December 10, 2025

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 10

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 10의 핵심 개념 학습

Contents

□ 핵심 요약

문서 개요: 이 문서는 자연어 처리(NLP)의 핵심 기술인 **개체명 인식(Named Entity Recognition, NER)**에 대해 다룹니다. NER이 무엇인지, 왜 중요한지, 그리고 어떻게 작동하는지 설명합니다.

핵심 내용:

- **NER 정의:** 텍스트에서 '이름'을 가진 개체(예: 사람, 기관, 장소)를 찾아내고 분류하는 기술입니다.
- **두 가지 접근법:**
  1. **규칙 기반(Rule-based):** 정규표현식(Regex)처럼 사전에 정의된 규칙으로 개체를 찾습니다. (예: "Inc."로 끝나면 '기관')
  2. **통계 기반(Statistical):** 대규모 데이터를 학습한 모델(예: 신경망)이 문맥을 파악하여 개체를 찾습니다.
- **구현:** 'NLTK'와 'spaCy' 라이브러리를 사용한 NER 구현 방법을 배웁니다.
- **심화:** 통계 기반 NER(특히 CNN)의 내부 작동 원리와 기존 모델을 특정 데이터에 맞게 추가 학습시키는 **미세조정(Fine-tuning)** 개념을 살펴봅니다.

학습 목표: 이 문서를 통해 수강생은 NER의 기본 원리를 이해하고, 파이썬으로 간단한 NER 시스템을 구현하며, NER을 다른 NLP 작업(예: 텍스트 분류)에 응용하는 방법을 설명할 수 있습니다.

1 개체명 인식(NER)이란 무엇인가?

한 줄 정의: NER (Named Entity Recognition)

텍스트에서 **'이름'이 붙은 고유한 대상**을 찾아내고, 그것이 어떤 유형(예: 사람, 기관, 장소)인지 **분류(Labeling)**하는 작업입니다.

직관적 비유: '형광펜 분류기'

NER을 "여러 색상의 형광펜을 가진 조교"라고 생각할 수 있습니다.

- **노란색:** 사람 이름 (예: "Donald Trump", "Elon Musk")
- **파란색:** 기관/조직 (예: "Tesla", "Federal Reserve", "Apple Inc.")
- **녹색:** 장소 (예: "America", "China")
- **주황색:** 날짜/시간 (예: "this year", "March")
- **분홍색:** 돈/수량 (예: "$1trn", "40%")

NER의 임무는 긴 문서(텍스트)를 읽으며 이 형광펜으로 정확하게 밑줄을 긋고 분류하는 것입니다.

1.1 NER의 주요 개체 유형

NER이 인식하는 '개체'는 표준화된 범주를 따르는 경우가 많습니다.

Table 1: *NER*의 일반적인 개체 유형

태그	원어 (Type)	설명 및 예시
`PERSON`	Person	사람 이름 (예: "Donald Trump", "Dr. John Smith")
`ORG`	Organization	기관, 회사, 정부 (예: "Tesla", "Bank of England")
`GPE`	Geopolitical Entity	지정학적 개체 (국가, 도시, 주) (예: "America", "New York City")
`DATE`	Date	날짜 (예: "November 18, 2024", "this year")
`TIME`	Time	시간 (예: "10:00 AM")
`MONEY`	Money	화폐 단위 (예: "$1trn", "$90,000")
`PERCENT`	Percent	백분율 (예: "40%", "2.6%")
`CARDINAL`	Cardinal Number	기수 (일반 숫자) (예: "50", "44,000")
`ORDINAL`	Ordinal Number	서수 (순서) (예: "first")
`NORP`	Nationalities or ⋯	국적, 종교, 정치 단체 (예: "Chinese")

1.2 NER 적용 예시

다음은 특정 경제 기사(The Economist)에 NER을 적용한 예시입니다.

□ **예제:**

원본 텍스트: "Markets continued to rally in response to **Donald Trump**'s election victory. The **S&P 500** hit another high (it has broken **more than 50** records so far **this year**) ⋯ The rise in **Tesla**'s stock pushed the carmaker above a valuation of **$1trn**, which it last achieved in **early 2022**. ⋯ Traders still expect the **Federal Reserve** to cut interest rates ⋯"

NER 적용 결과:

- "Donald Trump": **PERSON**
- "S&P 500": **ORG** (기관으로 분류될 수 있음)
- "more than 50": **CARDINAL**
- "this year": **DATE**
- "Tesla": **ORG**
- "$1trn": **MONEY**
- "early 2022": **DATE**
- "Federal Reserve": **ORG**

참고: *"Bitcoin"*의 경우, 문맥에 따라 ***PERSON***으로 잘못 분류될 수도 있습니다. (예: *"Bitcoin surged ... "* 이 구문만 보면 사람 이름처럼 보일 수 있음) 이는 모델이 문맥을 어떻게 학습했는지에 따라 달라집니다.

2 NER은 왜 중요한가? (주요 활용 분야)

NER은 단순히 텍스트에 밑줄을 긋는 것을 넘어, 다른 NLP 작업들의 성능을 비약적으로 향상시키는 **전처리(preprocessing)** 또는 **특성 공학(feature engineering)**의 핵심 단계입니다.

