December 10, 2025

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 01

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 01의 핵심 개념 학습

□ **핵심 요약**

본 문서는 하버드 익스텐션 스쿨의 CSCI E-89B 자연어 처리 입문 강의 내용을 통합 정리한 노트입니다.

- 강의 운영 방식, 평가 기준, 과제 제출법 등 **학사 정보**를 명확히 안내합니다.
- 딥러닝의 기초가 되는 **신경망(Neural Network)**의 개념을 다룹니다.
- 신경망의 핵심 구성 요소인 **활성화 함수(Activation Function)**의 종류와 역할을 설명합니다.
- 모델을 학습시키는 원리인 **손실/비용 함수(Loss/Cost Function)**와 **최적화 알고리즘(Optimization Algorithm)**을 배웁니다.
- 이론적 개념을 실제 코드로 구현하는 **Keras 예제**를 포함합니다.

Contents

1 학습 로드맵 및 핵심 용어

□ 핵심 정보

학습 로드맵

1. **기초 다지기:** 강의 운영 방식을 숙지하고, 신경망의 기본 아이디어를 이해합니다.
2. **핵심 개념:** 활성화 함수, 손실 함수, 비용 함수의 정의와 역할을 명확히 구분합니다.
3. **훈련 원리:** 경사 하강법(GD), 확률적 경사 하강법(SGD), 미니배치 경사 하강법의 차이를 비교하고 이해합니다.
4. **실습 적용:** Keras 코드를 통해 신경망을 구축하고, 다양한 손실 함수와 옵티마이저를 적용하는 방법을 익힙니다.
5. **심화:** 최종 프로젝트를 염두에 두고, 다양한 문제(회귀, 분류)에 어떤 함수와 알고리즘이 적합할지 고민합니다.

1.1 용어 정리

Table 1: 자연어 처리 및 신경망 핵심 용어

용어	쉬운 설명	원어	비고
신경망	인간의 뇌 신경을 모방한 데이터 학습 모델	Neural Network (NN)	입력, 은
활성화 함수	뉴런의 활성/비활성을 결정하는 비선형 함수	Activation Function	ReLU, S
손실 함수	**단일 데이터**에 대한 모델 예측과 실제 값의 차이	Loss Function	예: 제곱
비용 함수	**전체 데이터셋**에 대한 손실의 평균	Cost Function	훈련의
경사 하강법	비용 함수를 최소화하기 위해 파라미터를 업데이트하는 방법	Gradient Descent	기울기(
원-핫 인코딩	범주형 데이터를 0과 1의 벡터로 변환하는 기법	One-Hot Encoding	고양이
하이퍼파라미터	모델이 학습하지 않고, 사용자가 직접 설정하는 값	Hyperparameter	학습률,
에포크	전체 훈련 데이터셋을 한 번 모두 사용한 훈련 주기	Epoch	

2 강의 운영 및 평가

2.1 소통 채널 및 강의 세션

효율적인 학습과 소통을 위해 목적에 따라 다른 채널을 사용합니다.

소통 채널 가이드

- **Piazza (피아짜):** 강의 내용, 과제에 대한 **공식적인 질문**을 위한 공간입니다. 교수 및 조교(TA)가 답변하며, 다른 학생들과 질문과 답변을 공유할 수 있습니다.
- **WhatsApp (왓츠앱):** 학생들 간의 **자유로운 토론 및 정보 교류**를 위한 비공식 채널입니다. 교수나 조교가 항상 확인하지는 않으므로, 공식적인 답변이 필요하면 Piazza를 사용해야 합니다.
- **Canvas Inbox (캔버스 인박스):** 개인적인 사안에 대해 교수나 조교에게 직접 연락할 때 사용합니다.

Table 2: 주간 세션 일정

세션 종류	요일 (예상)	주요 내용
강의 (Lecture)	화요일	핵심 이론 및 개념 설명
조교 세션 1 (TA Session)	수요일 또는 목요일	이론 복습, 예제 풀이 (1)
강사 세션 (Instructor's Section)	금요일	Python 코드 구현 예제, 심화 토론
조교 세션 2 (TA Session)	토요일 또는 일요일	이론 복습, 예제 풀이 (2)

모든 세션은 녹화되어 제공되므로 실시간 참여가 어려워도 학습이 가능합니다. 단, 두 조교 세션은 서로 다른 내용을 다루므로 중복되지 않습니다.

2.2 평가 기준 및 정책

최종 성적은 과제, 퀴즈, 최종 프로젝트의 점수를 합산하여 산출됩니다.

Table 3: 성적 평가 비중

항목	비중
주간 과제 (Homework Assignments)	65%
주간 퀴즈 (Quizzes)	20%
최종 프로젝트 (Final Project)	15%

주의사항

제출 기한 정책

- **퀴즈: 절대 지각 제출 불가.** 퀴즈 마감 직후 다음 수업에서 해설이 진행되기 때문에, 시스템적으로 재응시 기회를 제공하기 어렵습니다. 7일의 충분한 기간 내에 제출해야 합니다.
- **과제:** 매주 일요일 23:59 (보스턴 시간 기준) 마감. 지각 제출 시 정해진 비율에 따라 감점됩니다 (1일 지각 시 10%, 2일 20% ... 5일 100%).
- **최종 프로젝트:** 마감 기한이 매우 엄격하며, 연장이 거의 불가능합니다. 특별한 사유가 있을 시, 사전에 Extension School을 통해 공식적인 절차를 밟아야 합니다.

이 강의에서는 가장 낮은 점수의 퀴즈나 과제를 제외하는 정책이 **적용되지 않습니다.**

2.3 과제 제출 가이드

과제는 코드와 보고서를 함께 제출해야 합니다.

과제 제출 체크리스트

1. **보고서 (Report) 작성:**
   - MS Word 또는 PDF 형식으로 제출합니다.
   - 문제 해결의 핵심이 되는 코드 일부, 결과(플롯, 표), 그리고 결과에 대한 간단한 논의를 포함해야 합니다.
   - Jupyter Notebook에서 직접 PDF로 변환하여 제출하는 것도 허용됩니다.
2. **소스 코드 (Source Code) 제출:**
   - 조교가 코드를 직접 실행하고 검토할 수 있도록 원본 코드를 반드시 제출해야 합니다. (예: `.ipynb` 파일)
   - 파일이 여러 개일 경우, 하나의 `.zip` 파일로 압축하여 제출합니다.
3. **최종 제출물 확인:** 보고서 파일 1개와 소스 코드 파일(또는 ZIP 파일) 1개를 모두 제출했는지 확인합니다.

3 신경망 입문 (Introduction to Neural Networks)

3.1 선형 회귀에서 신경망으로

전통적인 머신러닝 모델인 선형 회귀(Linear Regression)는 데이터의 패턴을 잘 표현하기 위해 사람이 직접 **특성(feature)**을 설계해야 했습니다. 예를 들어, 비선형 관계를 표현하기 위해 입력값 x 뿐만 아니라 x^2, x^3 같은 항을 직접 추가하는 방식입니다.

$$\hat{y} = w_0 + w_1 \cdot x + w_2 \cdot x^2 + w_3 \cdot x^3$$

하지만 이미지나 텍스트 같은 복잡한 데이터에서는 사람이 유의미한 특성을 직접 설계하기가 거의 불가능합니다. **신경망(Neural Network)**은 이러한 특성을 데이터로부터 **자동으로 학습**하는 모델입니다. 여러 개의 층(layer)을 쌓아, 데이터의 표현(representation)을 점진적으로 학습해 나갑니다.

3.2 순방향 신경망 (Feedforward Neural Network, FNN)

가장 기본적인 신경망 구조로, 입력층에서 출력층으로 정보가 한 방향으로만 흐릅니다. 수학적으로는 **함수들의 중첩(nested functions)**으로 표현할 수 있습니다.

$$\hat{y} = f^{(L)}(f^{(L-1)}(...f^{(1)}(x)))$$

여기서 각 함수 $f^{(l)}$은 보통 **비선형 활성화 함수**가 적용된 **선형 변환**의 형태를 가집니다. 예를 들어, 2개의 입력(x_1, x_2)과 1개의 은닉층(hidden layer)을 갖는 간단한 신경망의 출력 $\hat{y}$는 다음과 같이 계산됩니다.

$$\begin{aligned} u_1 &= f(w_{01}^{(1)} + w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2) \\ u_2 &= f(w_{02}^{(1)} + w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2) \\ \hat{y} &= f(w_0^{(2)} + w_1^{(2)} u_1 + w_2^{(2)} u_2) \end{aligned}$$

□ 핵심 정보

왜 중간에 선형 변환을 사용할까? 신경망의 핵심은 비선형성을 표현하는 것이지만, 각 단계에서 선형 변환$(w \cdot x + b)$을 사용하는 이유는 **미분 가능성** 때문입니다. 최적화 과정에서 경사 하강법을 사용하려면 모델을 파라미터로 미분해야 하는데, 선형 함수는 미분이 매우 간단합니다. 연쇄 법칙(chain rule)을 통해 복잡한 신경망 전체의 미분도 효율적으로 계산할 수 있습니다.

3.3 활성화 함수 (Activation Functions)

활성화 함수는 신경망에 비선형성(non-linearity)을 부여하는 핵심 요소입니다. 만약 활성화 함수가 없다면, 여러 층을 쌓더라도 결국 하나의 선형 변환과 같아져 복잡한 패턴을 학습할 수 없습니다.

주의사항

생물학적 뉴런과의 비유 활성화 함수라는 이름은 뇌의 생물학적 뉴런에서 유래했습니다. 뉴런은 여러 다른 뉴런으로부터 신호를 받아, 그 신호의 합이 특정 **임계값(threshold)**을 넘으면 **활성화(activate)**되어 다음 뉴런으로 신호를 전달합니다. 초기 인공 신경망은 이를 모방하여 계단 함수(step function)를 사용했지만, 이 함수는 미분이 불가능한 지점이 있어 경사 하강법에 적합하지 않습니다.

Table 4: 주요 활성화 함수 비교

함수	수식	특징	주요 용도
ReLU (Rectified Linear Unit)	$\max(0, x)$	계산이 빠르고, 널리 사용됨. 음수 입력에 대해 0을 출력.	은닉층(Hidden layers)
Leaky ReLU	$\max(0.1x, x)$	ReLU의 변형. 음수 입력에도 작은 기울기(0.1)를 가짐.	은닉층
Sigmoid	$\frac{1}{1+e^{-x}}$	출력을 (0, 1) 사이로 압축.	이진 분류(Binary classification) 출력층
Softmax	$\frac{e^{z_i}}{\sum_j e^{z_j}}$	다차원 입력의 출력을 합이 1인 확률 분포로 변환.	다중 클래스 분류(Multi-class) 출력층

```python
import keras
from keras import models, layers

model = models.Sequential()
# 은닉층 1: 개16 뉴런, ReLU 활성화함수
model.add(layers.Dense(16, activation='relu', input_shape=(900,)))
# 출력층: 개2 뉴런, Softmax 활성화함수다중  ( 분류)
model.add(layers.Dense(2, activation='softmax'))

model.summary()
```

Listing 1: Keras를 이용한 활성화 함수 지정 예시

4 신경망 훈련 (Training Neural Networks)

신경망 훈련의 목표는 모델의 예측값($\hat{y}$)과 실제 정답(y) 사이의 오차를 최소화하는 파라미터(가중치 w와 편향 b)를 찾는 것입니다. 이 과정은 **손실 함수**, **비용 함수**, **최적화 알고리즘** 세 가지 요소로 구성됩니다.

4.1 1단계: 손실 함수 (Loss Function) 정의

손실 함수는 **하나의 데이터 샘플**에 대한 모델의 오차를 측정하는 함수입니다. 어떤 문제를 푸느냐에 따라 적절한 손실 함수를 선택해야 합니다.

손실 함수(Loss) vs. 비용 함수(Cost)

- **손실 함수 (Loss Function):** 사과 하나가 얼마나 썩었는지 보는 것. 즉, 개별 데이터 포인트 $(x^{(i)}, y^{(i)})$ 하나에 대한 예측 오차 $L^{(i)}(w)$를 측정합니다.
- **비용 함수 (Cost Function):** 사과 상자 전체가 평균적으로 얼마나 썩었는지 보는 것. 즉, 전체 데이터셋에 대한 손실의 평균 $J(w) = \frac{1}{m}\sum_{i=1}^{m} L^{(i)}(w)$를 측정합니다.

훈련의 실제 목표는 이 **비용 함수**를 최소화하는 것입니다.

4.1.1 회귀(Regression) 문제의 손실 함수

연속적인 값을 예측하는 문제에 사용됩니다.

- **제곱 오차 (Squared Error):** $L(w) = (\hat{y} - y)^2$
  - 오차가 클수록 패널티를 더 크게 부여합니다.
  - 수학적으로 다루기 쉽고 미분이 용이하여 널리 쓰입니다.
- **절대 오차 (Absolute Error):** $L(w) = |\hat{y} - y|$
  - 이상치(outlier)에 덜 민감합니다.
  - 최솟값 지점에서 미분이 불가능한 단점이 있습니다.

4.1.2 분류(Classification) 문제의 손실 함수

데이터를 특정 카테고리로 분류하는 문제에 사용됩니다.

- **교차 엔트로피 (Cross-Entropy):** $L(w) = -\sum_{j=1}^{M} y_j \log(\hat{y}_j)$
  - 모델이 예측한 확률 분포($\hat{y}$)와 실제 레이블의 원-핫 인코딩 분포(y) 사이의 차이를 측정합니다.
  - 모델이 정답을 높은 확률로 맞추면 손실이 0에 가까워지고, 틀린 답을 높은 확률로 예측하면 손실이 무한대에 가깝게 커집니다.

주의사항

분류 문제에 제곱 오차를 쓰면 안 되는 이유 분류 문제의 레이블(예: [1, 0])과 모델의 확률 예측(예: [0.9, 0.1]) 사이에 제곱 오차를 사용하면, 비용 함수의 형태가 매우 비선형적이고 복잡해져 **나쁜 지역 최솟값(bad local minima)**에 빠지기 쉽습니다. 이는 최적화 과정을 매우 불안정하게 만들어 좋은 성능을 얻기 어렵게 합니다. 따라서 분류 문제에는 교차 엔트로피를 사용하는 것이 표준입니다.

4.2 2단계: 최적화 알고리즘 (Optimization Algorithm) 선택

최적화 알고리즘은 비용 함수 $J(w)$를 최소화하기 위해 파라미터 w를 반복적으로 업데이트하는 방법입니다. 대부분의 알고리즘은 **경사 하강법(Gradient Descent)**에 기반합니다.

Table 5: 주요 경사 하강법 알고리즘 비교

알고리즘	업데이트 단위	장점	단점
경사 하강법 (GD)	전체 데이터셋	안정적, 전역 최솟값 수렴(볼록 함수)	계산 비용 매우 높음, 지
확률적 경사 하강법 (SGD)	데이터 1개	계산 빠름, 지역 최솟값 탈출 용이	매우 불안정, 수렴 속도
미니배치 경사 하강법	데이터 N개 (배치)	GD와 SGD의 장점 절충, 병렬화 용이	배치 크기라는 하이퍼파

□ 핵심 정보

미니배치와 "수프 스푼" 비유 큰 솥에 끓고 있는 수프의 간을 볼 때, 수프 전체를 마실 필요도 없고, 숟가락 크기가 솥의 크기에 비례할 필요도 없습니다. 적당한 크기의 스푼 하나면 충분합니다. 마찬가지로, 거대한 데이터셋(수프 솥)의 전체적인 경사(간)를 추정할 때, 적절한 크기의 미니배치(스푼)만 사용해도 충분히 효율적입니다. 데이터셋이 100만 개든 1억 개든, 32나 64 크기의 미니배치가 효과적인 이유입니다.

