.4 corpora, which are essential for text processing tasks like lemmatization and word sense disambiguation

```
nltk.download('wordnet', "/kaggle/working/nltk_data/")
nltk.download('omw-1.4', "/kaggle/working/nltk_data/")
! unzip /kaggle/working/nltk_data/corpora/wordnet.zip -d /kaggle/working/nltk_data/corpora
! unzip /kaggle/working/nltk_data/corpora/omw-1.4.zip -d /kaggle/working/nltk_data/corpora
nltk.data.path.append("/kaggle/working/nltk_data/")
```

3. Reading and preprocessing Data:

```
training_data =pd.read_csv("twitter_training.csv")
test_data=pd.read_csv("twitter_validation.csv")
```

```
test_data.columns = ['Header1', 'company','labels','text']
training_data.columns = ['Header1', 'company','labels','text']
training_data.drop(columns=["Header1","company"],inplace=True)
test_data.drop(columns=["Header1","company"],inplace=True)
```

training dataset

	labels	text
0	Positive	I am coming to the borders and I will kill you...
1	Positive	im getting on borderlands and i will kill you ...
2	Positive	im coming on borderlands and i will murder you...
3	Positive	im getting on borderlands 2 and i will murder ...
4	Positive	im getting into borderlands and i can murder y...
...	...	...
74676	Positive	Just realized that the Windows partition of my...
74677	Positive	Just realized that my Mac window partition is ...
74678	Positive	Just realized the windows partition of my Mac ...
74679	Positive	Just realized between the windows partition of...
74680	Positive	Just like the windows partition of my Mac is l...

74681 rows × 2 columns

testing dataset

	labels	text
0	Neutral	BBC News - Amazon boss Jeff Bezos rejects clai...
1	Negative	@Microsoft Why do I pay for WORD when it funct...
2	Negative	CSGO matchmaking is so full of closet hacking,...
3	Neutral	Now the President is slapping Americans in the...
4	Negative	Hi @EAHelp I've had Madeleine McCann in my cel...
...	...	...
994	Irrelevant	🌟 Toronto is the arts and culture capital of ...
995	Irrelevant	tHIS IS ACTUALLY A GOOD MOVE TOT BRING MORE VI...
996	Positive	Today sucked so it's time to drink wine n play...
997	Positive	Bought a fraction of Microsoft today. Small wins.
998	Neutral	Johnson & Johnson to stop selling talc baby po...

999 rows × 2 columns

The following code combines the training and test data into one DataFrame, removes any rows with missing values, and then removes duplicate rows to ensure the dataset is clean and ready for further processing and analysis.

```
sentiment=pd.concat([training_data,test_data],ignore_index=True)
sentiment.dropna(inplace=True)
sentiment.drop_duplicates(inplace=True)
```

Then, I create a function that performs a series of text preprocessing steps to prepare text data for further analysis, specifically for tasks like text classification. It takes a text string as input and returns a cleaned list of words.

Steps:

1. Remove Extra White Space:
    - Uses regular expression re.sub(r'\ s +', ' ', text, flags=re.I) to replace multiple consecutive whitespace characters with a single space. This removes unnecessary whitespace and improves consistency.

2. Remove Special Characters:
    - Uses regular expression re.sub(r'\W', ' ', str(text)) to replace all non-alphanumeric characters (special characters) with spaces. This removes punctuation, symbols, and other extraneous characters that might not be relevant for the analysis.

3. Remove Single Characters:
    ○ Uses regular expression re.sub(r'\ s +[a-zA-Z]\ s +', ' ', text) to remove single letters surrounded by whitespace. This might be helpful for handling typos or abbreviations that don't contribute much meaning.

4. Remove Non-Alphanumeric Characters:
    ○ Usesregular expression re.sub(r'[^a-zA-Z\s]', '', text) to remove any character that isn't a letter or a space. This ensures the text only contains alphanumeric characters relevant for the analysis.

5. Convert to Lowercase:
    ○ Converts all characters in the text to lowercase using text.loIr(). This ensures case-insensitivity and improves consistency for further processing.

6. Tokenization:
    ○ Usesword_tokenize(text) from NLTK to split the text into individual words. This creates a list of words for further processing.

7. Lemmatization:
    ○ UsesWordNetLemmatizer() to convert words to their base or dictionary form. This reduces variations

of words (e.g., "running" becomes "run") and improves consistency for analysis.

