Lecture Information

Course: CSCI E-89B: Natural Language Processing
Lecture: Lecture 10
Topic: Named Entity Recognition (NER)
Date: Fall 2024

Contents

1 Introduction to Named Entity Recognition

Overview

Named Entity Recognition (NER) is the task of identifying and classifying named entities in text into predefined categories such as person names, organizations, locations, dates, and more. It's a fundamental NLP task with applications in information extraction, question answering, and machine translation.

1.1 What are Named Entities?

Named Entity Categories

Common named entity types include:

- **PERSON**: Names of people (Donald Trump, Marie Curie)
- **ORG**: Organizations, companies, institutions (Tesla, Harvard University)
- **GPE**: Geopolitical entities—countries, cities, states (America, Paris)
- **LOC**: Non-GPE locations (Mount Everest, Pacific Ocean)
- **DATE**: Dates and time periods (November 4, 2024, this year)
- **TIME**: Times (3:00 PM, midnight)
- **MONEY**: Monetary values ($1 trillion, 50 euros)
- **PERCENT**: Percentages (40%, two-thirds)
- **CARDINAL**: Numbers not fitting other categories (50, three)
- **ORDINAL**: Ordinal numbers (first, 2nd)
- **NORP**: Nationalities, religious/political groups (Chinese, Republican)

1.2 Why is NER Important?

Important

NER is crucial for many downstream tasks:

- **Machine Translation**: Knowing "Tesla" is an organization helps translate correctly
- **Information Extraction**: Extract structured data from unstructured text
- **Question Answering**: Identify entities mentioned in questions
- **Search**: Improve semantic search by understanding entity types
- **Sentiment Analysis**: Attribute sentiment to specific entities

NER in Action

Input text: "Donald Trump won more than 50 electoral votes this year. Tesla's stock rose 2.4%."
NER output:

Entity	Type	Position
Donald Trump	PERSON	0–11
more than 50	CARDINAL	17–29
this year	DATE	47–56
Tesla	ORG	58–63
2.4%	PERCENT	78–82

1.3 NER for Feature Enhancement

Enhancing Classification with NER

NER can improve text classification by:

1. **Adding entity counts**: Concatenate counts of persons, organizations, etc.
2. **Entity-based features**: Create binary indicators for entity presence
3. **Structured metadata**: Extract entities as document metadata
4. **Relationship extraction**: Find connections between entities

2 Two Approaches to NER

Overview

NER can be performed using two main approaches: rule-based methods using pattern matching, and statistical/neural methods using machine learning.

2.1 Approach Comparison

Aspect	Rule-Based	Statistical/Neural
Training Data	Not required	Required (labeled)
Context Awareness	Limited	High
Maintenance	High burden	Lower
Adaptability	Poor	Good
Interpretability	High	Lower
Scalability	Poor	Good
Accuracy	Depends on rules	Generally higher

3 Rule-Based NER

Overview

Rule-based NER uses handcrafted patterns (regular expressions, dictionaries, linguistic rules) to identify entities. While limited, it's interpretable and requires no training data.

3.1 Regular Expressions for Entity Detection

Common Pattern Components

- `\b` : Word boundary
- `\d{n}` : Exactly n digits
- `\d{1,2}` : 1 or 2 digits
- `\s+` : One or more whitespace characters
- `[A-Z]` : One uppercase letter
- `[a-z]+` : One or more lowercase letters
- `(?:...)` : Non-capturing group
- `?` : Makes preceding element optional
- : OR operator

3.2 Date Pattern Example

```python
import re

# Pattern for dates like "11/04/2024" or "November 4, 2024"
date_pattern = r'''
    \b                             # Word boundary
    (?:
        \d{1,2}/\d{1,2}/\d{4}      # MM/DD/YYYY format
        |
        (?:January|February|March|April|May|June|
           July|August|September|October|November|December)
        \s+                         # Required space
        \d{1,2}                     # Day (1-31)
        (?:,\s*)?                   # Optional comma and space
        \d{4}                       # Year
    )
    \b
'''

text = "The event is on November 4, 2024 or 11/04/2024."
dates = re.findall(date_pattern, text, re.VERBOSE)
print(dates)  # ['November 4, 2024', '11/04/2024']
```