파라미터 업데이트 규칙:

$$w_{\text{new}} := w_{\text{old}} - \alpha \nabla J(w_{\text{old}})$$

여기서 α는 **학습률(learning rate)**로, 얼마나 큰 보폭으로 이동할지를 결정하는 중요한 하이퍼파라미터입니다.

```
model.compile(
    optimizer='adam',  # Adam 옵티마이저사용
    loss='categorical_crossentropy', # 다중분류용교차엔트로피
    metrics=['accuracy'] # 훈련중정확도를모니터링
)

# 모델훈련
history = model.fit(
    X_train, y_train,
    batch_size=128, # 미니배치크기
    epochs=35,      # 전체데이터셋반복횟수
    validation_data=(X_test, y_test)
)
```

Listing 2: Keras에서 손실 함수와 옵티마이저 지정하기

`Adam`은 현재 가장 널리 쓰이는 진보된 옵티마이저 중 하나입니다.

5 1페이지 요약

핵심 개념 퀵 리뷰

1. 신경망의 기본 구조

- **역할:** 데이터로부터 특성을 자동으로 학습하는 모델.
- **구조:** 입력층 → 은닉층(들) → 출력층
- **원리:** 비선형 활성화 함수와 선형 변환의 중첩.

$$\hat{y} = f(W \cdot x + b)$$

2. 활성화 함수

- **역할:** 모델에 비선형성을 부여하여 복잡한 패턴 학습을 가능하게 함.
- **은닉층용:** **ReLU** ($\max(0, x)$)가 가장 보편적.
- **출력층용:**
  - 이진 분류: **Sigmoid**
  - 다중 분류: **Softmax**

3. 손실 함수와 비용 함수

- **손실 함수:** 단일 데이터의 오차 측정.
- **비용 함수:** 전체 데이터셋의 평균 손실.
- **회귀 문제:** 제곱 오차(MSE), 절대 오차(MAE)
- **분류 문제: 교차 엔트로피**

4. 최적화 알고리즘

- **목표:** 비용 함수를 최소화하는 파라미터 찾기.
- **핵심:** 경사 하강법.
- **실용적 선택: 미니배치 경사 하강법**
- **주요 하이퍼파라미터:** 학습률(α), 배치 크기.

December 10, 2025

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 02

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 02의 핵심 개념 학습

Abstract

본 문서는 텍스트 데이터를 수치로 변환하는 TF-IDF 기법부터 시작하여, 순차적 데이터 처리에 특화된 순환 신경망(RNN)의 기초와 한계, 그리고 이를 극복하기 위한 LSTM, GRU와 같은 고급 모델까지의 핵심 내용을 통합하여 정리합니다. 각 개념은 구체적인 계산 예시와 응용 사례를 통해 설명되며, 신경망 학습 과정에서 발생하는 주요 문제들과 해결책을 다룹니다. 이 자료는 단독으로 학습이 가능하도록 구성되었습니다.

Contents

Part I

텍스트 정량화와 분류

1 개요

자연어 처리의 첫 단계는 컴퓨터가 이해할 수 있도록 텍스트를 숫자 형태의 벡터로 변환하는 것입니다. 이 과정에서 가장 널리 사용되는 기법 중 하나가 **TF-IDF**입니다.

TF-IDF는 각 문서에서 특정 단어가 얼마나 중요한지를 나타내는 통계적 수치입니다. 이 수치를 이용해 텍스트의 내용을 정량화하고, 이를 바탕으로 문서 분류와 같은 머신러닝 작업을 수행할 수 있습니다.

본 파트에서는 TF-IDF의 원리를 계산 예시와 함께 살펴보고, 실제 특허 문서를 산업 분야별로 분류하는 응용 사례를 통해 데이터 준비부터 모델 학습까지의 전 과정을 학습합니다. 특히, 수많은 단어 중 분류에 실질적으로 도움이 되는 핵심 단어를 선별하기 위해 통계적 기법인 **t-검정(t-test)**을 활용하는 차원 축소 방법을 심도 있게 다룹니다.

2 TF-IDF를 이용한 텍스트 표현

2.1 TF-IDF의 개념

TF-IDF는 **Term Frequency(단어 빈도)**와 **Inverse Document Frequency(역문서 빈도)**라는 두 가지 값을 곱하여 계산됩니다.

- **TF (Term Frequency, 단어 빈도)**: 특정 문서 내에서 한 단어가 얼마나 자주 등장하는지를 나타냅니다. 자주 나올수록 해당 문서 내에서 중요할 가능성이 높습니다.
- **IDF (Inverse Document Frequency, 역문서 빈도)**: 전체 문서 집합에서 특정 단어가 얼마나 희귀한지를 나타냅니다. 여러 문서에 공통으로 등장하는 단어(예: a, the, and)는 중요도가 낮은 반면, 소수의 문서에만 나타나는 단어는 해당 문서의 주제를 잘 나타내므로 중요도가 높습니다.

TF-IDF 핵심 원리

한 문서 안에서는 자주 등장하지만(**TF가 높음**), 전체 문서들 중에서는 드물게 나타나는 단어(**IDF가 높음**)가 해당 문서를 대표하는 중요한 단어라는 아이디어에 기반합니다.

2.2 TF-IDF 계산 방법

간단한 예시를 통해 TF-IDF 계산 과정을 단계별로 살펴보겠습니다. 3개의 문서가 있고, 첫 번째 문서에서 'cat'이라는 단어의 TF-IDF를 계산해 봅시다.

- **문서 1**: "the cat set on the mat" (총 6개 단어)
- **문서 2**: "the cat did something again"
- **문서 3**: "some sentence but no word"

1. **TF(cat, 문서 1) 계산**: 문서 1에서 단어 'cat'의 등장 횟수는 1번이고, 전체 단어 수는 6개입니다.

$$\text{TF}(\text{'cat'}, \text{문서 } 1) = \frac{\text{'cat'의 등장 횟수}}{\text{문서 } 1\text{의 전체 단어 수}} = \frac{1}{6}$$

2. **IDF(cat) 계산**: 전체 3개의 문서 중 'cat'이 포함된 문서는 2개입니다. IDF는 이 비율에 역수를 취해 계산합니다.

$$\text{IDF('cat')} = \log\left(\frac{\text{전체 문서 수}}{\text{'cat'을 포함한 문서 수} + 1}\right)$$

여기서 분모에 1을 더하는 것은 특정 단어가 모든 문서에 등장하여 분모가 0이 되는 것을 방지하는 **스무딩(smoothing)** 기법입니다. 로그를 취하는 것은 값의 스케일을 조절하기 위함입니다.

$$\text{IDF('cat')} = \log\left(\frac{3}{2+1}\right) \text{ 또는 간단히 } \log\left(\frac{3}{2}\right)$$

계산 방식에는 여러 변형이 존재하며, 라이브러리마다 기본값이 다를 수 있습니다.

3. **TF-IDF(cat, 문서 1) 계산**: TF와 IDF 값을 곱하여 최종 점수를 얻습니다.

$$\text{TF-IDF('cat', 문서 1)} = \text{TF('cat', 문서 1)} \times \text{IDF('cat')} = \frac{1}{6} \times \log\left(\frac{3}{2}\right)$$

실무적 고려사항

정규화(Normalization): 실제 라이브러리에서는 계산된 TF-IDF 벡터의 크기를 1로 만드는 정규화 과정을 거칩니다. 이는 문서 길이에 따른 편향을 줄여줍니다.
학습/테스트 데이터 분리: 모델 학습 시, IDF 값은 **학습 데이터(train data)만으로 계산**해야 합니다. 테스트 데이터에 적용할 때는 이 학습된 IDF 값을 그대로 가져와 사용합니다. 이는 테스트 데이터 정보가 모델 학습에 미리 유출되는 것을 막기 위함입니다.

3 응용 사례: 특허 데이터 분류

3.1 프로젝트 목표 및 데이터 준비

이 프로젝트의 목표는 1895년부터 1935년 사이의 미국 특허 문서를 **자동차 산업 관련 특허**와 **비관련 특허**로 자동 분류하는 것입니다.

데이터 소스 특허 제목(title)과 본문(description) 텍스트 데이터.

학습 데이터 생성
- **레이블 1 (자동차 관련)**: 당시 자동차를 생산했던 기업 목록을 확보하여, 해당 기업들이 출원한 특허를 모두 자동차 관련으로 간주하고 레이블 '1'을 부여합니다.
- **레이블 0 (비관련)**: 전체 특허 풀에서 무작위로 샘플을 추출하여 레이블 '0'을 부여합니다. 당시 수많은 분야의 특허가 존재했으므로, 무작위 샘플은 대부분 자동차와 무관할 것이라는 가정에 기반합니다.

데이터 레이블링의 한계

이 방식은 100% 정확하지 않습니다. 자동차 회사가 아닌 독립 연구자가 출원한 관련 특허는 '0'으로 잘못 분류될 수 있고, 무작위 샘플에 우연히 자동차 관련 특허가 포함될 수도 있습니다. 하지만 대규모 데이터를 다룰 때 현실적인 접근 방식입니다.

3.2 차원 축소: t-검정(t-test)의 활용

특허 문서에는 수만 개 이상의 고유한 단어가 존재합니다. 이 모든 단어를 모델의 입력 피처(feature)로 사용하면 계산량이 방대해지고, 모델 성능이 저하되는 **차원의 저주(curse of dimensionality)** 문제가 발생합니다.

따라서 분류에 실질적으로 도움이 되는 핵심 단어들만 선별하는 과정이 필요합니다. 이때 **t-검정(t-test)**을 활용할 수 있습니다.

[title=t-검정(t-test)] 두 집단의 평균값에 통계적으로 유의미한 차이가 있는지를 검정하는 방법입니다. 여기서는 특정 단어('engine' 등)의 TF-IDF 점수가 '자동차 관련 특허 집단(레이블 1)'과 '비관련 특허 집단(레이블 0)' 사이에서 평균적으로 차이가 나는지를 확인합니다.

3.2.1 t-검정 기반 피처 선택 절차

1. **단어별 t-검정 수행**: 모든 고유 단어에 대해 다음을 수행합니다.
   - 집단 1: 자동차 관련 특허들에서 해당 단어의 TF-IDF 점수 분포
   - 집단 2: 비관련 특허들에서 해당 단어의 TF-IDF 점수 분포
   - 두 집단의 평균 TF-IDF 점수 차이에 대한 t-검정을 실시합니다.
2. **p-값(p-value) 확인**: t-검정 결과로 p-값을 얻습니다.
   - **p-값이 작다**: 두 집단 간 평균 차이가 우연히 발생했을 확률이 낮다는 의미입니다. 즉, 해당 단어는 두 집단을 구분하는 데 **매우 유의미**합니다. (예: 'engine', 'wheel')
   - **p-값이 크다**: 두 집단 간 평균 차이가 뚜렷하지 않다는 의미입니다. 해당 단어는 분류에 별 도움이 되지 않습니다. (예: 'the', 'is', 'and')
3. **핵심 단어 선택**: 모든 단어의 p-값을 오름차순으로 정렬한 뒤, p-값이 가장 작은 상위 N개(예: 300개)의 단어만 최종 피처로 선택합니다.

t-검정을 이용한 차원 축소의 효과

t-검정을 통해 'and', 'of'와 같이 분류에 기여하지 못하는 일반적인 단어들은 제거하고, 'engine', 'chassis', 'vehicle'처럼 특정 도메인을 강력하게 나타내는 단어들만 선별할 수 있습니다. 이를 통해 모델의 성능과 효율을 크게 향상시킬 수 있습니다.

Part II

신경망 기초와 순환 신경망(RNN)

4 신경망의 기본 연산

신경망은 입력 데이터로부터 복잡한 패턴을 학습하여 예측 결과를 출력하는 모델입니다. 이 과정은 크게 **순전파(Forward Propagation)**와 **역전파(Backward Propagation)** 두 단계로 이루어집니다.

4.1 순전파 (Forward Propagation)

순전파는 입력 데이터가 신경망의 각 층(layer)을 순서대로 거쳐 최종 출력값($\hat{y}$)을 계산하는 과정입니다. 각 뉴런은 이전 층의 출력에 가중치(w)를 곱하고 편향(b)을 더한 뒤, 활성화 함수(f)를 적용하여 다음 층으로 신호를 전달합니다.

□ 예제: title

입력 $x_1 = 1.3, x_2 = 0.7$이고, 활성화 함수가 $\mathrm{ReLU}(f(z) = \max(0, z))$일 때의 계산 과정입니다.

1. 첫 번째 은닉층(Hidden Layer) 계산

- 첫 번째 뉴런(u_1):

$$\begin{aligned} z_1 &= w_{01}^{(1)} + w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 \\ &= -1.2 + (0.1 \times 1.3) + (0.5 \times 0.7) = -0.72 \\ u_1 &= f(z_1) = \mathrm{ReLU}(-0.72) = 0 \end{aligned}$$

- 두 번째 뉴런(u_2):

$$\begin{aligned} z_2 &= w_{02}^{(1)} + w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 \\ &= 0.9 + (0.8 \times 1.3) + (0.3 \times 0.7) = 2.15 \\ u_2 &= f(z_2) = \mathrm{ReLU}(2.15) = 2.15 \end{aligned}$$

2. 출력층(Output Layer) 계산

- 최종 출력($\hat{y}$):

$$\begin{aligned} z_{out} &= w_0^{(2)} + w_1^{(2)} u_1 + w_2^{(2)} u_2 \\ &= 0.2 + (0.8 \times 0) + (1.2 \times 2.15) = 2.78 \\ \hat{y} &= f(z_{out}) = \mathrm{ReLU}(2.78) = 2.78 \end{aligned}$$

이처럼 입력에서 출력 방향으로 차례대로 값을 계산해 나가는 것이 순전파입니다.

4.2 역전파 (Backward Propagation)

역전파는 신경망의 예측값($\hat{y}$)과 실제값(y) 사이의 오차(손실, loss)를 줄이기 위해 각 가중치(w)를 어떻게 조정해야 하는지 계산하는 과정입니다.

핵심 원리는 **연쇄 법칙(Chain Rule)**을 사용하여 손실 함수를 각 가중치로 미분한 값, 즉 **기울기(gradient)**를 구하는 것입니다. 이 기울기는 '오차를 가장 빠르게 줄일 수 있는 방향'을 알려줍니다.

역전파의 핵심

역전파는 출력층에서부터 입력층 방향으로, 순전파 과정에서 계산했던 중간값들을 재활용하여 효율적으로 기울기를 계산합니다. 계산된 기울기는 경사 하강법(Gradient Descent)을 통해 가중치를 업데이트하는 데 사용됩니다.

4.3 가중치와 하이퍼파라미터

[title=파라미터 vs. 하이퍼파라미터]

파라미터 (Parameter) 모델이 학습 과정에서 데이터로부터 스스로 학습하는 변수입니다. **가중치**(w)와 **편향**(b)이 여기에 해당합니다.

하이퍼파라미터 (Hyperparameter) 모델이 학습을 시작하기 전에 사용자가 직접 설정해야 하는 값입니다. **학습률(learning rate)**, 은닉층의 수, 뉴런의 수, 옵티마이저 종류 등이 해당됩니다. 이 값들은 모델의 학습 방식과 최종 성능에 큰 영향을 미칩니다.

5 신경망 학습의 핵심: 경사 하강법

경사 하강법은 손실 함수의 기울기를 이용해 점진적으로 손실이 최소가 되는 지점의 파라미터 값을 찾아가는 최적화 알고리즘입니다. 가중치 업데이트 규칙은 다음과 같습니다.

$$w_{\text{new}} = w_{\text{old}} - \alpha \times \nabla J(w)$$

여기서 α는 **학습률(learning rate)**이며, 한 번의 업데이트에서 얼마나 큰 폭으로 이동할지를 결정하는 중요한 하이퍼파라미터입니다.

5.1 학습률(α)의 중요성

학습률을 어떻게 설정하느냐에 따라 모델의 학습 속도와 안정성이 크게 달라집니다.