- **정보 추출 (Information Extraction):** 비정형 텍스트(예: 뉴스 기사, 이메일)에서 핵심 정보를 뽑아내어 정형 데이터(예: 데이터베이스, 엑셀 시트)를 만드는 데 사용됩니다.
  - 예시: 수천 개의 이력서에서 '사람 이름', '회사명', '직무'를 자동으로 추출하여 표로 정리합니다.
- **텍스트 분류 (Text Classification) 성능 향상:** 문서에 어떤 '유형'의 개체가 포함되어 있는지를 모델에 알려주어 분류 정확도를 높입니다.
  - 예시: 문서에 `ORG` (기관) 태그가 많이 등장하면 '비즈니스' 또는 '정치' 기사일 확률이 높습니다. (자세한 내용은 8 참조)
- **질의응답 (Question Answering) 시스템:** 사용자의 질문(Query)과 문서 내의 잠재적 답변(Answer)에서 개체 유형을 일치시켜 정확한 답을 찾습니다.
  - 질문: "Who is the CEO of Tesla?" (질문 유형: `PERSON`)
  - 문서 검색: "Tesla (`ORG`) ⋯ Elon Musk (`PERSON`) ⋯"
  - 답변: 질문이 `PERSON`을 물었으므로, 문서에서 찾은 `PERSON`인 "Elon Musk"를 답변으로 반환합니다.
- **기계 번역 (Machine Translation):** '이름'을 일반 명사로 오인하여 잘못 번역하는 것을 방지합니다.
  - 오번역 예시: "Apple (회사) is hiring." → "사과가 고용 중이다." (X)
  - *NER* 적용: "Apple (`ORG`) is hiring." → "애플(사)이 고용 중이다." (O)
- **의미 검색 (Semantic Search):** 단순 키워드 검색이 아닌 '의미' 기반 검색을 가능하게 합니다.
  - 검색어: "Apple"
  - 결과: '사과(과일)'에 대한 문서와 '애플(회사)'에 대한 문서를 NER로 구분하여 사용자에게 제시합니다.

3 접근법 1: 규칙 기반 NER (Rule-based NER)

한 줄 정의: 개발자가 사전에 정의한 **명시적인 규칙(Rule)의 목록**을 사용하여 텍스트에서 개체명을 찾는 방식입니다.

직관적 비유: '엄격한 체크리스트 검사원' 규칙 기반 NER은 "체크리스트만 보고 판단하는 검사원"과 같습니다.

- "텍스트에 'Mr.', 'Mrs.', 'Dr.'가 있으면 그 뒤 단어는 `PERSON`이다."
- "텍스트가 '000-000-0000' 형식이면 `PHONE_NUMBER`이다."
- "텍스트가 'Inc.', 'Ltd.', 'Corp.'로 끝나면 `ORG`이다."

이 검사원은 빠르고 정확하지만, 체크리스트에 없는 새로운 패턴(예: "Tesla"처럼 'Inc.'가 없는 회사명)은 절대로 인식하지 못합니다.

3.1 주요 기술: 정규표현식 (Regular Expressions)

규칙 기반 NER에서 가장 많이 사용되는 도구는 **정규표현식(Regex)**입니다. Regex는 특정 문자열 패턴을 정의하는 문법입니다.

□ **예제:**

사례 1: 날짜, 이메일, 조직명(Inc) 인식하기
다음은 파이썬의 're' 라이브러리를 사용한 예시입니다.

```python
import re

text = """
Dr. John Smith met with Apple Inc. on November 18, 2024.
His email is john.smith@email.com, and the meeting was at 10:00 AM.
"""

# 1. 날짜패턴예  (: MM/DD/YYYY 또는 Month D, YYYY)
date_pattern = r'\b(?:January|February|...|November|December)\s+\d
    {1,2},\s*\d{4}\b'
# 2. 이메일패턴
email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
# 3. 시간패턴  (AM/PM)
time_pattern = r'\b\d{1,2}:\d{2}\s*(?:AM|PM)\b'
# 4. 조직명패턴  (Inc, Ltd, 로Corp 끝나는경우 )
org_pattern = r'\b[A-Z][a-zA-Z]+\s+(?:Inc|Ltd|Corp)\b'

print(f"Dates: {re.findall(date_pattern, text)}")
print(f"Emails: {re.findall(email_pattern, text)}")
print(f"Times: {re.findall(time_pattern, text, flags=re.IGNORECASE)}")
print(f"Orgs: {re.findall(org_pattern, text)}")

# --- 출력결과  ---
# Dates: ['November 18, 2024']
# Emails: ['john.smith@email.com']
```

```
    # Times: ['10:00 AM']
    # Orgs: ['Apple Inc']
```

Listing 1: 규칙 기반 NER을 위한 정규표현식 예시 (날짜, 이메일 등)

□ 예제:

사례 2: 호칭(Title)을 이용한 사람 이름 인식하기

'Mr.', 'Dr.' 등의 호칭(Title)을 기준으로 사람을 찾는 더 간단한 규칙입니다.

```
    text = "Mr. Musk and Mr. Trump met with Dr. Emily White."

    # 1. 호칭패턴  (Mr, Mrs, Ms, Dr, Professor)
    # \s+ : 하나이상의공백
    # [A-Z][a-z]+ : 대문자로시작하고소문자가이어지는단어이름      ()
    person_pattern = r'\b(Mr|Mrs|Ms|Dr|Professor)\.?\s+[A-Z][a-z]+(?:\s+[A-
        Z][a-z]+)?'
    # (?:\s+[A-Z][a-z]+)? : 선택적() 공백과성 (Last Name)이 나올수도있음

    print(f"Persons: {re.findall(person_pattern, text)}")

    # --- 출력결과  ---
    # Persons: [('Mr', ' Musk'), ('Mr', ' Trump'), ('Dr', ' Emily White')]
    # 참고(: 정규식의캡처그룹설정에따라출력형태가다를수있습니다          .)
```

Listing 2: 호칭(Title) 기반 정규표현식 예시

3.2 규칙 기반 NER의 장단점

Table 2: 규칙 기반 *NER*의 장점과 단점

장점 (Pros)	단점 (Cons)
높은 정밀도(Precision): 규칙이 명확한 도메인(예: 이메일, 주민번호)에서는 거의 100% 정확하게 작동합니다.	**낮은 재현율(Recall) / 유연성 부족:** 규칙에 없는 새로운 패턴(예: "Apple")을 놓치기 쉽습니다.
해석 가능성 (Interpretability): 왜 해당 개체를 인식했는지 규칙을 통해 100% 설명 가능합니다.	**유지보수 부담:** 언어 사용이 변하거나 새로운 유형의 개체가 생기면(예: "COVID-19") 매번 규칙을 수동으로 업데이트해야 합니다.
데이터 불필요: 모델을 학습시키기 위한 대규모 라벨링 데이터가 필요 없습니다.	**도메인 전문성 요구:** 효과적인 규칙을 작성하려면 해당 분야(예: 법률, 의료)의 도메인 지식이 필요합니다.

주의사항

규칙 기반의 한계: 문맥(Context)의 부재

규칙 기반 방식의 가장 큰 한계는 **문맥을 이해하지 못한다**는 것입니다.

- 예시: "I bought an **Apple**." vs "I work at **Apple**."

규칙 기반 시스템은 이 두 "Apple"을 구분할 수 없습니다. 반면, 통계 기반 모델은 "bought"라는 단어(과일)와 "work at"라는 단어(회사)라는 문맥을 통해 이를 구분할 수 있습니다.

4 접근법 2: 통계 기반 NER (Statistical NER)

한 줄 정의: 대규모의 **정답(라벨링) 데이터**를 학습하여, 특정 단어 시퀀스가 개체명일 **확률(Probability)**을 계산하는 방식입니다.

직관적 비유: '수천 건의 사례를 학습한 탐정' 통계 기반 NER은 "수천 건의 사건 파일을 학습한 탐정"과 같습니다.

- 이 탐정은 명시적인 '규칙'에만 의존하지 않습니다.
- 대신, **문맥(Context)**과 **패턴(Pattern)**을 학습합니다.
- *"'Tesla'라는 단어가 'electric car', 'stock', 'CEO' 같은 단어 근처에 나오면, 99% 확률로 `ORG`(기관)이다."*
- *"'Tesla'라는 단어가 'physicist', 'invention', 'Wardenclyffe' 같은 단어 근처에 나오면, 98% 확률로 `PERSON`(사람)이다."*

이 방식은 새로운 데이터에도 유연하게 적응할 수 있지만, 왜 그렇게 판단했는지 정확히 설명하기는 어렵습니다. (블랙박스)

4.1 주요 모델 및 기술

통계 기반 NER은 전통적인 머신러닝 모델에서 딥러닝 모델로 발전해 왔습니다.

- **은닉 마르코프 모델 (Hidden Markov Models, HMMs):** 관찰된 단어(Observable) 뒤에 숨겨진(Hidden) 개체 태그(State)를 예측하는 초기 확률 모델입니다.
- **조건부 무작위장 (Conditional Random Fields, CRFs):** HMM보다 발전된 모델로, 단어 시퀀스 전체의 문맥을 고려하여 가장 확률이 높은 태그 시퀀스를 예측합니다. (오랫동안 NER의 표준으로 사용됨)
- **신경망 (Neural Networks, NN):** 최근의 NER 시스템은 대부분 딥러닝, 특히 순환 신경망(RNN, LSTM)이나 트랜스포머(BERT)를 사용합니다. 'spaCy'의 기본 모델 중 하나는 **CNN (Convolutional Neural Networks)**을 사용합니다.