Listing 1: Regular Expression for Dates

3.3 Person Name Pattern

```python
# Pattern for names with titles
```

```
person_pattern = r'''
    \b
    (?:Mr\.|Mrs\.|Ms\.|Dr\.|Professor)  # Title
    \s+                                  # Space
    [A-Z][a-z]+                         # First name (capitalized)
    (?:\s+[A-Z][a-z]+)?                 # Optional last name
    \b
'''

text = "Mr. Trump met with Dr. Smith yesterday."
persons = re.findall(person_pattern, text, re.VERBOSE)
print(persons)  # ['Mr. Trump', 'Dr. Smith']
```

Listing 2: Pattern for Names with Titles

3.4 Organization Pattern

```
# Pattern for company names
org_pattern = r'''
    \b
    [A-Z][a-zA-Z\s]+                    # Company name
    (?:Inc\.|Ltd\.|Corporation|Corp\.)  # Corporate suffix
    \b
'''

text = "Apple Inc. announced new products."
orgs = re.findall(org_pattern, text, re.VERBOSE)
print(orgs)  # ['Apple Inc.']
```

Listing 3: Pattern for Organizations

3.5 Complete Rule-Based NER System

```
import re

def rule_based_ner(text):
    entities = []

    # Date patterns
    date_patterns = [
        r'\b\d{1,2}/\d{1,2}/\d{4}\b',
        r'\b(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[a-z]*\s+\d{1,2},?\s*\d{4}\b'
    ]

    # Email pattern
    email_pattern = r'\b[\w.]+@[\w.]+\.\w+\b'

    # Time pattern
    time_pattern = r'\b\d{1,2}:\d{2}\s*(?:AM|PM|am|pm)?\b'

    # Person pattern (with titles)
    person_pattern = r'\b(?:Mr\.|Mrs\.|Ms\.|Dr\.)\s+[A-Z][a-z]+(?:\s+[A-Z][a-z]+)?\b'

    # Organization pattern
    org_pattern = r'\b[A-Z][a-zA-Z\s]+(?:Inc\.|Ltd\.|Corp\.)\b'

    # Apply patterns
    for pattern in date_patterns:
        for match in re.finditer(pattern, text):
```

```
            entities.append(('DATE', match.group(), match.span()))

    for match in re.finditer(email_pattern, text):
        entities.append(('EMAIL', match.group(), match.span()))

    for match in re.finditer(time_pattern, text):
        entities.append(('TIME', match.group(), match.span()))

    for match in re.finditer(person_pattern, text):
        entities.append(('PERSON', match.group(), match.span()))

    for match in re.finditer(org_pattern, text):
        entities.append(('ORG', match.group(), match.span()))

    return entities

# Test
text = """
Meeting with Dr. Smith at 3:00 PM on November 18, 2024.
Contact: john.doe@company.com. Apple Inc. will attend.
"""
entities = rule_based_ner(text)
for entity_type, entity, span in entities:
    print(f"{entity_type}: {entity} at {span}")
```

Listing 4: Simple Rule-Based NER System

3.6 Limitations of Rule-Based NER

Rule-Based Limitations

- **No context awareness**: "Tesla" could be a person (Nikola Tesla) or company
- **Missing entities**: "Donald Trump" without title won't be recognized
- **Language-specific**: Rules must be rewritten for each language
- **Date format variations**: US vs European formats differ
- **Maintenance burden**: Rules must be constantly updated
- **Name variations**: "La Place" vs "Laplace" requires special handling

4 Statistical and Neural NER

Overview

Modern NER systems use machine learning to learn patterns from labeled data. Neural networks, particularly CNNs and transformers, achieve state-of-the-art performance.

4.1 NER as Sequence Labeling

BIO Tagging Scheme

NER is typically framed as a sequence labeling problem using BIO tags:

- **B-TYPE**: Beginning of an entity of TYPE
- **I-TYPE**: Inside/continuation of an entity

- **O**: Outside any entity

BIO Tagging Example

Sentence: "Donald Trump visited Tesla headquarters."