학습률이 너무 작은 경우

현상: 가중치 업데이트 폭이 매우 작아 손실이 거의 줄어들지 않고 학습이 정체됩니다.
결과: 최적점에 도달하는 데 시간이 매우 오래 걸리거나, 학습이 조기에 멈춘 것처럼 보일 수 있습니다. 손실 곡선이 거의 수평선을 그립니다.

학습률이 적절한 경우

현상: 손실이 꾸준히 감소하며 안정적으로 최적점을 찾아갑니다.
결과: 가장 효율적으로 모델을 학습시킬 수 있습니다. 손실 곡선이 부드러운 하강 곡선을 그립니다.

학습률이 너무 큰 경우

현상: 업데이트 폭이 너무 커서 최적점을 지나쳐 버리는 **오버슈팅(overshooting)**이 발생합니다.
결과: 손실 값이 줄어들지 않고 진동하거나 오히려 발산(divergence)하여 학습이 실패할 수 있습니다. 손실 곡선이 위아래로 크게 요동치거나 급격히 증가합니다.

데이터 스케일링의 중요성

경사 하강법이 잘 동작하려면 입력 피처들의 스케일을 비슷하게 맞춰주는 것이 매우 중요합니다. 예를 들어 어떤 피처는 0 1 사이의 값을 갖고, 다른 피처는 100만 단위의 값을 갖는다면 학습이 불안정해집니다. **표준화(Standardization)**나 **정규화(Normalization)**를 통해 모든 입력 데이터의 범위를 비슷하게 만들어주면, 기본 학습률 값으로도 안정적인 학습이 가능해집니다.

[title=옵티마이저 (Optimizer)] Adam, RMSprop과 같은 고급 옵티마이저는 학습 과정에서 학습률을 자동으로 조절해주는 기능을 포함하고 있습니다. 이를 통해 사용자가 학습률을 직접 세밀하게 튜닝하는 수고를 덜어주고, 더 빠르고 안정적인 수렴을 돕습니다.

Part III

순차 데이터 처리를 위한 순환 신경망(RNN)

6 순환 신경망(RNN)의 등장 배경

문장, 주가 데이터, 음성 신호와 같이 **시간적 순서**가 중요한 데이터를 **순차 데이터(Sequential Data)**라고 합니다. 기존의 완전 연결 신경망(Fully Connected Neural Network)은 각 입력이 독립적이라고 가정하기 때문에, 데이터의 순서 정보를 효과적으로 처리하기 어렵습니다.

순환 신경망(Recurrent Neural Network, RNN)은 이러한 한계를 극복하기 위해 설계되었습니다. RNN은 내부에 **'기억' 혹은 '상태(state)'**를 유지하는 순환 구조를 가지고 있어, 이전 시점(time step)의 정보를 현재 시점의 계산에 반영할 수 있습니다.

RNN의 핵심 아이디어

RNN은 이전 단계의 출력을 현재 단계의 입력으로 다시 사용하는 **되먹임 루프(feedback loop)** 구조를 가집니다. 이를 통해 순차적인 정보의 흐름을 모델링하고, 시간적 의존성을 학습할 수 있습니다.

7 기본 RNN (Vanilla RNN) 구조

7.1 RNN의 작동 원리

RNN은 순차 데이터의 각 요소를 시간 단계별로 처리합니다.

1. **t=1 시점**: 첫 번째 입력(x_1)과 초기 은닉 상태(h_0, 보통 0으로 설정)를 받아 첫 번째 은닉 상태(h_1)를 계산합니다.
2. **t=2 시점**: 두 번째 입력(x_2)과 이전 은닉 상태(h_1)를 받아 두 번째 은닉 상태(h_2)를 계산합니다.
3. 이 과정을 데이터의 마지막 요소까지 반복합니다.

각 시점 t에서의 은닉 상태 h_t는 다음과 같이 계산됩니다.

$$h_t = f(W_h h_{t-1} + W_x x_t + b)$$

여기서 f는 활성화 함수(주로 tanh)이며, W_h, W_x, b는 모든 시간 단계에서 동일하게 사용되는 **공유 파라미터(shared parameters)**입니다. 이 파라미터 공유 덕분에 RNN은 입력 시퀀스의 길이에 관계없이 모델을 학습시킬 수 있습니다.

□ 예제: title

RNN의 순환 구조는 시간의 흐름에 따라 네트워크를 길게 펼쳐놓은 형태로 시각화할 수 있습니다. 이는 마치 동일한 구조의 네트워크 층이 시간 단계별로 연결된 것처럼 보입니다.

```
h_0=0 --> [RNN Cell] -- h_1 --> [RNN Cell] -- h_2 --> ...
             ^                     ^
             |                     |
             x_1                   x_2
```

각 `[RNN Cell]`은 동일한 가중치(W_h, W_x)를 공유합니다.

7.2 RNN의 파라미터 수 계산

하나의 RNN 층(layer)에 있는 파라미터의 수는 입력 벡터의 차원, 은닉 상태 벡터의 차원, 그리고 뉴런의 수에 따라 결정됩니다.

계산 공식:

$$\begin{aligned}\text{파라미터 수} = (\text{입력 차원} \times \text{뉴런 수}) \quad &- \text{입력 가중치 } W_x \\ +(\text{이전 은닉 상태 차원} \times \text{뉴런 수}) \quad &- \text{순환 가중치 } W_h \\ +\text{뉴런 수} \quad &- \text{편향 } b\end{aligned}$$

□ 예제: title

입력 피처가 2개(x_t가 2차원 벡터)이고, RNN 층에 3개의 뉴런이 있다고 가정해 봅시다. 이 경우 은닉 상태 h_t는 3차원 벡터가 됩니다.

- 입력 가중치: $2 \times 3 = 6$개
- 순환 가중치: $3 \times 3 = 9$개
- 편향: 3개
- **총 파라미터 수**: $6 + 9 + 3 = 18$개

8 RNN의 한계: 기울기 문제

RNN은 이론적으로는 긴 시퀀스를 처리할 수 있지만, 실제 학습 과정에서는 심각한 문제에 직면합니다. 바로 역전파 과정에서 발생하는 **기울기 소실(Vanishing Gradient)**과 **기울기 폭주(Exploding Gradient)** 문제입니다.

8.1 기울기 소실 (Vanishing Gradient)

원인 역전파 시 기울기는 연쇄 법칙에 따라 여러 번의 곱셈 연산을 거칩니다. RNN에서는 활성화 함수(예: tanh)의 미분값이 1보다 작은 경우가 많기 때문에, 이 값들이 시간 단계를 거슬러 올라가며 반복적으로 곱해지면 기울기가 기하급수적으로 작아져 거의 0에 가까워집니다.

결과 시퀀스의 앞부분에 있는 정보로부터 온 기울기가 거의 사라져, 해당 정보가 모델 파라미터 업데이트에 거의 영향을 미치지 못하게 됩니다. 이로 인해 RNN은 문장의 시작 부분 단어나 오래전의 주가 데이터와 같은 **장기 의존성(long-term dependency)**을 학습하기 매우 어려워집니다.

8.2 기울기 폭주 (Exploding Gradient)

원인 기울기 소실과 반대로, 미분값이 1보다 큰 값들이 반복적으로 곱해지면 기울기가 기하급수적으로 커져 무한대에 가까워질 수 있습니다.

결과 파라미터 업데이트가 비정상적으로 크게 일어나 모델 학습이 불안정해지고, 결국 발산하게 됩니다. 이 문제는 **기울기 클리핑(Gradient Clipping)**이라는 기법을 통해, 기울기 값이 특정 임계치를 넘으면 강제로 잘라내어 어느 정도 해결할 수 있습니다.

기본 RNN의 근본적 한계

기울기 소실 문제 때문에, 기본 RNN(Vanilla RNN)은 실제 문제에서 긴 시퀀스의 의존 관계를 효과적으로 학습하지 못합니다. 이러한 한계를 극복하기 위해 LSTM과 GRU 같은 개선된 구조가 제안되었습니다.

Part IV

장기 의존성 문제 해결을 위한 고급 RNN 모델

9 LSTM (Long Short-Term Memory)

LSTM은 기본 RNN의 장기 의존성 학습 문제를 해결하기 위해 설계된 정교한 구조입니다. 핵심은 **셀 상태(Cell State)**와 3개의 **게이트(Gate)**를 도입하여 정보의 흐름을 효과적으로 제어하는 데 있습니다.

LSTM의 핵심 비유

LSTM의 셀 상태(c_t)를 '정보 고속도로'라고 생각할 수 있습니다. 이 고속도로를 따라 정보가 거의 변하지 않고 쭉 전달될 수 있어, 오래전 정보도 손실 없이 보존됩니다. 게이트들은 이 고속도로에 정보를 올리거나(**입력 게이트**), 내리거나(**삭제 게이트**), 또는 현재 정보를 외부에 보여줄지(**출력 게이트**)를 결정하는 '톨게이트' 역할을 합니다.

9.1 LSTM의 주요 구성 요소

셀 상태 (Cell State, c_t) LSTM의 핵심으로, 장기 기억을 담당합니다. 시퀀스를 따라 정보가 큰 변화 없이 전달될 수 있는 통로 역할을 합니다.

삭제 게이트 (Forget Gate) 이전 셀 상태(c_{t-1})에서 어떤 정보를 잊어버릴지(버릴지) 결정합니다. 시그모이드 함수를 통해 0(모두 잊음)에서 1(모두 기억) 사이의 값을 출력하여, 이전 정보에 곱해집니다.

입력 게이트 (Input Gate) 새로운 입력 정보(x_t) 중 어떤 것을 셀 상태에 저장할지 결정합니다. 시그모이드 함수로 저장할 정보의 비율을 정하고, tanh 함수로 새로운 후보 정보를 생성한 뒤 두 값을 곱해 셀 상태에 더합니다.

출력 게이트 (Output Gate) 현재 셀 상태를 바탕으로 어떤 정보를 외부로 출력하고, 다음 시점의 은닉 상태(h_t)로 넘겨줄지를 결정합니다.

이러한 게이트 구조 덕분에 LSTM은 기울기 소실 문제에 훨씬 강인하며, 수백 개의 시간 단계를 넘어선 장기 의존성도 성공적으로 학습할 수 있습니다.

10 GRU (Gated Recurrent Unit)

GRU는 LSTM의 복잡한 구조를 간소화하면서도 유사한 성능을 내는 모델입니다. LSTM의 셀 상태와 은닉 상태를 하나의 **은닉 상태**(h_t)로 통합하고, 게이트 수도 2개로 줄였습니다.

리셋 게이트 (Reset Gate) 이전 은닉 상태의 어느 부분을 무시할지를 결정합니다. 이는 새로운 입력을 기반으로 한 새로운 기억을 만드는 데 영향을 줍니다.

업데이트 게이트 (Update Gate) LSTM의 삭제 게이트와 입력 게이트 역할을 동시에 수행합니다. 이전 상태의 정보를 얼마나 유지하고, 새로운 정보를 얼마나 반영할지 결정합니다.

[title=LSTM vs. GRU] **구조**: LSTM은 3개의 게이트와 별도의 셀 상태를 가지지만, GRU는 2개의 게이트와 통합된 은닉 상태를 가집니다.
파라미터: GRU가 LSTM보다 파라미터 수가 적어 계산 효율이 높고, 데이터가 적을 때 과적합의 위험이 적을 수 있습니다.

성능: 대부분의 문제에서 두 모델의 성능은 비슷하지만, 문제의 특성이나 데이터의 양에 따라 어느 한쪽이 더 나을 수 있습니다. 일반적으로는 LSTM이 더 복잡한 패턴을 학습할 수 있는 잠재력이 있습니다.

11 양방향 RNN (Bidirectional RNN)

문장 번역이나 감성 분석과 같은 문제에서는 특정 단어의 의미가 앞뒤 문맥에 모두 의존하는 경우가 많습니다. 예를 들어, "나는 그 영화가 정말 재미____"라는 문장에서 마지막 단어를 예측하려면 앞의 내용뿐만 아니라, 문장 끝에 "없었다"가 오는지 "있었다"가 오는지를 알아야 합니다.

기본적인 RNN은 과거 정보만을 이용하므로 이러한 양방향 문맥을 파악할 수 없습니다. **양방향 RNN(Bidirectional RNN, Bi-RNN)**은 이 문제를 해결하기 위해 고안되었습니다.

11.1 Bi-RNN의 구조와 원리

Bi-RNN은 내부적으로 두 개의 독립적인 RNN 층을 가집니다.

1. **정방향 RNN (Forward RNN)**: 입력 시퀀스를 원래 순서대로 (예: $x_1, x_2, \ldots, x_T$) 처리합니다.
2. **역방향 RNN (Backward RNN)**: 입력 시퀀스를 역순으로 (예: $x_T, \ldots, x_2, x_1$) 처리합니다.

각 시간 단계 t에서, Bi-RNN의 최종 출력은 정방향 RNN의 은닉 상태($\overrightarrow{h_t}$)와 역방향 RNN의 은닉 상태($\overleftarrow{h_t}$)를 **연결(concatenate)**하여 만들어집니다.

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$$

이를 통해 모델은 각 시점에서 과거와 미래의 정보를 모두 활용하여 더 풍부한 문맥적 표현을 학습할 수 있습니다.