4.2 통계 기반 NER의 장단점

Table 3: 통계 기반 *NER*의 장점과 단점

장점 (Pros)	단점 (Cons)
데이터 기반 학습 (유연성): 새로운 패턴이나 문맥을 데이터로부터 스스로 학습하여 적응합니다. (예: "Apple"을 문맥으로 구분)	**대규모 데이터 요구:** 높은 성능을 내기 위해 **잘 라벨링된(annotated)** 대량의 데이터가 필수적입니다.
도메인 적응성: 새로운 도메인(예: 의료)의 라벨링된 데이터를 제공하면 해당 도메인에 맞게 모델을 '학습' 또는 '미세조정'할 수 있습니다.	**블랙박스 (해석 불가):** 왜 모델이 "Tesla"를 `PERSON`으로 분류했는지 명확히 설명하기 어렵습니다.
높은 재현율(Recall): 규칙에 없는 새로운 개체명이라도 문맥이 비슷하다면 인식할 가능성이 높습니다.	**높은 컴퓨팅 비용:** 딥러닝 모델(예: BERT)을 학습시키려면 고사양의 GPU와 많은 시간이 필요합니다.

5 심층 분석: spaCy는 NER을 어떻게 수행하는가? (CNN 기반)

'spaCy'와 같은 최신 라이브러리는 어떻게 문맥을 파악하여 NER을 수행할까요? 강의에서는 **CNN(합성곱 신경망)**을 이용한 NER의 작동 원리를 설명합니다.

핵심 아이디어: 텍스트를 '이미지'로 바라보기

우리는 보통 텍스트를 '단어의 1차원 배열'로 생각합니다. 하지만 딥러닝 기반 NER은 텍스트를 **'2차원 이미지'**처럼 처리합니다.

1. **임베딩 (Embedding):** 각 단어를 고유한 숫자 벡터(예: 300차원)로 변환합니다.
2. **2D 변환:** "The cat sat on the mat" (5개 단어)라는 문장은 5×300 크기의 **행렬(Matrix)**이 됩니다.
3. **이미지 비유:** 이 5×300 행렬을 5픽셀(세로) $\times$ 300픽셀(가로) 크기의 흑백 '이미지'라고 상상할 수 있습니다.

5.1 CNN을 이용한 NER의 3단계

1단계: 입력 (텍스트 → 임베딩 행렬) 문장 "The cat sat on the mat"은 각 단어의 임베딩 벡터로 구성된 행렬(이미지)이 됩니다.

[Vector for "The"]
Vector for "cat"
Vector for "sat"
Vector for "on"
Vector for "mat"

2단계: 합성곱 (CNN 필터 적용) 이미지 처리에서 CNN 필터(커널)가 이미지의 '특징'(예: 선, 모서리)을 감지하듯, NER에서 CNN 필터는 **'문맥적 특징'**을 감지합니다.

- 3×300 크기의 필터(창문)가 이 '이미지'를 위에서 아래로 훑고 지나갑니다.
- 첫 번째 위치에서 "The", "cat", "sat"의 임베딩을 **동시에** 봅니다.
- 다음 위치에서 "cat", "sat", "on"의 임베딩을 **동시에** 봅니다.
- **이유:** 이 필터는 "sat"이라는 단어 하나만 보는 것이 아니라, 그 주변 단어("cat", "on")를 함께 봄으로써 **지역적 문맥(Local Context)**을 포착합니다.

3단계: 출력 (Softmax → 태그 예측) CNN을 통과한 결과는 **각 단어(토큰)마다** 모든 개체 유형에 대한 확률 벡터를 출력합니다.

□ 예제:

단어 "Tesla"에 대한 가상의 출력 확률:
문맥 1: "...physicist **Tesla** invented..."

- `PERSON`: 98%
- `ORG`: 1%
- `GPE`: 0.5%
- `O` (Other/없음): 0.5%
- → **최종 예측: PERSON**

문맥 2: "...electric car **Tesla** announced..."

- `PERSON`: 1%
- `ORG`: 97%
- `GPE`: 0.5%
- `O` (Other/없음): 1.5%
- → **최종 예측: ORG**

5.2 자주 묻는 질문 (FAQ)

Q: "Bank of England"처럼 여러 단어로 된 개체는 어떻게 인식하나요?

A: 훌륭한 질문입니다. 딥러닝 모델은 단순히 단어마다 태그를 붙이는 것이 아니라, **BIO 태깅 스킴 (Tagging Scheme)**을 사용합니다.

- **B** (Beginning): 개체가 시작되는 단어
- **I** (Inside): 개체 내부에 속하는 단어 (시작 아님)
- **O** (Outside): 개체에 속하지 않는 단어

"Bank of England"는 다음과 같이 태깅됩니다.

- Bank: **B-ORG** (기관 개체의 시작)
- of: **I-ORG** (기관 개체의 내부)
- England: **I-ORG** (기관 개체의 내부)

모델은 "B-ORG" 뒤에 "I-ORG"가 나올 확률을 학습하여 여러 단어로 구성된 개체를 하나의 덩어리(Chunk)로 묶습니다.

Q: "Tesla"라는 단어가 한 문서에 여러 번 나오면 어떻게 되나요?

A: NER은 문서 전체가 아닌, **각 토큰(단어)의 위치마다** 문맥을 평가합니다. 한 문서의 앞부분에서 "Tesla (`ORG`)"가 나왔더라도, 뒷부분에서 "Tesla (`PERSON`)"가 다른 문맥으로 나오면 각각 다르게 태깅할 수 있습니다. CNN 필터는 **지역 문맥(Local Context)**에 집중하기 때문입니다.