Token	Tag
Donald	B-PERSON
Trump	I-PERSON
visited	O
Tesla	B-ORG
headquarters	O
.	O

4.2 Statistical Methods

Traditional ML for NER

Hidden Markov Models (HMM):

- Model sequence of tags as Markov chain
- Assume current tag depends only on previous tag
- Limited feature representation

Conditional Random Fields (CRF):

- Discriminative model (directly models $P(\text{tags}|\text{words})$)
- Can use arbitrary features
- Global normalization (considers entire sequence)

4.3 Neural Network Architecture for NER

Important

SpaCy uses a **Convolutional Neural Network** (CNN) for NER. Here's why:

1. Word embeddings capture semantic meaning
2. CNN filters capture local context (neighboring words)
3. Sliding window naturally handles variable-length text
4. Efficient parallel computation

4.4 How Neural NER Works

Neural NER Pipeline

1. **Input**: Document text
2. **Tokenization**: Split into tokens
3. **Embedding**: Convert tokens to vectors (e.g., 4D, 100D)
4. **CNN**: Apply filters over embedding sequences
5. **Output Layer**: Softmax over entity types for each token

NER as CNN Classification

Input: "The cat sat on mat"
Step 1: Embed each token:

Token	Dim 1	Dim 2	Dim 3	Dim 4
The	0.8	0.1	0.3	0.7
cat	0.5	0.7	0.2	0.9
sat	0.3	0.4	0.6	0.1
on	0.2	0.8	0.1	0.5
mat	0.4	0.3	0.7	0.2

Step 2: CNN filters slide over embeddings (captures context)
Step 3: For each token, output probabilities:

Token	PERSON	ORG	GPE	DATE	O
The	0.01	0.01	0.01	0.02	0.95
cat	0.10	0.02	0.01	0.02	0.85
...					

5 Using SpaCy for NER

Overview

SpaCy provides pre-trained NER models that are easy to use and highly accurate for common entity types.

5.1 Basic SpaCy NER

```python
import spacy
from spacy import displacy

# Load pre-trained model
nlp = spacy.load("en_core_web_sm")

# Process text
text = """Donald Trump won more than 50 electoral votes this year.
Tesla's stock rose 2.4% after the Federal Reserve announcement."""

doc = nlp(text)

```

```python
# Extract entities
for ent in doc.ents:
    print(f"{ent.text:20} {ent.label_:10} {ent.start_char}-{ent.end_char}")
```

Listing 5: SpaCy NER Basics

Output:

```
Donald Trump          PERSON      0-12
more than 50          CARDINAL    17-29
this year             DATE        47-56
Tesla                 ORG         58-63
2.4%                  PERCENT     78-82
Federal Reserve       ORG         94-109
```

5.2 Visualizing Entities

```python
# Render in Jupyter notebook
displacy.render(doc, style="ent", jupyter=True)

# Or save to HTML file
html = displacy.render(doc, style="ent", page=True)
with open("ner_visualization.html", "w") as f:
    f.write(html)
```

Listing 6: Visualize NER with displacy

5.3 SpaCy NER Label Reference

Label	Description
PERSON	People, including fictional
NORP	Nationalities, religious, political groups
FAC	Facilities (buildings, airports, highways)
ORG	Companies, agencies, institutions
GPE	Countries, cities, states
LOC	Non-GPE locations
PRODUCT	Objects, vehicles, foods
EVENT	Named hurricanes, battles, wars
WORK_OF_ART	Titles of books, songs
LAW	Named documents made into laws
DATE	Absolute or relative dates
TIME	Times smaller than a day
PERCENT	Percentage
MONEY	Monetary values
QUANTITY	Measurements
ORDINAL	"first", "second", etc.
CARDINAL	Numerals not falling into other categories

6 Fine-Tuning SpaCy NER

Overview

Pre-trained NER models may not recognize domain-specific entities. SpaCy allows fine-tuning on custom data to add new entity types or improve accuracy.