□ 예제: title

Keras에서는 `Bidirectional` 래퍼(wrapper)를 사용하여 간단하게 양방향 모델을 구현할 수 있습니다.

```
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense, Bidirectional

model = Sequential()
model.add(Embedding(input_dim=200, output_dim=32))
model.add(Bidirectional(LSTM(32))) # LSTM 층을로 Bidirectional 감싸기
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

Listing 1: Keras를 이용한 양방향 LSTM 모델 구축

위 코드에서 양방향 LSTM 층의 출력 차원은 (None, 64)가 됩니다. 이는 정방향 LSTM(32차원)과 역방향 LSTM(32차원)의 출력이 연결되었기 때문입니다.

Part V

FAQ 및 요약

12 자주 묻는 질문 (FAQ)

Q. 에포크(Epoch), 배치(Batch), 이터레이션(Iteration)의 차이는 무엇인가요?

A. 세 용어는 모델 학습 단위를 나타냅니다.

- **에포크 (Epoch)**: 전체 학습 데이터셋을 한 번 모두 사용했을 때 1 에포크가 됩니다.
- **배치 (Batch)**: 전체 데이터를 한 번에 처리하기엔 너무 크므로, 작은 묶음으로 나눕니다. 이 묶음 하나를 배치라고 합니다.
- **이터레이션 (Iteration)**: 하나의 배치를 처리하여 가중치를 한 번 업데이트하는 것을 1 이터레이션이라고 합니다. 즉, (총 데이터 수 / 배치 크기) 만큼의 이터레이션이 1 에포크가 됩니다.

Q. 학습 정확도(Training Accuracy)는 계속 오르는데, 검증 정확도(Validation Accuracy)는 정체되거나 떨어집니다. 왜 그런가요?

A. 이는 모델이 학습 데이터에만 너무 과하게 맞춰져 새로운 데이터에 대한 일반화 성능이 떨어지는 **과적합(Overfitting)**의 전형적인 신호입니다. 모델이 데이터의 실제 패턴이 아닌 노이즈까지 암기하기 시작했다는 의미입니다. 이 경우, 검증 정확도가 가장 높았던 시점에서 학습을 조기 종료(Early Stopping)하거나, 드롭아웃(Dropout)과 같은 규제(Regularization) 기법을 적용해야 합니다.

Q. 데이터를 스케일링(scaling)하는 것이 왜 중요한가요?

A. 신경망의 학습 알고리즘인 경사 하강법은 각 피처(feature)의 스케일에 민감합니다. 만약 피처들의 값 범위가 크게 다르면(예: 나이는 0-100, 소득은 수천만 단위), 손실 함수의 표면이 한쪽으로 길게 찌그러진 타원형이 됩니다. 이런 경우 최적점을 찾아가는 경로가 매우 비효율적인 지그재그 형태가 되어 학습이 느려지거나 불안정해집니다. 모든 피처의 스케일을 비슷하게 맞춰주면(예: 0 1 사이로 정규화) 손실 함수 표면이 원형에 가까워져, 훨씬 빠르고 안정적으로 최적점을 찾을 수 있습니다.

13 한 페이지 요약

TF-IDF

문서 내 단어의 중요도를 평가하는 지표. **단어 빈도(TF)**와 **역문서 빈도(IDF)**의 곱으로 계산. 문서의 핵심 키워드를 추출하고 텍스트를 벡터로 변환하는 데 사용.

순전파 & 역전파

순전파: 입력에서 출력으로 값을 계산하는 과정. **역전파**: 출력의 오차를 바탕으로, 가중치를 업데이트하기 위해 기울기를 계산하는 과정.

학습률 (Learning Rate)

경사 하강법에서 가중치를 업데이트하는 보폭(step size). 너무 작으면 학습이 느리고, 너무 크면 학습이 발산할 수 있음. Adam과 같은 옵티마이저는 이를 자동으로 조절.

순환 신경망 (RNN)

순서가 있는 데이터를 처리하기 위한 신경망. 이전 시점의 **은닉 상태(hidden state)**를 현재 계산에 활용하여 시간적 의존성을 학습. **가중치 공유**가 특징.

기울기 소실 & 폭주

RNN의 고질적인 문제. 긴 시퀀스에서 역전파 시 기울기가 0 또는 무한대로 수렴하여 **장기 의존성** 학습을 방해함.

LSTM & GRU

기울기 소실 문제를 해결하기 위한 RNN의 변형. **게이트(Gate)** 메커니즘을 도입하여 정보의 흐름을 제어하고, 장기 기억을 효과적으로 보존함.

양방향 RNN (Bi-RNN)

시퀀스를 정방향과 역방향으로 모두 처리하여 각 시점에서 **과거와 미래의 문맥**을 모두 활용하는 모델. 언어 처리와 같이 양방향 문맥이 중요한 작업에서 성능이 높음.

December 10, 2025

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 03

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 03의 핵심 개념 학습

Contents

1 개요: 자연어 처리란 무엇인가?

□ 핵심 요약

자연어 처리(Natural Language Processing, NLP)는 컴퓨터가 인간의 언어를 이해하고, 해석하며, 생성하게 만드는 기술 분야입니다. 이 분야는 언어학(Linguistics)과 인공지능(Artificial Intelligence, AI)이 만나는 지점에 있습니다. 궁극적인 목표는 인간과 컴퓨터가 보다 자연스럽게 소통하는 것입니다. NLP는 텍스트 분류, 기계 번역, 챗봇 등 다양한 응용 분야의 핵심 기술입니다. 성공적인 NLP 모델을 구축하려면 텍스트를 컴퓨터가 이해할 수 있는 형태로 가공하는 전처리 과정이 매우 중요합니다.

1.1 인공지능, 머신러닝, 딥러닝, NLP의 관계

이 네 가지 개념은 종종 혼용되지만, 포함 관계를 이해하는 것이 중요합니다.

- **인공지능(AI)**: 가장 넓은 개념으로, 인간의 지능을 모방하는 모든 기술을 포함합니다. 여기에는 규칙 기반 시스템(rule-based system)처럼 전문가가 직접 규칙을 코딩하는 방식도 포함됩니다.
- **머신러닝(ML)**: AI의 한 분야로, 데이터로부터 컴퓨터가 스스로 '규칙'을 학습하게 하는 접근 방식입니다. 데이터를 입력하면 모델이 패턴을 찾아내 예측이나 분류를 수행합니다.
- **딥러닝(DL)**: 머신러닝의 한 분야로, 인간의 뇌 신경망을 모방한 심층 신경망(Deep Neural Network)을 사용합니다. 특히 이미지, 음성, 텍스트와 같은 비정형 데이터 처리에서 뛰어난 성능을 보입니다.
- **자연어 처리(NLP)**: AI의 한 분야로, 인간 언어에 특화된 기술입니다. NLP 문제를 해결하기 위해 규칙 기반 접근법, 전통적인 머신러닝, 그리고 최근에는 딥러닝 방법론이 활발히 사용됩니다.

개념 간의 관계 (벤 다이어그램 설명)

가장 큰 원인 **인공지능(AI)** 안에 **머신러닝(ML)**이 포함됩니다.
머신러닝 원 안에 **딥러닝(DL)**이 더 작은 원으로 포함됩니다.
자연어 처리(NLP)는 AI의 큰 원 안에 있으면서, 머신러닝과 딥러닝 영역과 상당 부분 겹칩니다.
이는 NLP가 AI의 다양한 기술을 활용한다는 것을 의미합니다.

2 핵심 용어 정리

용어	쉬운 설명	원어	비고
토큰화	문장을 단어나 문장 같은 의미 있는 작은 단위(토큰)로 쪼개는 과정	Tokenization	NLP 전처리의 가장 첫 단계
어간 추출	단어의 접사(접두사, 접미사)를 제거하여 어간(stem)을 추출하는 과정	Stemming	빠르지만, 결과가 사전에 없는 단어일 수 있음 ('running' → 'run')
표제어 추출	단어의 문법적 형태를 고려하여 기본형, 즉 표제어(lemma)를 찾는 과정	Lemmatization	문맥을 파악해야 하므로 느리지만, 결과가 사전에 있는 단어임 ('driving' → 'drive')
임베딩	단어를 저차원의 밀집된 숫자 벡터(dense vector)로 변환하는 기술	Embedding	단어 간의 의미적 관계를 벡터 공간에 표현. 원-핫 인코딩의 단점을 보완.
원-핫 인코딩	단어 사전에 있는 단어 중 하나만 1이고 나머지는 모두 0인 희소 벡터(sparse vector)로 표현하는 방식	One-Hot Encoding	단어 수가 많아지면 벡터 차원이 커지고, 단어 간 유사성을 표현하지 못함.
불용어	문장에서 큰 의미를 갖지 않아 분석에 불필요한 단어 (조사, 관사 등)	Stop Words	전처리 과정에서 제거하여 계산 효율성을 높임. (예: a, the, is, 의, 는)
OOV	훈련 데이터의 단어 사전에 없어 알 수 없는 단어	Out-of-Vocabulary	테스트 시 새로운 단어가 나타날 때를 대비해 특별 토큰으로 처리.

3 핵심 개념 및 원리

3.1 텍스트 전처리(Text Preprocessing)의 중요성

컴퓨터는 텍스트를 그대로 이해할 수 없습니다. 따라서 모델에 입력하기 전에 텍스트를 숫자 형태의 데이터로 변환하고 정제하는 과정이 필수적입니다. 이 과정을 '전처리'라고 부릅니다.

- **목표**: 분석에 불필요한 노이즈를 제거하고, 텍스트의 핵심 정보를 보존하며, 모델이 학습하기 좋은 형태로 데이터를 가공하는 것.
- **주요 단계**: 토큰화(Tokenization) → 정제(Cleaning, 예: 불용어 제거) → 정규화(Normalization, 예: 어간/표제어 추출) → 벡터화(Vectorization).

3.1.1 1. 토큰화 (Tokenization)

텍스트를 최소 단위인 '토큰(token)'으로 분할하는 과정입니다. 어떤 단위를 토큰으로 삼을지에 따라 여러 기법이 있습니다.

- **단어 토큰화 (Word Tokenization)**: 띄어쓰기나 구두점을 기준으로 텍스트를 단어 단위로 나눕니다.

  □ **예제:**

  입력: "Henry Ford's innovation, the assembly line."
  결과: `['Henry', 'Ford', "'s", 'innovation', ',', 'the', 'assembly', 'line', '.']`

- **문장 토큰화 (Sentence Tokenization)**: 마침표(.), 물음표(?) 등 문장 끝을 나타내는 기호를 기준으로 텍스트를 문장 단위로 나눕니다.

  □ **예제:**

  입력: "He created assembly lines. This revolutionized production."
  결과: `['He created assembly lines.', 'This revolutionized production.']`

- **서브워드 토큰화 (Subword Tokenization)**: 단어를 의미 있는 더 작은 단위(subword)로 나눕니다. OOV(Out-of-Vocabulary) 문제를 완화하는 데 효과적입니다.

3.1.2 2. 어간 추출 (Stemming)과 표제어 추출 (Lemmatization)

'running', 'runs', 'ran'과 같은 단어들은 형태는 다르지만 '달리다'라는 동일한 의미를 갖습니다. 이렇게 다양한 형태의 단어들을 하나의 기본 형태로 통일하여 단어 사전의 크기를 줄이고 모델의 일반화 성능을 높일 수 있습니다.

어간 추출 vs. 표제어 추출 비교

특징	어간 추출 (Stemming)	표제어 추출 (Lemmatization)
목표	단어의 어미를 잘라내어 어간(기본 형태)을 찾음	단어의 사전적 기본형(표제어)을 찾음
방식	규칙 기반으로 접사를 기계적으로 제거	품사 등 문맥 정보를 활용하여 사전을 참조
속도	빠름	느림
결과	사전에 없는 단어가 될 수 있음	항상 사전에 있는 단어
예시	"driving" → "driv"	"driving" → "drive"
	"transportation" → "transport"	"was" → "be"
	"electric" → "electr"	"cars" → "car"

언제 무엇을 쓸까?

- **어간 추출**: 검색 엔진의 인덱싱처럼 속도가 중요하고, 결과 단어의 언어적 정확성이 덜 중요한 경우.
- **표제어 추출**: 챗봇이나 기계 번역처럼 결과의 의미적 정확성이 매우 중요한 경우.

3.1.3 3. 임베딩 (Embedding)

텍스트를 벡터로 만드는 과정에서, 원-핫 인코딩은 단어 수만큼 차원이 커지고 단어 간 유사성을 표현하지 못하는 한계가 있습니다. 임베딩은 이러한 문제를 해결하기 위해 등장했습니다.

- **핵심 아이디어**: 단어를 고차원의 희소 벡터(sparse vector)에서 저차원의 밀집 벡터(dense vector)로 변환하는 것.
- **과정**:
  1. 단어 사전에 있는 각 단어에 대해 고유한 인덱스(정수)를 부여합니다.
  2. 신경망에 '임베딩 레이어'를 추가합니다. 이 레이어는 각 인덱스를 입력받아 특정 차원(예: 128차원)의 벡터로 매핑합니다.
  3. 이 매핑 가중치(lookup table)는 모델이 훈련하는 과정에서 다른 가중치들과 함께 학습됩니다. 결과적으로 의미가 비슷한 단어들은 벡터 공간에서 서로 가까운 위치에 배치됩니다.
- **장점**:
  - **차원 축소**: 수만 개의 차원을 수백 개의 차원으로 줄여 계산 효율성을 높입니다.
  - **의미 학습**: 단어의 문맥적 의미를 벡터에 담을 수 있습니다. (예: `king - man + woman ≈ queen`)
  - **훈련 가능**: 임베딩 벡터 자체가 모델의 파라미터로서 특정 과제에 맞게 최적화됩니다.

□ 예제: title

'female'과 'male'이라는 두 단어가 있다고 가정해봅시다.

- **원-핫 인코딩**: female: `[1, 0]`, male: `[0, 1]` (2차원)
- **임베딩 (1차원으로 축소)**: female: `[1]`, male: `[0]`

수만 개의 단어가 있는 실제 문제에서는, 10000차원의 원-핫 벡터를 128차원의 임베딩 벡터로 변환하는 것과 같습니다. 이 변환 과정은 훈련을 통해 최적의 값을 찾습니다.

4 NLP 텍스트 분류 절차

다음은 `20 newsgroups` 데이터셋을 사용하여 '하키(hockey)'와 '판매(for sale)' 관련 게시물을 분류하는 모델을 구축하는 일반적인 절차입니다.

1. **데이터 로드 및 준비**
   - 훈련(train) 데이터와 테스트(test) 데이터를 로드합니다.
   - 분류할 카테고리(예: `rec.sport.hockey`, `misc.forsale`)를 지정합니다.
2. **텍스트 전처리 및 단어 사전 구축**
   - **토큰화**: 모든 훈련 텍스트를 단어 토큰으로 분할합니다.
   - **정규화 (선택)**: 어간 추출(stemming)이나 표제어 추출(lemmatization)을 적용하여 단어를 기본 형태로 통일합니다.
   - **빈도 계산**: 각 단어의 등장 빈도를 계산합니다.
   - **단어 사전 생성**: 가장 빈도가 높은 N개(예: 10,000개)의 단어만 선택하여 단어 사전을 만듭니다.
   - **OOV 처리**: 사전에 없는 단어(Out-of-Vocabulary)를 처리하기 위해 특별 토큰(예: 인덱스 0)을 예약합니다.
3. **텍스트를 시퀀스로 변환**
   - 각 텍스트(게시물)를 단어 사전에 따라 정수 인덱스의 시퀀스로 변환합니다.
   - 예: "the game was fun" → `[5, 120, 25, 8]`
   - OOV 단어는 예약된 인덱스(예: 0)로 변환합니다.
4. **패딩 (Padding)**
   - 신경망 모델에 입력하려면 모든 시퀀스의 길이가 동일해야 합니다.
   - 최대 길이(예: 500)를 정하고, 이보다 짧은 시퀀스는 뒤쪽에 특정 값(보통 0)을 채워 길이를 맞춥니다.
5. **신경망 모델 구축**
   - **임베딩 레이어**: 정수 인덱스를 입력받아 밀집 벡터로 변환합니다. (입력 차원: 단어 사전 크기, 출력 차원: 임베딩 차원)
   - **순환 신경망 (RNN/LSTM) 레이어**: 시퀀스 데이터의 시간적 패턴을 학습합니다.
   - **드롭아웃 (Dropout) 레이어**: 과적합(overfitting)을 방지하기 위해 훈련 중 일부 뉴런을 무작위로 비활성화합니다.
   - **출력 레이어**: 최종적으로 분류 결과를 출력합니다. (이진 분류의 경우, Sigmoid 활성화 함수를 사용하는 하나의 뉴런)
6. **모델 훈련 및 평가**
   - 모델을 컴파일합니다. (손실 함수: `binary`$_c$*rossentropy*, : `adam`)
   - 준비된 훈련 데이터로 모델을 훈련시킵니다.
   - 훈련이 끝난 후, 테스트 데이터로 모델의 성능(예: 정확도)을 평가합니다.

5 실습 코드 및 결과 분석

5.1 1. NLTK를 이용한 토큰화 및 어간 추출

5.1.1 토큰화 (Tokenization)

NLTK(Natural Language Toolkit) 라이브러리를 사용하면 단어 및 문장 토큰화를 쉽게 수행할 수 있습니다. [title=NLTK 단어 토큰화 코드]

```
import nltk
from nltk.tokenize import word_tokenize

# NLTK 데이터다운로드최초 ( 회1 필요)
nltk.download('punkt')

text = "Henry Ford's innovation, the assembly line process, changed the car industry's dynamics profoundly."