6 파이썬을 이용한 NER 구현

파이썬에서는 'NLTK'와 'spaCy'가 NER 및 관련 작업을 위해 널리 사용됩니다.

6.1 NLTK를 이용한 품사 판별 (POS Tagging)

NER에 앞서, 각 단어의 **품사(Part-of-Speech, POS)**를 판별하는 것은 문장의 구조를 이해하는 데 도움이 됩니다. NER이 '고유명사'를 찾는 작업이라면, POS는 '명사', '동사', '형용사' 등 일반적인 문법 요소를 찾습니다.

```python
import nltk
# 의NLTK 필요리소스다운로드최초    ( 회1)
# nltk.download('punkt')
# nltk.download('averaged_perceptron_tagger')

text = "From electric cars to solar panels, Mr. Musk is busy."

# 1. 문장을단어토큰 ()로 분리 (Word Tokenization)
tokens = nltk.word_tokenize(text)
# ['From', 'electric', 'cars', 'to', 'solar', 'panels', ',',
#  'Mr.', 'Musk', 'is', 'busy', '.']

# 2. 토큰에대해  POS Tagging 수행
pos_tags = nltk.pos_tag(tokens)

print(pos_tags)

# --- 출력결과튜플의  ( 리스트) ---
# [('From', 'IN'),          # IN: 전치사 (Preposition)
#  ('electric', 'JJ'),      # JJ: 형용사 (Adjective)
#  ('cars', 'NNS'),         # NNS: 명사, 복수형 (Noun, plural)
#  ('to', 'TO'),            # TO: 'to'
#  ('solar', 'JJ'),
#  ('panels', 'NNS'),
#  (',', ','),
#  ('Mr.', 'NNP'),         # NNP: 고유명사, 단수 (Proper noun, singular)
#  ('Musk', 'NNP'),
#  ('is', 'VBZ'),           # VBZ: 동사, 인칭3 단수현재  (Verb)
#  ('busy', 'JJ'),
#  ('.', '.')]
```

Listing 3: NLTK를 사용한 문장 토큰화 및 POS Tagging

6.2 spaCy를 이용한 개체명 인식 (NER)

'spaCy'는 현대적인 NLP 라이브러리로, 고도로 최적화된 통계 기반 NER 모델을 기본 제공합니다.

```python
import spacy
from spacy import displacy # 시각화도구

```

```python
# 1. 사전학습된 spaCy 모델로드영어 (, 스몰버전 )
# 설치 (: python -m spacy download en_core_web_sm)
nlp = spacy.load("en_core_web_sm")

# 2. 텍스트처리
# 의 spaCy nlp 객체는토큰화 , POS, NER 등전체파이프라인을실행합니다 .
text = """
    From electric cars to solar panels, Mr. Musk is busy.
    Tesla sells electric vehicles.
    Donald J. Trump is a person.
"""
doc = nlp(text)

# 3. 인식된개체 (Entities) 순회및출력
print("--- 인식된개체목록 ---")
for ent in doc.ents:
    # ent.text: 개체텍스트
    # ent.label_: 개체유형태그 ()
    print(f"Text: {ent.text}, Label: {ent.label_}")

# 4. (Jupyter웹/) 시각화
# displacy.render(doc, style="ent", jupyter=True)

# --- 출력결과 ---
# --- 인식된개체목록 ---
# Text: Musk, Label: PERSON
# Text: Tesla, Label: ORG
# Text: Donald J. Trump, Label: PERSON
```

Listing 4: spaCy를 사용한 NER 수행

주의사항

spaCy의 NLP 파이프라인

'doc = nlp(text)'라는 한 줄의 코드는 'spaCy' 내부에서 복잡한 **파이프라인(Pipeline)**을 실행합니다.

1. **Tokenizer:** 텍스트를 토큰으로 분리
2. **Tagger:** POS 태깅
3. **Parser:** 문장 구조 분석 (의존성 파싱)
4. **NER:** 개체명 인식
5. …(기타)

'doc' 객체에는 이 모든 분석 결과가 포함되어 있습니다.

7 NER 모델 미세조정 (Fine-Tuning)

사전 학습된 'spaCy' 모델이 내가 가진 특정 도메인(예: 자동차 산업)의 용어를 잘 인식하지 못할 수 있습니다. 예를 들어, "Honda Civic"을 ORG(기관)가 아닌 VEHICLE(차량)이라는 새로운 유형으로 인식하게 하고 싶을 수 있습니다.

이때, 새로운 데이터를 추가하여 기존 모델을 **추가 학습(Fine-Tuning)**시킬 수 있습니다.

미세조정 (Fine-Tuning)이란?

이미 수많은 데이터로 학습된 똑똑한 모델(Pre-trained model)을 가져와서, 내가 가진 **소량의 특정 데이터**를 **추가로 학습**시켜 내 입맛에 맞게 '조율'하는 과정입니다.

