Named Entity Recognition

What is Named Entity Recognition (NER)?

Name-entity recognition (NER) is also referred to as entity identification, entity chunking, and entity extraction. NER is the component of information extraction that aims to identify and categorize named entities within unstructured text. NER involves the identification of key information in the text and classification into a set of predefined categories. An entity is the thing that is constantly talked about or refer to in the text, such as person names, organizations, locations, time expressions, quantities, percentages and more predefined categories.

NER system finds applications across various domains, including question answering, information retrieval and machine translation. NER plays an important role in enhancing the precision of other NLP tasks like part-of-speech tagging and parsing. At its core, NLP is just a two-step process, below are the two steps that are involved:

- Detecting the entities from the text
- Classifying them into different categories

Ambiguity in NER

- For a person, the category definition is intuitively quite clear, but for computers, there is some ambiguity in classification. Let's look at some ambiguous examples:
- England (Organization) won the 2019 world cup vs The 2019 world cup happened in England (Location).
- Washington (Location) is the capital of the US vs The first president of the US was Washington (Person).

How Named Entity Recognition (NER) works?

The working of Named Entity Recognition is discussed below:

- The NER system analyses the entire input text to identify and locate the named entities.

- The system then identifies the sentence boundaries by considering capitalization rules. It recognizes the end of the sentence when a word starts with a capital letter, assuming it could be the beginning of a new sentence. Knowing sentence boundaries aids in contextualizing entities within the text, allowing the model to understand relationships and meanings.
- NER can be trained to classify entire documents into different types, such as invoices, receipts, or passports. Document classification enhances the versatility of NER, allowing it to adapt its entity recognition based on the specific characteristics and context of different document types.
- NER employs machine learning algorithms, including supervised learning, to analyze labeled datasets. These datasets contain examples of annotated entities, guiding the model in recognizing similar entities in new, unseen data.
- Through multiple training iterations, the model refines its understanding of contextual features, syntactic structures, and entity patterns, continuously improving its accuracy over time.
- The model's ability to adapt to new data allows it to handle variations in language, context, and entity types, making it more robust and effective.

```
import spacy
from spacy import displacy
import pandas as pd
```

```
paragraph = " Hello! My name is Harsh Maniya. I am studying engineering in artificial intelligence and data science. I live in Surat. My college name i
```

```
nlp = spacy.load('en_core_web_lg')
doc = nlp(paragraph)
```

```
for ent in doc.ents:
    print(ent.text,'-->', ent.label_)
```

```
Harsh Maniya --> PERSON
Surat --> GPE
SCET --> ORG
AI --> ORG
AppStoneLab Technologies LLP --> ORG
Next Month --> DATE
```

```
displacy.render(doc, style="ent")
```

Hello! My name is Harsh Maniya PERSON . I am studying engineering in artificial intelligence and data science. I live in Surat GPE . My college name is SCET ORG . I have keen interesrt in data science, deep learning and AI ORG agents. i am currently doing an internship at AppStoneLab Technologies LLP ORG . Next Month DATE we have an event at our college.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.chunk import ne_chunk
```

```
tokenized = word_tokenize(paragraph)
tagged = pos_tag(tokenized)
entities = ne_chunk(tagged)
```

```
for entity in entities:
    if hasattr(entity, 'label') and entity.label() == 'ORGANIZATION':
        print(entity.label(),'-->', ''.join(c[0] for c in entity))
    elif hasattr(entity, 'label') and entity.label() == 'GPE':
        print(entity.label(), '-->',''.join(c[0] for c in entity))
    elif hasattr(entity, 'label') and entity.label() == 'PERSON':
        print(entity.label(), '-->',''.join(c[0] for c in entity))
```

```
GPE --> Hello
PERSON --> HarshManiya
GPE --> Surat
ORGANIZATION --> SCET
ORGANIZATION --> AI
ORGANIZATION --> AppStoneLabTechnologies
```

Exercise : Classify restaurant reviews by categories (e.g., service, food, ambiance).

Importing libraries and loading dataset

```
import pandas as pd
import numpy as np
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from gensim.models import Word2Vec
from sklearn.cluster import KMeans
import re
from nltk.sentiment import SentimentIntensityAnalyzer
import nltk
```

Text Classification using K-Means clustering

Preprocessing data

```
stop_words = set(stopwords.words('english'))
```

```
def remove_stopwords(text) :
    tokens = word_tokenize(text)
    return [token for token in tokens if token.isalpha() and token not in stop_words]
```

```
remove_stopwords('My name is Harsh Maniya')
```

```
['My', 'name', 'Harsh', 'Maniya']
```

```
data['Tokens'] = data['Review'].apply(remove_stopwords)
```

Training Word2Vec Model

```
model = Word2Vec(sentences = data['Tokens'], vector_size = 200, min_count = 5)
```

```
def sentence_vector(tokens, model):
    valid_tokens = [word for word in tokens if word in model.wv]
    if valid_tokens:
        word_vectors = np.array([model.wv[word] for word in valid_tokens])
        return word_vectors.mean(axis=0)
    else:
        return np.zeros(model.vector_size)
```

```
data['Vector'] = data['Tokens'].apply(lambda x: sentence_vector(x, model))
```

K-Means clustering

```
X = np.vstack(data['Vector'].values)
kmeans = KMeans(n_clusters=3, random_state=27)
data['Cluster'] = kmeans.fit_predict(X)
```

```
C:\Users\kalat\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=4.
  warnings.warn(
```

```
cluster_labels = {0: 'Service', 1: 'Ambiance', 2: 'Food'}
data['Category'] = data['Cluster'].map(cluster_labels)
```