# 단어로토큰화
word_tokens = word_tokenize(text)
print(word_tokens)

# 결과:
# ['Henry', 'Ford', "'s", 'innovation', ',', 'the', 'assembly', 'line', 'process', ',', 'changed', 'the', 'car', 'industry', "'s", 'dynamics', 'profoundly', '.']
```

Listing 1: NLTK를 사용한 단어 토큰화

5.1.2 어간 추출 (Stemming)

가장 널리 쓰이는 Porter Stemmer와 Snowball Stemmer를 사용하여 토큰화된 단어들에 어간 추출을 적용할 수 있습니다. [title=NLTK 어간 추출 코드]

```
from nltk.stem import PorterStemmer, SnowballStemmer

words = ['Henry', 'Ford', "'s", 'innovation', 'changed', 'industry', 'dynamics', 'profoundly']

# Porter Stemmer 적용
porter = PorterStemmer()
porter_stems = [porter.stem(word) for word in words]
print("Porter Stemmer:", porter_stems)
# 결과: ['henri', 'ford', "'s", 'innov', 'chang', 'industri', 'dynam', 'profoundli']

# Snowball Stemmer 적용 (보다Porter 개선됨)
snowball = SnowballStemmer("english")
snowball_stems = [snowball.stem(word) for word in words]
print("Snowball Stemmer:", snowball_stems)
```

```
# 결과: ['henri', 'ford', "'s", 'innov', 'chang', 'industri', 'dynam', '
    profound']
```

Listing 2: Porter 및 Snowball Stemmer 적용

주의사항

결과에서 볼 수 있듯이, 어간 추출은 단어를 `industri`나 `profoundli`처럼 사전에 존재하지 않는 형태로 만들 수 있습니다. 이는 단순히 규칙에 따라 접미사를 제거하기 때문입니다. `Henry`가 `Henri`로 변하는 등 고유명사에도 적용될 수 있습니다.

5.2 2. SpaCy를 이용한 표제어 추출

SpaCy는 산업 수준의 성능을 제공하는 NLP 라이브러리로, 정확한 표제어 추출 기능을 제공합니다.
[title=SpaCy 표제어 추출 코드]

```
import spacy

# SpaCy 모델로드최초 ( 회1 다운로드필요 )
# python -m spacy download en_core_web_sm
nlp = spacy.load("en_core_web_sm")

text = "Henry Ford's innovation changed the car industry's dynamics profoundly
    ."
doc = nlp(text)

# 표제어추출
spacy_lemmas = [token.lemma_ for token in doc]
print(spacy_lemmas)

# 결과:
# ['Henry', 'Ford', "'s", 'innovation', 'change', 'the', 'car', 'industry', "'
    s", 'dynamic', 'profoundly', '.']
```

Listing 3: SpaCy를 사용한 표제어 추출

결과 비교: Stemming vs. Lemmatization

- `changed` → `chang` (Porter Stemmer) vs. `change` (SpaCy Lemmatizer) - `dynamics` → `dynam` (Porter Stemmer) vs. `dynamic` (SpaCy Lemmatizer)
표제어 추출이 문법적으로 더 올바르고 해석 가능한 결과를 제공함을 알 수 있습니다.

5.3 3. 텍스트 분류 모델 결과 분석

20 `newsgroups` 데이터셋으로 훈련한 LSTM 모델의 테스트 정확도는 전처리 방식에 따라 미세한 차이를 보였습니다.

- **단순 토큰화만 적용**: 약 93.5%의 테스트 정확도
- **어간 추출(Stemming) 적용**: 약 97%의 테스트 정확도

- **표제어 추출(Lemmatization) 적용**: 약 95-96%의 테스트 정확도

결과 해석

이 특정 과제에서는 어간 추출을 적용했을 때 성능이 가장 좋았습니다. 이는 단어의 다양한 변형을 하나의 형태로 통일함으로써 모델이 더 적은 수의 특징으로 핵심 패턴을 학습할 수 있었기 때문일 수 있습니다. 표제어 추출은 어간 추출보다 더 정교하지만, 이로 인한 계산 비용 증가나 미미한 성능 차이로 인해 특정 상황에서는 어간 추출이 더 효율적일 수 있습니다.

결론: 어떤 전처리 기법이 최적인지는 데이터와 과제에 따라 다르므로, 여러 방법을 실험하고 검증 데이터셋에서의 성능을 비교하여 결정하는 것이 좋습니다.

6 자주 묻는 질문 (FAQ)

Q1: 어간 추출과 표제어 추출 중 항상 더 좋은 방법이 있나요?

A: 아니요, 항상 더 좋은 방법은 없습니다. 작업의 목표에 따라 선택이 달라집니다.

- **속도가 중요하다면** (예: 대규모 문서 인덱싱) 어간 추출이 더 나은 선택일 수 있습니다.
- **의미의 정확성이 중요하다면** (예: 챗봇, 기계 번역) 표제어 추출이 필수적입니다.

실제 프로젝트에서는 두 가지 방법을 모두 시도해보고 성능이 더 잘 나오는 쪽을 선택하는 경우가 많습니다.

Q2: 왜 원-핫 인코딩 대신 임베딩을 사용해야 하나요?

A: 두 가지 주된 이유가 있습니다.

1. **차원의 저주 회피**: 원-핫 인코딩은 단어 수가 많아지면 벡터의 차원이 수만, 수십만으로 커져 계산이 비효율적입니다. 임베딩은 이를 수백 차원의 밀집 벡터로 압축합니다.
2. **의미 관계 학습**: 원-핫 벡터들은 모두 서로 직교하므로 단어 간 유사도를 계산할 수 없습니다. 임베딩은 훈련 과정에서 비슷한 의미의 단어들을 벡터 공간상에 가깝게 배치하여 의미적 관계를 학습합니다.

Q3: OOV(사전에 없는 단어)는 왜 중요하고 어떻게 처리하나요?

A: 훈련 데이터에 없던 단어가 테스트 데이터에 나타나면 모델은 이를 처리할 수 없어 오류가 발생하거나 성능이 저하됩니다. 이것이 OOV 문제입니다.

- **처리 방법**: 단어 사전을 만들 때 '알 수 없는 단어'를 의미하는 특별 토큰(예: `<UNK>` 또는 `<OOV>`)을 추가합니다. 그리고 훈련 시에도 일부러 드물게 나타나는 단어들을 이 토큰으로 대체하여 모델이 OOV 상황에 대처하도록 학습시킬 수 있습니다. 테스트 시 사전에 없는 단어가 나오면 이 특별 토큰으로 매핑하여 처리합니다.

Q4: 드롭아웃(Dropout)은 왜 사용하나요?

A: 드롭아웃은 모델의 **과적합(Overfitting)**을 방지하기 위한 정규화(regularization) 기법입니다. 과적합은 모델이 훈련 데이터에만 너무 과도하게 맞춰져서, 새로운 데이터(테스트 데이터)에 대해서는 성능이 떨어지는 현상을 말합니다.

- **작동 원리**: 훈련 과정에서 각 미니배치마다 신경망의 뉴런 중 일부를 확률적으로 비활성화(출력을 0으로 만듦)합니다. 이를 통해 모델이 특정 뉴런에 과도하게 의존하는 것을 막고, 여러 뉴런이 협력하여 더 강건한(robust) 특징을 학습하도록 유도합니다.

7 한눈에 보는 핵심 요약

NLP 기본 개념

자연어 처리 (NLP)
컴퓨터가 인간의 언어를 다루게 하는 AI의 한 분야. 언어학과 컴퓨터 과학의 교차점.

텍스트 전처리 3대장

1. **토큰화 (Tokenization)** 문장을 단어/문장 등 작은 단위로 쪼개기.
2. **어간 추출 (Stemming)** 단어 끝을 잘라 기본형 찾기. 빠르지만 부정확할 수 있음.
3. **표제어 추출 (Lemmatization)** 사전을 이용해 진짜 기본형(표제어) 찾기. 정확하지만 느림.

단어의 벡터화: 원-핫 인코딩 vs. 임베딩

	원-핫 인코딩	**임베딩**
형태	희소 벡터 (대부분 0)	밀집 벡터 (의미 있는 실수값)
차원	고차원 (단어 수만큼)	저차원 (사용자 지정, 예: 128)
의미 표현	불가능 (모든 단어 독립적)	가능 (비슷한 단어는 벡터 공간에서 가까움)
결론	간단하지만 한계 명확	차원 축소 및 의미 학습에 효과적

NLP 텍스트 분류 파이프라인

데이터 로드 → 토큰화 → 정규화 (Stem/Lemma) → 정수 인코딩 → 패딩 → 모델 훈련 (Embedding+RNN) → 평가

핵심 모델 구성 요소

- **LSTM (Long Short-Term Memory)**: 순서가 중요한 시퀀스 데이터(문장 등)를 잘 처리하는 RNN의 한 종류.
- **Dropout**: 훈련 데이터에 모델이 과하게 맞춰지는 과적합을 방지하는 기술.
- **이진 교차 엔트로피 (Binary Cross-Entropy)**: 두 개 중 하나를 맞추는 이진 분류 문제에서 주로 사용하는 손실 함수.

December 10, 2025

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 04

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 04의 핵심 개념 학습

Contents

CSCI E-89B 자연어 처리 4주차 노트

Bag of Words, n-grams, and CNN

Contents

1 개요: 텍스트를 숫자로 바꾸는 여정

□ 핵심 요약

핵심 요약 이번 강의에서는 텍스트를 컴퓨터가 이해할 수 있는 숫자 벡터로 변환하는 방법을 다룹니다. 가장 기본적인 **Bag of Words**부터 시작하여, 단어의 순서를 일부 고려하는 **n-grams**를 배웁니다. 마지막으로, 이미지 처리에 주로 쓰이는 **Convolutional Neural Networks (CNN)**를 텍스트에 적용하는 원리를 탐구합니다. 이러한 기법들은 텍스트 분류, 감성 분석 등 다양한 자연어 처리(NLP) 문제의 기초가 됩니다. 궁극적으로는 텍스트의 의미적, 구조적 정보를 어떻게 효과적으로 포착할 것인가에 대한 고민이 담겨 있습니다.

□ 예제:

학습 로드맵

1. **기초 다지기**: Bag of Words (BoW)의 개념과 한계를 명확히 이해합니다.
2. **문맥 추가하기**: n-grams가 BoW의 어떤 단점을 보완하는지 파악합니다.
3. **고급 모델 맛보기**: 텍스트를 이미지처럼 다루는 CNN의 아이디어를 이해합니다.
4. **실습으로 체득하기**: Python 라이브러리(`sklearn`, `NLTK`, `spaCy`)를 사용해 BoW와 n-grams를 직접 구현해봅니다.
5. **개념 연결하기**: BoW의 희소성(sparsity) 문제를 해결하기 위한 대안으로 임베딩(embedding)의 필요성을 인식합니다.

2 핵심 용어 정리

자주 사용되는 전문 용어를 미리 익혀두면 학습에 도움이 됩니다.

Table 1: *4*주차 핵심 용어

용어	쉬운 설명	원어	비고
Bag of Words	문장의 단어 순서를 무시하고, 단어의 출현 빈도수만 가방에 담듯이 세는 방법	Bag of Words (BoW)	간단하지만 문맥 정보를 잃어버림
토큰화	문장을 의미 있는 단위(토큰)로 쪼개는 과정	Tokenization	단어, 글자, 서브워드(subword) 등이 토큰이 될 수 있음
n-gram	텍스트에서 연속적으로 나타나는 n개의 단어 묶음	n-gram	2-gram(bigram), 3-gram(trigram) 등이 있음. 지역적 문맥 포착
어간 추출	단어에서 접사(prefix, suffix)를 제거하여 기본형(어간)을 찾는 과정	Stemming	빠르지만, 결과가 실제 단어가 아닐 수 있음 (예: octopi → octop)
표제어 추출	단어의 사전적 기본형(표제어)을 찾는 과정	Lemmatization	문법적 품사를 고려하여 더 정확하지만, 어간 추출보다 느림
임베딩	단어를 의미를 담은 저차원의 조밀한(dense) 벡터로 표현하는 기법	Embedding	단어 간의 의미적 유사성을 벡터 공간의 거리로 표현 가능
CNN	이미지의 지역적 패턴을 추출하는 데 특화된 딥러닝 모델	Convolutional Neural Network	텍스트에 적용 시, n-gram처럼 지역적 단어 패턴을 학습
필터 (커널)	CNN에서 특정 특징(예: 수직선, 특정 단어 조합)을 감지하는 가중치 행렬	Filter (Kernel)	필터를 입력 데이터에 슬라이딩하며 특징 맵(feature map)을 생성
패딩	필터 연산 시 출력 크기가 줄어드는 것을 막기 위해 입력 데이터 주변을 특정 값(주로 0)으로 채우는 것	Padding	`VALID`(패딩 없음) vs `SAME`(출력 크기 유지)
스트라이드	필터가 입력 데이터 위를 한 번에 이동하는 칸의 수	Strides	스트라이드가 크면 출력 크기가 더 많이 줄어듦
풀링	특징 맵의 크기를 줄여(downsampling) 계산량을 감소시키고, 주요 특징을 강조하는 과정	Pooling	Max Pooling은 특정 영역에서 가장 큰 값만 남김

December 10, 2025

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 05

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 05의 핵심 개념 학습

Contents

□ 핵심 요약

개요: 이 노트의 핵심 이 문서는 자연어 처리의 핵심적인 두 가지 텍스트 표현 기법을 다룹니다.
첫째, **TF-IDF**는 어떤 단어가 특정 문서 내에서는 자주 나타나지만, 전체 문서 집합에서는 드물게 나타날수록 중요하다고 판단하는 가중치 계산 방법입니다.
둘째, **단어 임베딩(Word Embeddings)**은 단어를 의미가 풍부한 저차원의 실수 벡터로 표현하여 단어 간의 의미적, 문법적 관계를 포착하는 기법입니다.
이를 통해 컴퓨터가 단순히 단어의 빈도를 세는 것을 넘어, 단어와 문서의 '의미'를 이해하도록 돕는 원리와 실제 구현 방법을 학습합니다.

□ 예제:

학습 로드맵

1. **기초 다지기:** 단어의 빈도만 세는 Bag-of-Words(BoW) 방식의 한계를 이해합니다.
2. **핵심 개념 (TF-IDF):** BoW를 개선하여 '중요한 단어'에 가중치를 부여하는 TF-IDF의 원리를 배웁니다.
3. **심화 개념 (단어 임베딩):** 단어의 '의미' 자체를 벡터 공간에 표현하는 단어 임베딩의 개념으로 나아갑니다.
4. **주요 모델:** 대표적인 단어 임베딩 모델인 Word2Vec과 GloVe의 차이점을 파악합니다.
5. **실습:** Python의 Scikit-learn과 Gensim 라이브러리를 사용해 TF-IDF와 Word2Vec을 직접 구현해봅니다.

주요 용어 정리

용어	쉬운 설명	원어	비고
TF-IDF	특정 문서에서 중요하지만 전체에서는 흔치 않은 단어에 높은 점수를 주는 가중치	Term Frequency-Inverse Document Frequency	키워드 추출, 문서 분류에 사용
단어 임베딩	단어를 의미를 담은 촘촘한(dense) 숫자 벡터로 변환하는 기법	Word Embedding	단어 간 의미 관계 포착 가능
Bag-of-Words (BoW)	문서를 단어의 순서는 무시하고, 출현 빈도만 담은 가방(bag)으로 보는 표현 방식	Bag-of-Words	가장 단순한 텍스트 표현
One-Hot Encoding	단어 사전에 있는 단어 중 하나만 1이고 나머지는 0인 벡터로 단어를 표현하는 방식	One-Hot Encoding	희소 (sparse), 고차원, 의미 없음
코사인 유사도	두 벡터 사이의 각도를 이용해 얼마나 유사한지 측정하는 지표. (1에 가까울수록 유사)	Cosine Similarity	단어/문서 벡터의 유사도 측정
불용어	분석에 큰 의미가 없는 단어들 (예: a, the, is, in)	Stop Words	전처리 과정에서 보통 제거
어간 추출 (스테밍)	단어의 어미를 잘라 어간(기본 형태)을 추출하는 과정 (예: cats → cat)	Stemming	형태적으로 단순화

1 TF-IDF (Term Frequency-Inverse Document Frequency)

1.1 핵심 개념: 왜 TF-IDF가 필요한가?

단순히 단어의 빈도만 세는 Bag-of-Words(BoW) 방식은 큰 한계가 있습니다. 예를 들어, 보스턴 지역 뉴스를 분석할 때 '보스턴(Boston)'이라는 단어는 모든 기사에 자주 등장할 것입니다. BoW 방식에서는 이 단어가 매우 중요하다고 판단하겠지만, 실제로는 모든 문서에 나타나므로 문서를 구별하는 데 아무런 도움이 되지 않습니다.

TF-IDF는 이러한 문제를 해결하기 위해 등장했습니다. 핵심 아이디어는 다음과 같습니다.

> **한 문서 안에서 자주 등장하는 단어(TF, Term Frequency)일수록 중요하지만, 여러 문서에 걸쳐 공통적으로 자주 나타나는 단어(DF, Document Frequency)일수록 중요도는 낮아진다.**

즉, 특정 주제를 잘 나타내는 핵심 단어에 높은 가중치를 부여하는 방식입니다.

1.2 계산 원리

TF-IDF는 **Term Frequency (TF)**와 **Inverse Document Frequency (IDF)** 두 값의 곱으로 계산됩니다.

□ 핵심 요약

TF-IDF 계산 공식

- **TF (단어 빈도):** 특정 문서 내에서 단어가 얼마나 자주 등장하는가?

$$\mathrm{TF}(t,d) = \frac{\text{문서 } d\text{에서 단어 } t\text{의 등장 횟수}}{\text{문서 } d\text{의 전체 단어 수}}$$

- **IDF (역문서 빈도):** 특정 단어가 전체 문서 집합에서 얼마나 희귀한가?

$$\mathrm{IDF}(t) = \ln\left(\frac{\text{총 문서의 수}}{\text{단어 } t\text{를 포함하는 문서의 수}}\right)$$

  - 이 공식에서 분모가 0이 되는 것을 방지하고, 모든 단어가 최소한의 값을 갖도록 실제 구현에서는 분모와 분자에 1을 더하는 '스무딩(smoothing)' 기법이 자주 사용됩니다.

- **최종 TF-IDF:**

$$\text{TF-IDF}(t,d) = \mathrm{TF}(t,d) \times \mathrm{IDF}(t)$$

1.3 계산 예시

다음 4개의 문서가 있다고 가정해봅시다.

- **Doc 1:** "cat dog cat"
- **Doc 2:** "dog mouse dog"
- **Doc 3:** "dog mouse"
- **Doc 4:** "mouse cat dog"

December 10, 2025

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 06

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 06의 핵심 개념 학습

□ 핵심 요약

이번 강의에서는 단어를 구성하는 개별 문자를 벡터로 표현하는 **문자 임베딩**을 학습합니다.
이를 통해 OOV(Out-of-Vocabulary) 문제와 철자 오류에 강건한 모델을 만들 수 있습니다.
다음으로, 레이블 없는 데이터로부터 효율적인 데이터 표현을 학습하는 비지도 학습 모델인 **오토인코더**를 다룹니다.
오토인코더의 기본 구조, 스택 오토인코더, 희소 오토인코더, 그리고 변형 오토인코더의 개념과 활용법을 알아봅니다.

Contents

1 용어 정리

[title=핵심 용어 정리]

용어	쉬운 설명	원어 / 관련 개념
문자 임베딩	단어가 아닌 개별 문자(character)를 벡터로 변환하는 기술. 접두사, 접미사 등 단어 내부 구조를 학습할 수 있음.	Character Embeddings
오토인코더	데이터를 압축(인코딩)했다가 다시 원본으로 복원(디코딩)하도록 학습하는 신경망. 차원 축소, 특징 추출 등에 사용됨.	Autoencoder (AE)
인코더	오토인코더의 일부로, 입력 데이터를 저차원의 잠재 공간(latent space) 벡터로 압축하는 역할.	Encoder
디코더	오토인코더의 일부로, 압축된 잠재 벡터를 다시 원본 데이터 차원으로 복원하는 역할.	Decoder
재구성 손실	오토인코더의 원본 입력과 디코더가 복원한 출력 간의 차이. 이 손실을 최소화하는 방향으로 학습이 진행됨.	Reconstruction Loss
t-SNE	고차원 데이터를 시각화하기 좋은 2차원 또는 3차원으로 축소하는 알고리즘. 데이터 클러스터를 시각적으로 확인할 때 유용함.	t-Distributed Stochastic Neighbor Embedding
희소 오토인코더	오토인코더의 잠재 공간(코딩) 뉴런 중 일부만 활성화되도록 제약을 가하는 방식. 데이터의 특징을 더 명확하게 분리할 수 있음.	Sparse Autoencoder
KL 발산	두 확률 분포의 차이를 측정하는 지표. 희소 오토인코더에서 목표 활성도(sparsity)와 실제 활성도의 차이를 줄이는 손실 함수로 사용됨.	Kullback-Leibler Divergence
변형 오토인코더	잠재 공간을 정규분포 같은 연속적인 확률 분포로 만드는 생성 모델. 새로운 데이터를 생성하는 데 강점이 있음.	Variational Autoencoder (VAE)

2 문자 임베딩 (Character Embeddings)

2.1 개요

문자 임베딩은 단어(word) 단위가 아닌, 단어를 구성하는 개별 문자(character)를 연속적인 벡터 공간에 표현하는 방법입니다.

이 접근법은 모델이 단어의 철자(orthographic) 및 형태론적(morphological) 특징을 학습하도록 돕습니다. 예를 들어, 'running', 'runner', 'ran'과 같은 단어들이 'run'이라는 어근을 공유한다는 것을 파악할 수 있게 됩니다.

문자 임베딩의 주요 장점

- **Out-of-Vocabulary (OOV) 문제 완화**: 훈련 데이터에 없던 새로운 단어가 등장해도, 그 단어를 구성하는 문자들은 이미 학습했을 가능성이 높으므로 의미 있는 벡터 표현을 생성할 수 있습니다.
- **철자 오류(Misspellings)에 강건함**: 단어에 오타가 있어도, 대부분의 문자는 올바르기 때문에 어느 정도 유사한 벡터 표현을 유지할 수 있습니다.
- **형태론적 정보 포착**: 접두사(prefix), 접미사(suffix), 어근(stem)과 같은 단어 내부의 미세한 구조를 포착하여, 형태론적으로 풍부한 언어(예: 핀란드어, 터키어)에서 특히 효과적입니다.

2.2 적용 분야

- **형태론이 복잡한 언어 처리**: 핀란드어는 15개의 문법적 격(case)을 가지며, 터키어는 어근에 많은 접사가 붙어 단어가 길어집니다. 이런 언어에서는 단어 사전 크기가 매우 커지므로, 문자 임베딩을 통해 모델을 더 효율적으로 만들 수 있습니다.
- **개체명 인식 (Named Entity Recognition, NER)**: 사람 이름, 지명 등은 매우 다양하고 훈련 데이터에 등장하지 않은 경우가 많습니다. 문자 임베딩은 이런 새로운 개체명을 인식하는 데 도움을 줍니다.
- **맞춤법 교정**: 단어의 철자 오류를 탐지하고 교정하는 모델을 자연스럽게 구축할 수 있습니다.

2.3 구현 방법

문자 임베딩은 주로 RNN, CNN과 같은 신경망 모델이나 하이브리드 접근법을 통해 구현됩니다.

- **Character-Level RNNs/CNNs**: 문자의 시퀀스를 RNN이나 CNN에 입력하여 단어 벡터를 동적으로 생성합니다. RNN은 순차적 정보를, CNN은 지역적 패턴(예: 'ing' 접미사)을 포착하는 데 유리합니다.
- **Embedding Layer**: 각 문자를 고유한 정수 인덱스에 매핑한 후, 임베딩 레이어를 통과시켜 저차원의 밀집 벡터(dense vector)로 변환합니다.
- **Hybrid Approaches**: 문자 임베딩과 단어 임베딩을 결합하여 사용하는 방식입니다. 두 임베딩 벡터를 연결(concatenate)하여 모델의 입력으로 사용하면, 단어의 미세 구조와 전체적인 의미를 모두 활용할 수 있습니다.

2.4 Python 실습 예제

TensorFlow/Keras를 사용하여 문자 임베딩을 적용한 간단한 텍스트 분류 모델을 구현하는 예제입니다.

문자 임베딩을 위한 데이터 전처리