처음부터 모든 것을 학습(Training from scratch)하는 것보다 훨씬 효율적이고 적은 데이터로도 좋은 성능을 낼 수 있습니다.

□ 예제:

spaCy NER 모델에 'VEHICLE' 유형 추가 학습 시도 (개념적 예시)
다음 코드는 'spaCy' 모델의 'ner' 파이프라인만 선택적으로 비활성화하고, 새로운 'VEHICLE' 라벨이 포함된 데이터로 추가 학습(update)을 시도하는 과정을 보여줍니다.

```
import spacy
import random

# 1. 기존모델로드
nlp = spacy.load("en_core_web_sm")

# 2. 새로운학습데이터   (VEHICLE 유형추가 )
# 텍스트   (, 개체{ 목록: 시작[( 인덱스, 끝인덱스 , 라벨)]})
TRAIN_DATA = [
    ("Cars in China", {"entities": [(0, 4, "VEHICLE")]}),
    ("My family loves our Honda Civic", {"entities": [(20, 31, "VEHICLE
        ")]}),
    ("This car is the last one", {"entities": [(5, 8, "VEHICLE")]})
]

# 3. 파이프라인에서 'ner' 컴포넌트가져오기
ner = nlp.get_pipe("ner")

# 4. 새로운라벨 (VEHICLE)을 ner 컴포넌트에추가
ner.add_label("VEHICLE")

# 5. 다른파이프라인은비활성화하고   'ner만' 학습
other_pipes = [pipe for pipe in nlp.pipe_names if pipe != "ner"]
with nlp.disable_pipes(*other_pipes):
    optimizer = nlp.begin_training()
    for itn in range(20): # 회20 반복학습
        random.shuffle(TRAIN_DATA)
        losses = {}
        for text, annotations in TRAIN_DATA:
```

```
                doc = nlp.make_doc(text)
                example = spacy.training.Example.from_dict(doc, annotations
                    )
                nlp.update([example], sgd=optimizer, losses=losses)
            # print(f"Iteration {itn}, Losses: {losses}")

    # 6. 학습된모델로테스트
    doc = nlp("My family loves our Honda Civic. Tesla is a company.")
    for ent in doc.ents:
        print(f"Text: {ent.text}, Label: {ent.label_}")

    # --- 기대하는출력성공 ( 시) ---
    # Text: Honda Civic, Label: VEHICLE
    # Text: Tesla, Label: ORG
```

Listing 5: spaCy NER 모델 미세조정(Fine-Tuning) 시도

주의사항

미세조정의 위험: 파국적 망각 (Catastrophic Forgetting)
미세조정은 매우 적은 수의 예시(위 예제에서는 단 3개)로 시도할 경우, 모델이 새로운 데이터에 **과적합(overfitting)**되어 기존에 잘하던 것까지 잊어버리는 **'파국적 망각'** 현상이 발생할 수 있습니다. 실제 강의의 시연에서도, 위와 같이 'VEHICLE'을 학습시킨 후 **"Tesla"를 ORG가 아닌 PERSON으로 잘못 분류하는 오류**가 발생했습니다. 이는 모델이 "Tesla"를 학습 데이터에서 본 적이 없기 때문에, 새로운 학습 과정에서 가중치가 망가져 기존의 지식을 잃어버렸기 때문입니다.
해결책: 미세조정을 할 때는 새로운 데이터뿐만 아니라, **기존의 정답 예시(Original Examples)**도 함께 학습시켜야 합니다.

8 실전 응용: 텍스트 분류 모델에 NER 활용하기

NER은 그 자체로도 유용하지만, 다른 NLP 모델의 성능을 높이는 **'특성(Feature)'**으로 사용될 때 더욱 강력합니다.

목표: 뉴스 기사를 7개의 카테고리(예: 스포츠, 정치, 기술)로 분류하는 모델을 만든다고 가정합니다.

8.1 접근법 1: TF-IDF만 사용 (Baseline)

전통적인 방식은 TF-IDF 벡터를 만들어 모델(예: 로지스틱 회귀, 신경망)을 학습시키는 것입니다.

- **입력 데이터:** (기사 100개 × 단어 5000개) 크기의 TF-IDF 행렬
- **문제점:** 이 방식은 문맥을 잃어버립니다. "Apple"이라는 단어가 '기술' 기사에 중요한지, '요리' 기사에 중요한지 TF-IDF 값만으로는 알기 어렵습니다.

8.2 접근법 2: TF-IDF + NER 특성 (Enhanced)

기존 TF-IDF 특성에 **NER로 추출한 개체 정보를 추가**하여 모델을 더 '똑똑하게' 만들 수 있습니다.

작업 순서:

1. 모든 기사(예: 100개)에 대해 'spaCy'로 NER을 실행합니다.
2. 각 기사에 어떤 유형의 개체(`PERSON`, `ORG`, `GPE` 등)가 **존재하는지 여부**를 0 또는 1의 **더미 변수(Dummy Variable)**로 만듭니다.
3. 이 더미 변수들을 기존 TF-IDF 행렬에 **열(Column)로 추가**합니다.