6.1 When to Fine-Tune

Important

Consider fine-tuning when:

- Domain-specific entities (drug names, product codes)
- New entity categories not in default model
- Improving accuracy for your specific text type
- Handling industry jargon or technical terms

6.2 Training Data Format

```python
# Training data format: (text, {"entities": [(start, end, label)]})
train_data = [
    ("Cars in China are selling well",
     {"entities": [(0, 4, "VEHICLE")]}),

    ("Tesla has a lot on the line as an electric vehicle maker",
     {"entities": [(0, 5, "ORG"), (40, 56, "VEHICLE")]}),

    ("My family loves our Honda Civic",
     {"entities": [(23, 34, "VEHICLE")]}),

    ("This car is the best",
     {"entities": [(5, 8, "VEHICLE")]})
]
```

Listing 7: SpaCy Training Data Format

6.3 Fine-Tuning Process

```python
import spacy
from spacy.training import Example
import random

# Load existing model
nlp = spacy.load("en_core_web_sm")

# Get the NER component
ner = nlp.get_pipe("ner")

# Add new entity label
ner.add_label("VEHICLE")

# Disable other pipes during training
other_pipes = [pipe for pipe in nlp.pipe_names if pipe != "ner"]

# Training loop
```

```python
with nlp.disable_pipes(*other_pipes):
    optimizer = nlp.resume_training()

    for iteration in range(20):
        random.shuffle(train_data)
        losses = {}

        for text, annotations in train_data:
            doc = nlp.make_doc(text)
            example = Example.from_dict(doc, annotations)
            nlp.update([example], drop=0.5, losses=losses)

        print(f"Iteration {iteration}, Losses: {losses}")

# Test the model
doc = nlp("I bought a Toyota Camry yesterday")
for ent in doc.ents:
    print(f"{ent.text}: {ent.label_}")
```

Listing 8: Fine-Tuning SpaCy NER

Fine-Tuning Pitfalls

- **Catastrophic forgetting**: Model may "forget" original entities
- **Insufficient data**: Need many examples per entity type
- **Label consistency**: Annotations must be consistent
- Always include some original data in training to prevent forgetting

7 Part-of-Speech Tagging

Overview

Part-of-Speech (POS) tagging identifies grammatical categories (noun, verb, adjective, etc.) for each word. It's closely related to NER and often used as a preprocessing step.

7.1 Common POS Tags

Tag	Description	Example
NN	Noun, singular	cat, dog, house
NNS	Noun, plural	cats, dogs, houses
NNP	Proper noun, singular	John, London
VB	Verb, base form	run, eat, be
VBD	Verb, past tense	ran, ate, was
VBG	Verb, gerund	running, eating
JJ	Adjective	big, red, beautiful
RB	Adverb	quickly, very, well
IN	Preposition	in, on, at, by
DT	Determiner	the, a, an
CC	Coordinating conjunction	and, but, or
TO	"to"	to (as in "to run")

7.2 POS Tagging with NLTK

```python
import nltk
from nltk import word_tokenize, pos_tag

# Download required data
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('tagsets')

# View tag descriptions
nltk.help.upenn_tagset('NN')   # Noun
nltk.help.upenn_tagset('VB')   # Verb

# POS tagging
text = "The quick brown fox jumps over the lazy dog"
tokens = word_tokenize(text)
pos_tags = pos_tag(tokens)

print(pos_tags)
# [('The', 'DT'), ('quick', 'JJ'), ('brown', 'JJ'),
#  ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'),
#  ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN')]
```

Listing 9: POS Tagging with NLTK

7.3 POS Tagging with SpaCy

```python
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("The quick brown fox jumps over the lazy dog")

for token in doc:
    print(f"{token.text:10} {token.pos_:6} {token.tag_}")
```

Listing 10: POS Tagging with SpaCy

8 Enhancing Classification with NER

Overview

NER features can improve text classification by providing structured information about document content.

8.1 Feature Engineering with NER

NER-Based Features

Count-based features:

- Number of persons mentioned
- Number of organizations
- Number of locations
- Number of dates/times

Binary indicators:

- Contains person name? (0/1)
- Contains organization? (0/1)
- Contains specific entity (e.g., "Tesla")? (0/1)