```
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# 샘플데이터
texts = ["hello world", "machine learning", "deep learning"]
labels = np.array([1, 0, 0])

# 1. 문자수준토큰화 (Character-level Tokenization)
# 의Tokenizer char_level=True 옵션사용
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(texts)

# 텍스트를정수시퀀스로변환
sequences = tokenizer.texts_to_sequences(texts)
print정수(" 시퀀스:", sequences)

# 문자사전 (Vocabulary)
char_index = tokenizer.word_index
print문자(" 인덱스:", char_index)

# 2. 패딩 (Padding)
# 모든시퀀스의길이를통일하기위해가장긴시퀀스에맞춰을 0 채움
max_sequence_length = max(len(seq) for seq in sequences)
padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length,
    padding='post')
print패딩된(" 시퀀스:\n", padded_sequences)
```

Listing 1: Character-level tokenization and padding

사전 크기 및 패딩

`tokenizer.word`$_{i}ndex$. (

문자 임베딩을 사용한 LSTM 모델 구축

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

# 사전크기은 (+1 패딩용인덱스 0)
vocab_size = len(tokenizer.word_index) + 1
embedding_dim = 8 # 임베딩벡터의차원

# 모델구축
model = Sequential([
    # Embedding Layer: 정수인덱스를 embedding_dim 차원의벡터로변환
```

```
    Embedding(input_dim=vocab_size,
              output_dim=embedding_dim,
              input_length=max_sequence_length),
    LSTM(64),
    Dense(1, activation='sigmoid') # 이진분류
])

# 모델컴파일
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()

# 모델훈련
model.fit(padded_sequences, labels, epochs=10, batch_size=2)
```

Listing 2: Building and training a model with character embeddings

이 모델에서 임베딩 레이어는 훈련 과정 중에 데이터에 맞게 문자 벡터를 학습합니다. 문자 수가 단어 수보다 훨씬 적기 때문에, 임베딩 차원(`embedding`$_dim$) (: $8, 16$).

3 오토인코더 (Autoencoders)

3.1 기본 개념

오토인코더는 레이블이 없는 데이터로부터 효율적인 데이터 표현(인코딩)을 학습하기 위한 비지도 학습(unsupervised learning) 방식의 인공 신경망입니다.

오토인코더는 두 부분으로 구성됩니다:

- **인코더 (Encoder)**: 입력 데이터를 저차원의 잠재 공간(latent space)으로 압축합니다. 이 압축된 표현을 **인코딩(encoding)** 또는 **코딩(coding)**이라고 합니다.
- **디코더 (Decoder)**: 인코딩된 표현을 다시 원래의 입력 데이터와 같은 차원으로 복원합니다.

학습 목표는 입력 데이터와 디코더가 복원한 출력 데이터 간의 차이, 즉 **재구성 손실(reconstruction loss)**을 최소화하는 것입니다.

Figure: 오토인코더의 기본 구조. 입력(x)이 인코더를 통해 저차원 표현(z)으로 압축된 후, 디코더를 통해 다시 원본과 유사한 출력(x')으로 복원됩니다.

오토인코더의 직관적 예시

두 가지 숫자 시퀀스를 기억해야 한다고 가정해봅시다.

- **시퀀스 1**: 5, 7, 3, 5, 8, 9, 67, 5, ... (규칙 없음)
- **시퀀스 2**: 70, 68, 66, 64, 62, ... (규칙: 70에서 시작해서 2씩 감소)

시퀀스 1은 모든 숫자를 개별적으로 외워야 하지만, 시퀀스 2는 "시작 값 70, 규칙 -2"라는 훨씬 적은 정보로 압축하여 기억할 수 있습니다. 오토인코더는 이처럼 데이터 내의 패턴이나 구조를 학습하여 데이터를 효율적으로 압축하고 표현하는 방법을 배웁니다.

3.2 스택 오토인코더 (Stacked Autoencoders)

스택 오토인코더는 여러 개의 은닉층을 쌓아 만든 깊은(deep) 오토인코더입니다. 일반적으로 인코더와 디코더가 대칭적인 구조를 가집니다.

예를 들어, 28x28 픽셀 (총 784개) 크기의 이미지를 압축하는 경우 다음과 같은 구조를 가질 수 있습니다.

$$784 \to 100 \to 30(\text{인코딩}) \to 100 \to 784$$

여기서 30차원 벡터가 이미지의 핵심 특징을 담은 인코딩이 됩니다. 이처럼 내부 표현의 차원 수가 입력보다 낮은 오토인코더를 **불완전(undercomplete) 오토인코더**라고 하며, 이는 모델이 데이터의 가장 중요한 특징을 학습하도록 강제합니다.

Figure: 스택 오토인코더 예시. *784*차원 입력을 *30*차원의 코딩으로 압축한 후 다시 *784*차원으로 복원합니다.

3.3 활용: 비지도 사전학습 (Unsupervised Pretraining)

오토인코더의 가장 강력한 활용 사례 중 하나는 레이블이 부족한 대규모 데이터셋에서 특징 추출기(feature extractor)를 학습하는 것입니다.

비지도 사전학습 절차

Phase 1: 오토인코더 훈련 레이블 유무에 상관없이 전체 데이터를 사용하여 스택 오토인코더를 훈련시킵니다. 이 과정을 통해 인코더는 데이터를 잘 표현하는 방법을 학습하게 됩니다.

Phase 2: 분류기 훈련 훈련된 오토인코더에서 디코더 부분은 버리고 인코더 부분만 가져옵니다.
이 인코더 위에 새로운 분류기(예: Softmax 층)를 추가합니다.
이제 레이블이 있는 소량의 데이터만을 사용하여 이 새로운 모델(인코더 + 분류기)을 훈련시킵니다. 이때 인코더의 가중치는 고정(freeze)하거나 미세 조정(fine-tuning)할 수 있습니다.

이 방식은 적은 양의 레이블 데이터로도 높은 성능의 분류기를 만들 수 있게 해줍니다. 인코더가 대규모 비지도 데이터로부터 이미 데이터의 유용한 구조를 학습했기 때문입니다.

Figure: 오토인코더를 이용한 비지도 사전학습. *1*단계에서 전체 데이터로 오토인코더를 학습하고, *2*단계에서 학습된 인코더를 재사용하여 레이블된 데이터로 분류기를 학습합니다.