Table 4: *NER* 특성 추가 전후의 특성 행렬(*Feature Matrix*) 비교

문서 (기사)	접근법 1: TF-IDF만 사용			접근법 2: TF-IDF + NER 특성			
	TF-IDF("Apple")	TF-IDF("Musk")	...	TF-IDF("Apple")	TF-IDF("Musk")	...	Has_ORG (NER)
기사 1: "Apple...CEO..."	0.35	0.0	···	0.35	0.0	···	**1**
기사 2: "Musk...Tesla..."	0.0	0.41	···	0.0	0.41	···	**1**
기사 3: "recipe...apple..."	0.28	0.0	···	0.28	0.0	···	**0**

결과:

- 이제 모델은 TF-IDF 값(`"Apple"` = 0.35)뿐만 아니라, **문맥 정보(`Has_ORG` = 1)**를 함께 볼 수 있습니다.
- 모델은 '기사 1'과 '기사 3'이 모두 "Apple"이라는 단어를 포함하지만, '기사 1'은 `ORG`(기관)와 관련이 있으므로 '기술' 또는 '비즈니스' 카테고리일 것이라고 학습할 수 있습니다.
- 이처럼 NER 특성을 추가하는 것은 모델에게 강력한 **'힌트'**를 제공하여, 특히 적은 데이터셋(예: 124개 기사)에서도 분류 성능을 향상시키는 데 도움을 줄 수 있습니다.

9 학습 체크리스트 및 FAQ

9.1 학습 내용 점검 체크리스트

NER의 정의를 "형광펜 비유"를 들어 설명할 수 있는가?

NER이 사용되는 3가지 주요 응용 분야(예: Q&A, 분류)를 말할 수 있는가?

규칙 기반 NER과 통계 기반 NER의 핵심 차이점(규칙 vs 문맥)을 설명할 수 있는가?

규칙 기반 NER의 장점(해석 가능)과 단점(유연성 부족)을 아는가?

통계 기반 NER의 장점(문맥 이해)과 단점(데이터 요구, 블랙박스)을 아는가?

'spaCy'의 'nlp(text)' 코드가 단순한 작업이 아닌 '파이프라인'임을 이해하는가?

텍스트 분류 모델의 성능을 향상시키기 위해 NER 결과를 어떻게 활용할 수 있는지(특성 행렬) 설명할 수 있는가?

NER 모델을 '미세조정'한다는 것의 의미와 그 위험성(파국적 망각)을 이해하는가?

9.2 초심자 FAQ

Q: NER과 POS Tagging은 같은 것 아닌가요? A: 다릅니다. **POS Tagging**은 일반 문법(명사, 동사, 형용사)을 찾는 것이고, **NER**은 고유한 이름(사람, 기관, 장소)을 찾는 것입니다.

- "Musk (`NNP`, 고유명사)" → (*POS Tagging*)
- "Musk (`PERSON`, 사람)" → (*NER*)

모든 `PERSON`은 `NNP`(고유명사)일 수 있지만, 모든 `NNP`가 `PERSON`은 아닙니다. (예: "Tesla"는 `NNP`이지만 `ORG`입니다.)

Q: "Tesla"가 사람 이름으로도, 회사 이름으로도 쓰이는데, 모델은 어떻게 구분하나요? A: **문맥(Context)**입니다. 통계 기반 모델은 "Tesla"라는 단어 자체보다 주변 단어를 더 중요하게 봅니다.

- "Tesla **invented** …" → '발명하다'와 어울리는 것은 `PERSON`입니다.
- "Tesla **sells** …" → '판매하다'와 어울리는 것은 `ORG`입니다.

Q: 제 데이터에는 'spaCy'가 잘 작동하지 않습니다. 어떻게 해야 하나요? A: 두 가지 방법이 있습니다.

1. **규칙 기반 추가:** 'spaCy'의 통계 모델이 놓친 부분을 정규표현식(Regex)을 사용한 규칙 기반으로 보완합니다. (하이브리드 접근)
2. **미세조정 (Fine-Tuning):** 내 도메인에 맞는 정답 데이터를 수백 수천 건 만들어서 'spaCy' 모델을 추가 학습시킵니다. (위험성 인지!)

A 부록: 강의 10 이전 퀴즈 리뷰 (핵심 개념 복습)

강의 10 본편(NER)에 앞서, 이전 강의들의 핵심 개념들에 대한 퀴즈 리뷰가 진행되었습니다. 다음은 복습을 위한 요약입니다.

A.1 A1: 나이브 베이즈 (Naive Bayes)

- **질문:** 나이브 베이즈 분류기의 '나이브(Naive, 순진한)'한 기본 가정은 무엇인가?
- **핵심: 특성(Feature) 간의 조건부 독립(Conditional Independence)**을 가정합니다.
- **쉬운 설명:** 스팸 메일을 분류할 때, "viagra"라는 단어의 등장과 "free"라는 단어의 등장이 (스팸이라는 조건 하에서) **서로 아무런 영향을 주지 않는다**고 '순진하게' 가정하는 것입니다.
- **현실:** 실제로는 "viagra"와 "free"는 함께 등장할 확률이 높습니다. (즉, 독립이 아닙니다.)
- **결론:** 이 가정은 비현실적이지만(Naive), 그럼에도 불구하고 나이브 베이즈는 종종 빠르고 준수한 성능을 보여줍니다.