8.2 Implementation Example

```python
import spacy
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
import numpy as np

nlp = spacy.load("en_core_web_sm")

def extract_ner_features(text):
    """Extract NER-based features from text"""
    doc = nlp(text)

    features = {
        'n_persons': 0,
        'n_orgs': 0,
        'n_gpes': 0,
        'n_dates': 0,
        'n_money': 0,
        'n_percent': 0
    }

    for ent in doc.ents:
        if ent.label_ == 'PERSON':
            features['n_persons'] += 1
        elif ent.label_ == 'ORG':
            features['n_orgs'] += 1
        elif ent.label_ == 'GPE':
            features['n_gpes'] += 1
        elif ent.label_ == 'DATE':
            features['n_dates'] += 1
        elif ent.label_ == 'MONEY':
            features['n_money'] += 1
        elif ent.label_ == 'PERCENT':
            features['n_percent'] += 1

    return features

# Extract NER features for all documents
ner_features = [extract_ner_features(text) for text in documents]
ner_df = pd.DataFrame(ner_features)

# Combine TF-IDF and NER features
tfidf = TfidfVectorizer(max_features=20)
X_tfidf = tfidf.fit_transform(documents).toarray()
X_combined = np.hstack([X_tfidf, ner_df.values])

# Train classifier
X_train, X_test, y_train, y_test = train_test_split(
    X_combined, labels, test_size=0.2, random_state=42
)

clf = MLPClassifier(hidden_layer_sizes=(50,), max_iter=500)
```

```python
clf.fit(X_train, y_train)
accuracy = clf.score(X_test, y_test)
print(f"Accuracy with NER features: {accuracy:.4f}")
```

Listing 11: NER Feature Enhancement

8.3 Entity-Based Binary Features

```python
def get_entity_dummies(documents):
    """Create binary indicators for each unique entity"""
    all_entities = set()

    # First pass: collect all unique entities
    for text in documents:
        doc = nlp(text)
        for ent in doc.ents:
            all_entities.add((ent.text, ent.label_))

    # Second pass: create binary features
    feature_matrix = []
    for text in documents:
        doc = nlp(text)
        doc_entities = set((ent.text, ent.label_) for ent in doc.ents)

        row = [1 if entity in doc_entities else 0
               for entity in all_entities]
        feature_matrix.append(row)

    columns = [f"{text}_{label}" for text, label in all_entities]
    return pd.DataFrame(feature_matrix, columns=columns)

entity_features = get_entity_dummies(documents)
```

Listing 12: Binary Entity Indicators

9 One-Page Summary

Summary

Named Entity Recognition (NER) identifies and classifies named entities in text.
Common Entity Types:

- PERSON, ORG, GPE, LOC, DATE, TIME, MONEY, PERCENT, CARDINAL

Two Approaches:
Rule-Based:

- Uses regular expressions and dictionaries
- No training data needed
- Limited by predefined patterns
- No context awareness

Statistical/Neural:

- Learns from labeled data
- Uses context for disambiguation

- CNNs capture local context via filters
- SpaCy: `nlp = spacy.load("en_core_web_sm")`

NER as Sequence Labeling:

- BIO scheme: B-TYPE (begin), I-TYPE (inside), O (outside)
- Each token gets a tag
- Output: softmax probabilities over entity types

SpaCy Usage:

```python
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Donald Trump visited Tesla.")
for ent in doc.ents:
    print(ent.text, ent.label_)
```

Fine-Tuning: Add custom entity types with labeled examples. Watch for catastrophic forgetting.
Feature Enhancement:

- Add entity counts to feature vectors
- Create binary entity indicators
- Combine with TF-IDF for classification

Applications: Translation, information extraction, question answering, sentiment analysis, search.

10 Glossary

Key Terms

- **NER**: Named Entity Recognition—identifying entities in text
- **Named Entity**: Real-world object with a name (person, organization, place)
- **BIO Tagging**: Begin-Inside-Outside scheme for sequence labeling
- **POS Tagging**: Part-of-Speech tagging—grammatical categories
- **Regular Expression**: Pattern matching syntax for text
- **Rule-Based NER**: Pattern-matching approach to entity extraction
- **Statistical NER**: Machine learning approach to entity extraction
- **HMM**: Hidden Markov Model—probabilistic sequence model
- **CRF**: Conditional Random Fields—discriminative sequence model
- **SpaCy**: Industrial-strength NLP library for Python
- **Fine-Tuning**: Continuing training on domain-specific data

- **Catastrophic Forgetting**: Model losing original knowledge during fine-tuning
- **GPE**: Geopolitical Entity (countries, cities, states)
- **NORP**: Nationalities, religious, or political groups
- **displacy**: SpaCy's visualization module
- **NLTK**: Natural Language Toolkit—Python NLP library
- **Context**: Surrounding words that help disambiguate meaning
- **Word Boundary**: `\b` in regex—edges of words