3.4 잠재 공간 시각화: t-SNE

오토인코더가 학습한 잠재 공간(latent space, 즉 인코딩들의 공간)이 데이터를 얼마나 잘 군집화했는지 확인하기 위해 t-SNE와 같은 시각화 도구를 사용합니다.

t-SNE는 고차원의 인코딩 벡터(예: 30차원)를 2차원 평면에 매핑하여, 비슷한 데이터들이 가깝게 모이고 다른 데이터들은 멀리 떨어지도록 배치합니다. 이를 통해 각 클러스터가 어떤 종류의 데이터에 해당하는지(예: 신발, 가방, 셔츠) 시각적으로 확인할 수 있습니다.

t-SNE를 이용한 인코딩 시각화 코드 예시

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# 1. 훈련된인코더를사용하여검증데이터의인코딩압축된      ( 표현)을 얻음
X_valid_compressed = stacked_encoder.predict(X_valid)

# 2. t-SNE 모델을사용하여차원  30 인코딩을차원으로 2 변환
tsne = TSNE()
X_valid_2D = tsne.fit_transform(X_valid_compressed)

# 3. 차원2 평면에산점도로시각화색상은    ( 실제레이블 )
plt.scatter(X_valid_2D[:, 0], X_valid_2D[:, 1], c=y_valid, s=10, cmap="tab10")
```

```
plt.show()
```

Listing 3: Visualizing codings with t-SNE

4 희소 오토인코더 (Sparse Autoencoders)

4.1 개념과 필요성

기본적인 불완전 오토인코더는 병목(bottleneck) 층의 뉴런 수를 줄여 데이터 압축을 유도합니다. 하지만 만약 데이터가 매우 다양하다면(예: 숫자, 동물, 자동차 이미지가 섞여 있는 경우), 작은 병목 층만으로는 모든 종류의 특징을 학습하기 어려울 수 있습니다.

이때 병목 층의 뉴런 수를 입력보다 크게 설정하는 **과완전(overcomplete) 오토인코더**를 사용할 수 있지만, 이 경우 모델이 단순히 입력을 복사하는 항등 함수(identity function)를 학습할 위험이 있습니다.

희소 오토인코더는 과완전 오토인코더를 사용하면서도, 인코딩 층의 뉴런 대부분이 비활성화(출력이 0에 가까움)되도록 강제하는 규제(regularization)를 추가한 모델입니다. 즉, 각 입력에 대해 소수의 뉴런만 활성화하여 해당 입력의 특정 특징을 전문적으로 감지하도록 유도합니다.

희소 오토인코더의 작동 원리

이미지 데이터셋에 숫자 '7'과 '8'이 있다고 가정합시다. 희소 오토인코더는 다음과 같이 학습할 수 있습니다.

- 어떤 뉴런은 '7'의 위쪽 가로선을 감지할 때만 강하게 활성화됩니다.
- 다른 뉴런은 '8'의 아래쪽 원형 부분을 감지할 때만 강하게 활성화됩니다.
- '7' 이미지가 입력되면 '가로선' 뉴런은 활성화되지만, '아래쪽 원' 뉴런은 비활성화됩니다.

이처럼 각 뉴런이 데이터의 특정 부분 특징을 전담하게 하여, 더 유용하고 해석 가능한 특징을 학습할 수 있습니다.

4.2 구현 방법

희소성을 강제하기 위해 손실 함수에 규제 항을 추가합니다. 대표적인 두 가지 방법이 있습니다.

4.2.1 L1 활동 규제 (L1 Activity Regularization)

인코딩 층의 활성화 값(activation)에 대해 L1 손실을 추가합니다. L1 규제는 가중치를 0으로 만드는 경향이 있으므로, 많은 뉴런의 활성화 값을 0에 가깝게 만들어 희소성을 유도합니다.

$$\text{Total Loss} = \text{Reconstruction Loss} + \lambda \sum_i |a_i|$$

여기서 a_i는 인코딩 층의 i번째 뉴런의 활성화 값이고, λ는 규제 강도를 조절하는 하이퍼파라미터입니다.

Keras에서 L1 활동 규제 적용하기