A.2 A2: K-최근접 이웃 (K-Nearest Neighbors, KNN)

- **질문:** 키(cm)와 몸무게(kg)처럼 단위(Scale)가 다른 특성들을 KNN에 사용할 때 왜 문제가 되는가?
- **핵심:** KNN은 **유클리드 거리(Euclidean Distance)**를 기반으로 작동하기 때문입니다.
- **쉬운 설명:**
- 두 사람의 키 차이: 170cm vs 180cm → 차이 10 → 거리 계산 시 $10^2 = 100$
- 두 사람의 몸무게 차이: 70kg vs 71kg → 차이 1 → 거리 계산 시 $1^2 = 1$
- **문제점:** 키(cm)처럼 **값의 범위가 큰 특성**이 몸무게(kg) 같은 작은 특성보다 거리 계산에 **훨씬 더 큰 영향**을 미칩니다. 모델이 사실상 '키'만 보고 판단하게 됩니다.
- **해결책:** 모든 특성을 동일한 범위(예: 0 1)로 만드는 **정규화(Normalization)** 또는 **표준화(Standardization)** (피처 스케일링)가 필수적입니다.

A.3 A3: 로지스틱 회귀 (Logistic Regression)

- **질문:** 로지스틱 회귀에서 **최대가능도추정(Maximum Likelihood Estimation, MLE)**은 어떤 역할을 하는가?
- **핵심:** 우리가 가진 **관측 데이터(Observations)가 나타날 확률을 최대화**하는 파라미터(가중치)를 찾는 방법입니다.
- **쉬운 설명 (동전 던지기):**
- **데이터:** 동전을 10번 던졌더니 앞면(H) 7번, 뒷면(T) 3번이 나왔다.
- **질문:** 이 동전의 앞면이 나올 확률(P)은 얼마일까?
- **MLE 접근:**
- P=0.5라고 가정: $(0.5)^7 \times (0.5)^3$ → 매우 낮은 확률
- P=0.9라고 가정: $(0.9)^7 \times (0.1)^3$ → 낮은 확률
- P=0.7이라고 가정: $(0.7)^7 \times (0.3)^3$ → **가장 높은 확률 (Likelihood)**
- **결론:** "7번의 성공과 3번의 실패"라는 데이터를 관찰할 확률(Likelihood)을 **최대(Maximum)**로 만

드는 P는 0.7입니다. 로지스틱 회귀도 이와 같이, 주어진 데이터(X)와 라벨(Y=0 또는 1)이 관측될 확률을 최대로 만드는 최적의 가중치(W)를 찾습니다.

A.4 A4: 서포트 벡터 머신 (SVM)

- **질문:** SVM에서 **서포트 벡터(Support Vectors)**란 무엇인가?
- **핵심:** 두 클래스 간의 **경계(Decision Boundary)를 결정하는 데 직접적인 영향을 미치는** 데이터 포인트들입니다.
- **쉬운 설명:** 서포트 벡터는 두 나라 사이의 '국경선'에 가장 가까이 있는 '최전방 초소'들입니다.
- **특징:** 경계(마진)에서 멀리 떨어진 '후방'의 데이터 포인트들은 아무리 많이 추가되거나 이동해도 경계선에 영향을 주지 않습니다. 오직 이 '최전방 초소'들(서포트 벡터)만이 경계선을 결정합니다.
- **허용치(Allowance, C 파라미터):**
- **Hard Margin (C=무한대):** 단 하나의 오류도 허용하지 않음. 국경선(마진)이 매우 좁고, 아웃라이어에 민감함. (과적합 위험)
- **Soft Margin (C=작은 값):** 일부 데이터가 마진을 넘거나 잘못 분류되는 것을 '허용'함. 마진이 넓어지고, 아웃라이어에 덜 민감함. (일반화 성능 향상)

A.5 A5: 랜덤 포레스트 (Random Forest)

- **질문:** 단일 결정 트리(Decision Tree)에 비해 랜덤 포레스트가 갖는 주요 이점은 무엇인가?
- **핵심: 분산(Variance)을 줄여 일반화 성능을 높입니다.**
- **쉬운 설명:**
- **단일 트리:** 한 명의 '편협한' 전문가. 특정 데이터셋에 과적합(Overfitting)되기 쉬움. (분산이 높다)
- **랜덤 포레스트:** 수백 명의 '다양한' 전문가 집단. 각 전문가(트리)는 데이터를 무작위로 샘플링(배깅)하고, 질문(특성)도 무작위로 선택하여 학습합니다.
- **결론:** 각 트리는 조금씩 편향(Bias)되어 있지만, 이들의 예측을 **평균(Averaging) 또는 투표(Voting)**함으로써 개별 트리의 오류(분산)가 상쇄됩니다. 이는 모델 전체의 안정성과 일반화 성능을 크게 향상시킵니다.