```
data.head()
```

	Review	Liked	Tokens	Vector	Cluster	Category	Cleaned_Review
0	Wow... Loved this place.	1	[Wow, Loved, place]	[0.004115201, -0.0025320498, -0.00015886906, 0...	1	Ambiance	wow loved this place
1	Crust is not good.	0	[Crust, good]	[-0.0046123504, 0.0041337516, 0.0023621756, 0....	2	Food	crust is not good
2	Not tasty and the texture was just nasty.	0	[Not, tasty, texture, nasty]	[-0.0016557387, -0.0009831111, 0.002366534, 0....	0	Service	not tasty and the texture was just nasty
3	Stopped by during the late May bank holiday of...	1	[Stopped, late, May, bank, holiday, Rick, Stev...	[-0.00035783992, 0.0016495173, 0.00014460708, ...	0	Service	stopped by during the late may bank holiday of...
4	The selection on the menu was great and so wer...	1	[The, selection, menu, great, prices]	[-0.0009349378, 0.002154593, 0.0006511372, 0.0...	2	Food	the selection on the menu was great and so wer...

Context based Text Classification

Preprocessing data

```
def preprocess_text(text):
    text = re.sub(r'\W', ' ', text)
    text = re.sub(r'\s+', ' ', text)
    text = text.lower()
    return text

data2["Cleaned_Review"] = data2["Review"].apply(preprocess_text)
```

Sentiment Analysis

```
sia = SentimentIntensityAnalyzer()

def analyze_sentiment(text):
    score = sia.polarity_scores(text)
    if score['compound'] > 0.05:
        return "Positive"
    elif score['compound'] < -0.05:
        return "Negative"
    else:
        return "Neutral"

data2["Sentiment"] = data2["Cleaned_Review"].apply(analyze_sentiment)
```

Context based Text Classification

```
categories = {
    "Food": ["taste", "delicious", "pizza", "food", "meal", "dish", "tasty", "crust", "breakfast"],  # we have to manually
    "Service": ["service", "staff", "wait", "helpful", "slow", "menu"],                               # add context words
    "Ambiance": ["ambiance", "clean", "noisy", "decor", "environment", "atmosphere", "place"]        # for each category
}

def categorize_review(text):
    for category, keywords in categories.items():
        if any(keyword in text for keyword in keywords):
            return category
    return "Other"

data2["Category"] = data2["Cleaned_Review"].apply(categorize_review)

data2["Category_Sentiment"] = data2["Category"] + " - " + data2["Sentiment"]

data2.head()
```

	Review	Liked	Cleaned_Review	Sentiment	Category	Category_Sentiment
0	Wow... Loved this place.	1	wow loved this place	Positive	Ambiance	Ambiance - Positive
1	Crust is not good.	0	crust is not good	Negative	Food	Food - Negative
2	Not tasty and the texture was just nasty.	0	not tasty and the texture was just nasty	Negative	Food	Food - Negative
3	Stopped by during the late May bank holiday of...	1	stopped by during the late may bank holiday of...	Positive	Other	Other - Positive
4	The selection on the menu was great and so wer...	1	the selection on the menu was great and so wer...	Positive	Service	Service - Positive

Fasttext

FastText is a library for efficient learning of word representations and sentence classification developed by Facebook's AI Research (FAIR) lab. It builds on the success of word2vec but extends it by using n-gram features and hierarchical softmax to improve performance on large datasets.

FastText introduces several key concepts:

1. Word Embeddings: Like word2vec, FastText represents words as continuous vectors (embeddings) in a high-dimensional space. These embeddings capture semantic relationships between words based on their context in the training corpus.
2. Subword Information: Unlike traditional word2vec, FastText considers subword information by breaking words into character n-grams (subword units). This allows it to handle rare words and morphologically rich languages better.
3. Hierarchical Softmax: FastText uses a hierarchical softmax to efficiently compute the probability distribution over words in the vocabulary, speeding up training and inference compared to the softmax used in word2vec.

```
!pip install fasttext
```

```
Requirement already satisfied: fasttext in /usr/local/lib/python3.11/dist-packages (0.9.3)
Requirement already satisfied: pybind11>=2.2 in /usr/local/lib/python3.11/dist-packages (from fasttext) (2.13.6)
Requirement already satisfied: setuptools>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from fasttext) (75.1.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from fasttext) (1.26.4)
```

```
import fasttext
import urllib.request

link_en = 'https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.en.300.bin.gz'
urllib.request.urlretrieve(link_en, "en.gz")

!gunzip en.gz

model_en = fasttext.load_model('en')
```

```
model_en.get_nearest_neighbors('AAP')
```

```
[(0.6074421405792236, 'BJP'),
 (0.5642463564872742, 'Kejriwal'),
 (0.5436301827430725, 'AAPs'),
 (0.538779079914093, 'Kejriwal-led'),
 (0.5221178531646729, 'CPI-M'),
 (0.5152238607406616, 'UPA'),
 (0.5150542855262756, 'JD-U'),
 (0.49802857637405396, 'MLA'),
 (0.4927403032779936, 'Pawar'),
 (0.4922226369380951, 'Chhotepur')]
```

```
model_en.get_nearest_neighbors('india')
```

```
[(0.8226807117462158, 'pakistan'),
 (0.7812276482582092, 'delhi'),
 (0.7671213746070862, 'india.'),
 (0.7481372952461243, 'mumbai'),
 (0.7414579391479492, 'hyderabad'),
 (0.7363941669464111, 'kolkata'),
 (0.72954922914505, 'ahmedabad'),
 (0.7264755964279175, 'chennai'),
 (0.7260774374008179, 'bangalore'),
 (0.7184210419654846, 'bangladesh')]
```