8. Stop Word Removal:
    ○ Remove Commonstopwords From the list of words using stopwords.words("english"). Stop words are frequent words like "the", "a", "an", etc., that don't carry much meaning on their own.

9. Short Word Removal:
    - Remove Words With a length of 3 characters or less using a list comprehension. This might be helpful for handling typos, abbreviations, or meaningless short words.

10. Unique Word Ordering:
    - Uses Np.unique(Words, return_index=True)[1] to identify unique words in the list and their corresponding indices. This helps maintain the order of unique words while removing duplicates.

11. Creating Cleaned Text:
    - Sort The indices obtained in step 10 and use them to retrieve the corresponding words from the original word list. This ensures the cleaned text maintains the order of unique words while removing duplicates.

```python
def process_text(text):
    text = re.sub(r'\s+', ' ', text, flags=re.I) # Remove extra white space from text

    text = re.sub(r'\W', ' ', str(text)) # Remove all the special characters from text

    text = re.sub(r'\s+[a-zA-Z]\s+', ' ', text) # Remove all single characters from text

    text = re.sub(r'[^a-zA-Z\s]', '', text) # Remove any character that isn't alphabetical

    text = text.lower()

    words = word_tokenize(text) # tokenizes the text into words

    lemmatizer = WordNetLemmatizer() #  used to reduce words to their base or dictionary form
    words = [lemmatizer.lemmatize(word) for word in words]

    stop_words = set(stopwords.words("english"))
    Words = [word for word in words if word not in stop_words] #removes stopwords from the list of words.

    Words = [word for word in Words if len(word) > 3] # removes words with a len less than or equal to 3 characters.

    indices = np.unique(Words, return_index=True)[1] # gets the indices of unique words in the list of words.

    # to create a cleaned list of words by sorting the indices and retrieving corresponding words from the list:
    cleaned_text = np.array(Words)[np.sort(indices)].tolist()

    return cleaned_text
```

After that, I wrote these two lines separating the features (x) from the target variable (y) in the sentiment DataFrame, which is a common step in preparing data for supervised learning tasks like classification.

```python
x=sentiment.drop('labels',axis=1)
y=sentiment.labels
```

x:

	text
0	I am coming to the borders and I will kill you...
1	im getting on borderlands and i will kill you ...
2	im coming on borderlands and i will murder you...
3	im getting on borderlands 2 and i will murder ...
4	im getting into borderlands and i can murder y...
...	...
75668	♥ Suikoden 2\n🅾 Alex Kidd in Miracle World\...
75669	Thank you to Matching funds Home Depot RW paym...
75671	Late night stream with the boys! Come watch so...
75675	⭐ Toronto is the arts and culture capital of ...
75676	tHIS IS ACTUALLY A GOOD MOVE TOT BRING MORE VI...

70251 rows × 1 columns

y:

```
0          Positive
1          Positive
2          Positive
3          Positive
4          Positive
            ...
75668       Neutral
75669      Positive
75671       Neutral
75675    Irrelevant
75676    Irrelevant
Name: labels, Length: 70251, dtype: object
```

4.Training-Building model:

I wrote a text cleaning loop then I converted text Data to string, after that converted text Data to numerical Features using CountVectorizer

```
cleaned_text = [process_text(text) for text in list(x['text'])]
# Convert text data into numerical features using CountVectorizer:
vectorizer = CountVectorizer(lowercase=False)
cleaned_text_str = [' '.join(text) for text in cleaned_text]  # Convert each element to string
X = vectorizer.fit_transform(cleaned_text_str)
```

And here this code prepares the data by splitting it into training and testing sets, trains a Multinomial Naive Bayes classifier on the training data, and then uses the trained classifier to predict sentiment labels for the test data

```
# 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_state=42)

# Train a Naive Bayes classifier:
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)

# Predict sentiment on the test set:
y_pred = nb_classifier.predict(X_test)
```

5. **Evaluate the model:** to evaluate the performance of the trained Naive Bayes classifier on the test data.

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.7537541811970678
Classification Report:
              precision    recall  f1-score   support

  Irrelevant       0.82      0.63      0.71      2431
    Negative       0.71      0.86      0.78      4257
     Neutral       0.83      0.67      0.74      3467
    Positive       0.73      0.79      0.76      3896

    accuracy                           0.75     14051
   macro avg       0.77      0.74      0.75     14051
weighted avg       0.76      0.75      0.75     14051
```