```
from tensorflow.keras.layers import ActivityRegularization
```

```
2
3 # ... 인코더모델정의    ...
4 # 인코딩층 (Dense) 바로뒤에  ActivityRegularization 레이어를추가
5 keras.layers.Dense(300, activation="sigmoid"),
6 keras.layers.ActivityRegularization(l1=1e-3)
7 # ...
```

Listing 4: Applying L1 Activity Regularization

4.2.2 쿨백-라이블러 발산 (Kullback-Leibler Divergence)

더 정교한 방법으로, 각 뉴런의 평균 활성도를 특정 목표값(target sparsity, 예: 0.1)에 가깝게 유지하도록 강제합니다. 이는 KL 발산을 사용하여 구현됩니다.

KL 발산은 두 확률 분포의 차이를 측정하며, 여기서는 "뉴런이 활성화될 실제 확률(평균 활성도)"과 "목표 확률(target sparsity)" 사이의 차이를 측정합니다.

$$D_{KL}(p||\hat{p}) = p \log \frac{p}{\hat{p}} + (1-p) \log \frac{1-p}{1-\hat{p}}$$

여기서 p는 목표 희소성(예: 0.1), $\hat{p}$는 배치(batch) 데이터에 대한 뉴런의 실제 평균 활성도입니다. 이 D_{KL} 값이 손실 함수에 추가되어, $\hat{p}$가 p에 가까워지도록 모델을 훈련시킵니다.

Figure: KL 발산 손실 함수. 실제 활성도(Actual sparsity)가 목표 활성도(Target sparsity)와 일치할 때 비용(Cost)이 0이 되고, 멀어질수록 급격히 증가합니다.

주의사항

KL 발산은 L1 규제보다 더 강하게 목표 희소성을 강제하는 경향이 있습니다. 손실 함수의 모양이 목표 지점에서 더 뾰족하기(steeper) 때문에, 수렴 속도가 더 빠를 수 있습니다.

5 변형 오토인코더 (Variational Autoencoders, VAEs)

5.1 개념과 목적

일반 오토인코더가 학습한 잠재 공간은 불연속적일 수 있습니다. 즉, 두 개의 유효한 인코딩 사이의 중간 지점에 있는 벡터가 의미 없는 출력으로 디코딩될 수 있습니다.

변형 오토인코더(VAE)는 잠재 공간이 잘 구조화되고 연속적이도록 만드는 **생성 모델(generative model)**입니다. VAE는 입력을 하나의 고정된 벡터로 인코딩하는 대신, 평균(μ)과 표준편차(σ)를 갖는 확률 분포(일반적으로 가우시안 분포)로 인코딩합니다.

디코더는 이 분포에서 랜덤하게 샘플링한 벡터를 사용하여 입력을 재구성합니다. 이러한 확률적 접근 방식은 다음과 같은 장점을 가집니다.

- **연속적인 잠재 공간**: 비슷한 입력들은 잠재 공간에서 서로 겹치는 분포로 표현됩니다. 따라서 한 분포의 지점에서 다른 분포의 지점으로 점진적으로 이동하면, 출력도 자연스럽게 변환됩니다 (예: 숫자 '6' 이미지가 '0' 이미지로 부드럽게 변함).
- **새로운 데이터 생성**: 잠재 공간에서 임의의 점을 샘플링하여 디코더에 통과시키면, 훈련 데이터와 유사하지만 완전히 새로운 데이터를 생성할 수 있습니다.

Figure: VAE 아키텍처. 인코더는 평균(μ)과 로그 분산($\log \sigma$)을 출력하고, 가우시안 노이즈를 더해 샘플링된 코딩을 생성합니다. 이 코딩이 디코더의 입력이 됩니다.

5.2 손실 함수

VAE의 손실 함수는 두 가지 요소로 구성됩니다.

$$\text{Total Loss} = \text{Reconstruction Loss} + \text{Latent Loss}$$

1. 재구성 손실 (Reconstruction Loss) 일반 오토인코더와 동일하게, 원본 입력과 디코더의 출력 간의 차이를 측정합니다.

2. 잠재 손실 (Latent Loss) 인코더가 생성한 분포가 목표 분포(일반적으로 표준 정규 분포, 즉 평균=0, 분산=1)에 가깝도록 강제하는 규제 항입니다. 이 손실은 KL 발산을 사용하여 계산됩니다.

$$\mathcal{L}_{\text{latent}} = D_{KL}(\mathcal{N}(\mu, \sigma^2) || \mathcal{N}(0,1)) = -\frac{1}{2}\sum_{i=1}^{n}(1 + \log \sigma_i^2 - \sigma_i^2 - \mu_i^2)$$

이 손실 항은 모델이 잠재 공간에 분포들을 너무 멀리 흩어놓지 않고, 원점 근처에 잘 군집화하도록 유도합니다. 만약 이 항이 없다면, 인코더는 재구성 손실을 줄이기 위해 각 분포의 분산(σ)을 0으로 만들어 사실상 일반 오토인코더처럼 작동하려 할 것입니다.

VAE의 핵심 트레이드오프

VAE는 재구성(입력을 잘 복원하는 것)과 규제(잠재 공간을 잘 구조화하는 것) 사이의 균형을 맞추는 과정입니다. 잠재 손실이 너무 강하면 이미지가 흐릿하게 재구성될 수 있고, 재구성 손실만 강조하면 잠재 공간의 연속성이 떨어질 수 있습

니다.

6 빠르게 훑어보기 (1-Page Summary)

문자 임베딩 (Character Embeddings)

정의: 단어 대신 개별 문자를 벡터로 표현.
장점: OOV 문제, 철자 오류에 강건. 형태론적 정보 포착.
적용: 형태론이 복잡한 언어(터키어 등), 개체명 인식(NER).
구현: Tokenizer(char$_l evel = True$), $EmbeddingLayer + RNN/CNN$.

오토인코더 (Autoencoders)

정의: 비지도 학습으로 데이터의 효율적 표현을 학습하는 신경망 (인코더 + 디코더).
목표: 재구성 손실(입력-출력 차이) 최소화.
종류:

- **스택 AE**: 여러 층을 쌓아 깊게 만듦 ($784 \to 100 \to 30 \to 100 \to 784$).
- **희소 AE**: 인코딩 층 뉴런의 일부만 활성화되도록 규제 (L1, KL 발산).
- **변형 AE (VAE)**: 생성 모델. 잠재 공간을 확률 분포로 만들어 연속성을 확보.

핵심 활용: 비지도 사전학습 (레이블 없는 데이터로 인코더 학습 후, 소량의 레이블 데이터로 분류기 학습).

주요 규제 기법

- **L1 활동 규제**:
  - **목표**: 인코딩 층의 활성화 값을 희소하게(sparse) 만듦.
  - **방법**: 손실 함수에 활성화 값의 절댓값 합(L1 norm)을 추가.
- **KL 발산 규제**:
  - **목표 (희소 AE)**: 뉴런의 평균 활성도를 특정 목표값(예: 10%)으로 유도.
  - **목표 (VAE)**: 잠재 분포를 표준 정규 분포에 가깝게 만듦.
  - **방법**: 두 확률 분포(실제 vs 목표)의 차이를 측정하여 손실에 추가.

December 10, 2025

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 07

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 07의 핵심 개념 학습

_summarybox 이 문서는 자연어 처리(NLP)의 두 가지 주요 주제를 다룹니다.

첫째, 이전 퀴즈 6의 복습으로, **Autoencoder(오토인코더)**의 기본 원리와 **'Undercomplete(불완전)'** 표현의 중요성을 학습합니다. 이는 정보를 압축하여 핵심 특징을 추출하는 방법을 다룹니다.

둘째, 이 강의의 핵심 주제인 **토픽 모델링(Topic Modeling)**을 배웁니다. 이는 대량의 문서에서 '숨겨진 주제'를 발견하는 비지도 학습 기법입니다. 주요 알고리즘으로 확률 기반의 **LDA(Latent Dirichlet Allocation)**와 행렬 분해 기반의 **NMF(Non-Negative Matrix Factorization)**를 비교 분석하고, Python/R 실습 코드를 통해 구현 방법을 살펴봅니다. 마지막으로, 생성된 토픽의 품질을 평가하고 최적의 토픽 개수(K)를 찾는 지표(**응집도, 배타성**)를 학습합니다. summarybox

Contents

1 지난 시간 복습: Quiz 6 및 Autoencoder

본격적인 토픽 모델링 학습에 앞서, 데이터 표현(Representation)과 관련된 이전 퀴즈 6의 주요 개념들을 복습합니다. 이는 정보를 효율적으로 압축하는 Autoencoder(오토인코더)에 대한 이해를 돕습니다.

1.1 One-Hot Encoding의 단점

One-Hot Encoding(원-핫 인코딩)은 범주형 데이터를 벡터로 표현하는 기법이지만, 다음과 같은 명확한 단점이 있습니다.

- **희소성 (Sparsity):** 벡터의 대부분이 0으로 채워집니다. 예를 들어 10,000개의 단어 사전을 원-핫 인코딩하면, 각 단어 벡터는 9,999개의 0과 1개의 1을 갖게 됩니다.
- **높은 차원 (High Dimensionality):** 단어의 개수만큼 벡터의 차원이 증가하여 계산 비효율성을 초래합니다. 0과의 곱셈 연산이 많아져 리소스를 낭비하게 됩니다.
- **정보 비효율성 (Inefficient Representation):** 정보 자체를 효율적으로 나타내지 못합니다. (예: '고양이'와 '강아지' 벡터 간의 관계성을 표현하지 못함)

이러한 문제를 해결하기 위해 **Embedding(임베딩)**과 같은 저차원의 밀집된(dense) 벡터 표현 방식이 선호됩니다.

1.2 Autoencoder(오토인코더)와 Bottleneck

Autoencoder(오토인코더)는 입력 데이터를 더 낮은 차원의 벡터로 '압축'했다가(Encoding), 다시 원래 차원으로 '복원'(Decoding)하도록 학습되는 신경망입니다.

- **목표:** 입력과 출력이 최대한 같아지도록(예: $Input \approx Output$) 학습합니다.
- **핵심: Bottleneck(병목) 레이어**
  - 인코더(Encoder)와 디코더(Decoder) 사이에 위치하는 가장 차원이 낮은 은닉층입니다.
  - 입력 데이터의 핵심적인 특징(representation)이 이 '병목' 지점에 압축되어 저장됩니다.
  - 이 압축된 벡터(Encoding Vector)가 데이터의 저차원 표현이 됩니다.

1.3 Undercomplete Autoencoder

Autoencoder는 '병목' 레이어의 차원에 따라 'Undercomplete' 또는 'Overcomplete'로 불릴 수 있습니다.

QA: 왜 'Undercomplete(불완전)'라고 부르나요? **Q: 'Undercomplete'라는 용어의 의미가 무엇인가요? 데이터를 압축하는 것은 알겠습니다.**

A: 'Undercomplete'는 '완전하지 않다'는 의미로, 입력 데이터의 차원보다 **더 낮은 차원**의 표현(representation)을 사용한다는 뜻입니다.

예를 들어, 3차원 공간(x_1, x_2, x_3)에 분포하는 데이터가 있다고 가정해 봅시다. 이 데이터를 '완전하게(complete)' 표현하려면 3차원 공간이 그대로 필요합니다.

하지만 만약 우리가 이 데이터를 2차원 평면에 강제로 '압축'(투영)하여 표현하려 한다면, 이는 원본 데이터를 표현하기에 차원이 '부족'합니다. 이를 **Undercomplete Representation**이라고 부릅니다.

- **Undercomplete:** 입력 차원 (예: 100) > 병목 차원 (예: 10)
- **Overcomplete:** 입력 차원 (예: 100) < 병목 차원 (예: 200)

Autoencoder의 주된 목적 중 하나는 이 Undercomplete 표현, 즉 저차원의 핵심 특징 벡터를 학습하는 것입니다.

1.4 Stacked Autoencoder (적층 오토인코더)

Stacked Autoencoder(적층 오토인코더)는 여러 개의 은닉층을 '쌓아서(stack)' 만든 깊은(deep) 신경망 구조의 오토인코더를 의미합니다.

- **Shallow Network (얕은 신경망):** 은닉층이 1개인 경우.
- **Deep Network (깊은 신경망):** 은닉층이 2개 이상인 경우. 적층 오토인코더는 정의상 Deep Network입니다.

과거에는 깊은 신경망을 한 번에 학습시키기 어려웠습니다. (예: Gradient Vanishing/Exploding 문제) 그래서 'Stacked'라는 이름처럼, 한 번에 한 층씩(Shallow Autoencoder) 학습시키고 그 가중치를 고정(freeze)한 뒤, 다음 층을 샌드위치처럼 쌓아 올리는 방식(Layer-wise training)을 사용했습니다.

참고: 현대의 Deep Network 학습 현재는 다음과 같은 기술들 덕분에 깊은 신경망도 한 번에(end-to-end) 효과적으로 학습할 수 있게 되었습니다.

- **Adam Optimizer:** Gradient의 스케일을 조절하여 효율적으로 최적점을 찾아갑니다.
- **Batch Normalization (배치 정규화):** 각 층을 통과하는 신호(signal)를 국소적으로 정규화하여 학습을 안정시킵니다.
- **Shortcut Connections (예: ResNet):** 층을 건너뛰는 연결을 만들어 신호(Gradient)가 잘 전파되도록 돕습니다.

1.5 시퀀스(Sequence)를 위한 Autoencoder

Autoencoder는 이미지뿐만 아니라 시퀀스 데이터(예: 문장)에도 적용할 수 있습니다. 주로 **LSTM(Long Short-Term Memory)**과 같은 RNN(Recurrent Neural Network)을 사용합니다.

- **Encoder:** 시퀀스(문장)를 입력받아 전체 문맥을 압축한 **단일 벡터**(Context Vector)를 생성합니다. (Bottleneck 역할)
- **Decoder:** 이 단일 벡터를 입력받아 원래의 시퀀스(문장)를 다시 생성(복원)하도록 학습됩니다.

초심자의 오해 vs. 올바른 이해: Autoencoder의 진짜 목적 **Q: 데이터를 압축했다가 다시 복원할 거면 왜 굳이 압축을 하나요? 완벽하게 복원(Reconstruction)하려면 그냥 원본을 쓰면 되지 않나요?**

A: 아주 훌륭한 질문입니다. Autoencoder의 목적은 '완벽한 복원' 그 자체가 아닙니다.

- **잘못된 오해:** Autoencoder는 데이터를 손실 없이 압축했다가 복원하는 '파일 압축(zip)' 프로그램 같은 것이다.
- **올바른 이해:** Autoencoder의 진짜 목적은 복원 과정(Decoder)을 '미끼'로 사용하여, 입력 데이터의 핵심 특징을 압축한 **'저차원 표현(Encoding)'**을 얻는 것입니다.

우리는 이렇게 얻어낸 '압축된 벡터'(Encoder의 출력)를 **분류(Classification)**나 **군집화(Clustering)** 같은 다른 작업(Downstream Task)의 입력값으로 사용합니다.

예를 들어, 10,000차원의 희소한 텍스트 벡터를 100차원의 밀집된 벡터로 압축(Encoding)한 뒤, 이 100차원 벡터를 사용하여 문서 분류 모델을 학습시키는 것이 훨씬 효율적입니다. 몇 차원으로 '짜는(squeeze)' 것이 가장 좋은지는 **실험적(experimental)으로 결정**해야 합니다. 즉, 최종 작업(예: 분류)의 성능이

가장 좋게 나오는 차원을 선택합니다.

2 토픽 모델링 (Topic Modeling) 소개

토픽 모델링(Topic Modeling)은 대규모 텍스트 모음(Corpus)에서 자동으로 '숨겨진 주제(Latent Topics)'를 발견하는 **비지도 학습(Unsupervised Learning)** 알고리즘입니다.

여기서 '비지도 학습'이란, 데이터에 '정답(Label)'이 주어지지 않아도(예: '이 문서는 스포츠 기사다'라는 정답이 없음) 기계가 스스로 데이터 내부의 구조나 패턴을 찾아내는 것을 의미합니다.

2.1 토픽 모델링의 필요성: 클러스터링과의 차이

"문서에서 주제를 찾는다"는 개념은 '클러스터링(Clustering)'과 혼동하기 쉽습니다. 하지만 텍스트 데이터의 특성상 클러스터링만으로는 한계가 있습니다.

- **클러스터링 (예: K-Means):** 문서를 **단 하나의** 주제(클러스터)로 분류합니다. (Hard Assignment)
- **현실의 문서:** 하지만 현실의 문서는 여러 주제를 동시에 다룹니다. 예를 들어, '애플의 신형 아이폰 출시' 기사는 'IT(기술)', '경제(주가)', '디자인' 등 여러 주제를 포함할 수 있습니다.
- **토픽 모델링 (예: LDA):** 이 한계를 극복합니다. 문서를 단 하나의 주제로 분류하는 대신, **여러 토픽의 '혼합(Mixture)'**으로 간주합니다. (Soft Assignment)

비유: 과일 바구니 vs. 스무디

- **클러스터링 (Clustering):** 문서를 '사과 상자', '바나나 상자' 등 명확히 구분된 상자에 **하나씩** 집어넣는 것과 같습니다. (문서 1 → 상자 A)
- **토픽 모델링 (Topic Modeling):** 문서를 '사과 60

아래는 두 접근 방식의 차이점을 요약한 표입니다.

Table 1: 클러스터링과 토픽 모델링의 비교

특징	클러스터링 (예: K-Means)	토픽 모델링 (예: LDA)
문서 소속	문서는 **하나의** 클러스터에만 속함.	문서는 **여러** 토픽의 확률적 조합으로 표현됨.
결과 예시	"문서 A는 '스포츠' 그룹에 속한다."	"문서 A는 '스포츠' 60%, '경제' 30%, 'IT' 10%로 구성된다."
접근 방식	Hard Assignment (엄격한 할당)	Soft Assignment (유연한 할당)
주요 목적	문서를 상호 배타적인 그룹으로 분류.	문서 집합 내에 숨겨진 주제(Latent Topics)들의 분포를 발견.

토픽 모델링의 목표는 이 '숨겨진(Latent)' 주제들과, 각 문서가 이 주제들을 얼마나 포함하고 있는지(비율)를 알아내는 것입니다.

잠깐! '숨겨진(Latent)'이란 무슨 뜻인가요?

'Latent'는 '잠재적인', '숨겨진'을 의미합니다. 토픽 모델링에서 모델은 "이 단어들은 '주제 1'에 속한다"고 알려주지만, 그 '주제 1'이 인간의 언어로 '경제'인지 '정치'인지는 알려주지 않습니다. 모델이 찾아낸 주제(단어들의 집합)를 보고 **사람이 직접** "아, 이 토픽은 '경제'에 관한 것이구나"라고 라벨을 붙여야 합니다.

주요 토픽 모델링 방법론으로는 **LDA(Latent Dirichlet Allocation)**와 **NMF(Non-Negative Matrix Factorization)**가 있습니다.

3 Latent Dirichlet Allocation (LDA)

LDA(Latent Dirichlet Allocation, 잠재 디리클레 할당)는 가장 대표적인 토픽 모델링 알고리즘입니다. LDA는 **생성 확률 모델(Generative Probabilistic Model)**입니다.

"생성 모델"이란, '이 문서들은 어떻게 생성되었을까?'라는 과정을 역으로 추적하는 모델이라는 뜻입니다. LDA는 문서가 다음과 같은 두 가지 핵심 가정 하에 작성되었다고 봅니다.

1. **문서는 토픽의 혼합이다.** (A document is a mixture of topics.)
2. **토픽은 단어의 혼합이다.** (A topic is a mixture of words.)

3.1 LDA의 문서 생성 스토리 (직관적 비유)

LDA는 마치 로봇이 다음과 같은 확률적인 과정을 거쳐 문서를 '작성'한다고 가정합니다. 이 과정을 이해하는 것이 LDA의 핵심입니다.

LDA의 가상 문서 작성 과정 어떤 사람이 M 개의 문서로 이루어진 코퍼스(Corpus)를 작성하려 하고, 이 코퍼스에는 총 K 개의 토픽(예: 경제, 정치, 스포츠)이 있다고 가정합니다.

1. (문서 길이 결정) 첫 번째 문서($m = 1$)의 길이를 몇 단어(N)로 할지 정합니다. (예: 포아송 분포 $N \sim Poisson(\xi)$에서 숫자 100을 뽑음 $\rightarrow$ 100단어짜리 문서)

2. (문서의 토픽 분포 결정) 이 100단어짜리 문서의 '주제 배합 비율'(θ_m)을 정합니다. (예: 디리클레 분포 $\theta_m \sim Dirichlet(\alpha)$에서 [경제 60%, 정치 30%, 스포츠 10%]라는 비율을 뽑음) * 이 θ_m은 이 문서가 끝날 때까지 고정됩니다.

3. (개별 단어 생성) 이제 100개의 단어를 하나씩 채워 넣습니다. ($n = 1$ 부터 $N = 100$ 까지 반복)

- **3a. (이 단어의 토픽 선택)**: 방금 정한 문서의 토픽 분포([경제 60%, 정치 30%, ...])에 따라, **이번 단어 하나**에 할당할 토픽(z_n)을 뽑습니다. (예: 다항 분포 $z_n \sim Multinomial(\theta_m)$에서 '경제'가 뽑힘)
- **3b. (토픽에서 단어 선택)**: '경제' 토픽에 미리 정해진 단어 분포(예: β_k =[주식 40%, 금리 30%, ...]) 에 따라, 실제 단어(w_n)를 뽑습니다. (예: 다항 분포 $w_n \sim Multinomial(\beta_{z_n})$에서 '주식'이 뽑힘)

4. (반복) 두 번째 단어를 생성하기 위해 3a, 3b 과정을 반복합니다. (이번에는 '정치'가 뽑히고, '정치' 토픽에서 '선거'라는 단어가 뽑힐 수 있습니다.)

5. (모든 문서에 대해 반복) M 개의 모든 문서에 대해 1 4 과정을 반복합니다.

중요한 점: 이 모델은 'Bag-of-Words(단어 주머니)' 가정을 따르므로, 단어의 순서(*on* 다음에 *maximization* 이 나왔는지)는 전혀 고려하지 않습니다.

3.2 LDA의 수학적 프레임워크

위의 생성 과정을 수학적 기호로 표현하면 다음과 같습니다.

- M: 총 문서의 수
- K: 총 토픽의 수 (사용자가 지정하는 하이퍼파라미터)
- N: m 번째 문서의 단어 수 ($N_m \sim Poisson(\xi)$)
- α: 디리클레 분포의 하이퍼파라미터 (문서-토픽 분포 θ에 영향)
- β: 디리클레 분포의 하이퍼파라미터 (토픽-단어 분포 ϕ에 영향)

- θ_m: m 번째 문서의 토픽 분포 (K차원 벡터, $\theta_m \sim Dirichlet(\alpha)$)
- ϕ_k: k 번째 토픽의 단어 분포 (V차원 벡터, V=어휘 수, $\phi_k \sim Dirichlet(\beta)$)
- $z_{m,n}$: m 번째 문서의 n 번째 단어에 할당된 토픽 ($z_{m,n} \sim Multinomial(\theta_m)$)
- $w_{m,n}$: m 번째 문서의 n 번째 단어 (관찰된 값, $w_{m,n} \sim Multinomial(\phi_{z_{m,n}})$)

우리가 실제로 관찰하는 것은 w (단어들) 뿐입니다. LDA의 목표는 관찰된 w를 바탕으로, 숨겨진 변수인 θ (문서별 토픽 분포)와 ϕ (토픽별 단어 분포)를 추정하는 것입니다.

3.3 파라미터 추정: 최대 가능도 추정 (MLE)

모델은 우리가 가진 실제 문서들(관찰된 w)이 방금 설명한 'LDA 문서 생성 스토리'에서 나왔을 **확률(Likelihood)**을 계산합니다.

그리고 이 확률이 **최대**가 되도록 하는 파라미터(α, β, 그리고 그로부터 유도되는 θ, ϕ)를 찾는 과정을 거칩니다. 이를 **최대 가능도 추정(Maximum Likelihood Estimation, MLE)**이라고 합니다.

개념 이해: 최대 가능도 추정 (MLE) 비유 여러분의 손에 특정 확률분포(예: 정규분포)를 따르는 데이터(관찰값)가 주어졌다고 가정해 봅시다. 하지만 이 분포의 파라미터(예: 평균 μ, 분산 σ^2)는 모릅니다.

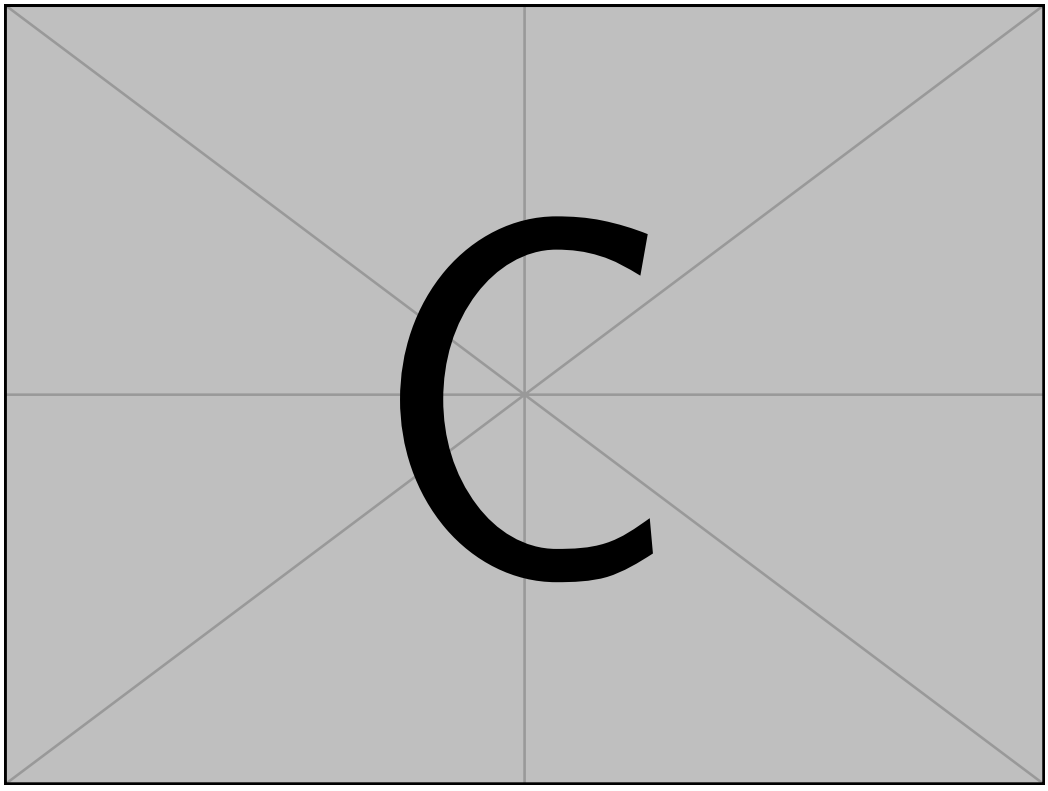

Figure 1: 관찰된 데이터(히스토그램)와 다른 파라미터(*A, B, C*)를 가진 모델

- **가정 A (모델 A):** "이 데이터는 평균이 아주 낮은 곳(A)에 있는 정규분포에서 나왔을 것이다." → 판단: "내가 가진 데이터(가운데 몰려있음)가 A에서 나왔을 확률(Likelihood)은 매우 낮다."
- **가정 B (모델 B):** "이 데이터는 평균이 약간 오른쪽(B)에 있는 정규분포에서 나왔을 것이다." → 판단: "A보다는 확률이 높지만, 여전히 낮다."
- **가정 C (모델 C):** "이 데이터는 평균이 데이터의 중심(C)에 있는 정규분포에서 나왔을 것이다." → 판단: "내가 가진 데이터가 C에서 나왔을 확률이 **가장 높다(Maximum Likelihood)**."

MLE는 이처럼 '관찰된 데이터를 가장 잘 설명하는' 모델의 파라미터(이 경우 C)를 찾는 방법입니다.

LDA도 마찬가지로, 우리가 가진 수많은 문서를 생성했을 확률이 가장 높은 토픽 분포(θ, ϕ)를 역으로 추정해냅니다. (실제로는 z와 같은 잠재 변수가 많아 **EM(Expectation-Maximization)** 알고리즘이나 깁스 샘플링 등을 사용합니다.)

3.4 LDA의 활용

LDA를 통해 추정된 문서별 토픽 분포(θ)와 토픽별 단어 분포(ϕ)는 다양하게 활용됩니다.

- **문서 분류 (Document Classification):** 문서의 토픽 분포(예: [경제 60%, ...]) 자체를 문서의 새로운 특징(Feature)으로 사용하여 분류 모델을 학습시킵니다.
- **추천 시스템 (Recommendation Systems):** 유사한 토픽 분포를 가진 문서(예: 기사, 상품)를 사용자에게 추천합니다.
- **콘텐츠 분석 (Content Analysis):** 대량의 텍스트에서 주요 주제의 동향(Trend)을 파악합니다.

4 Non-Negative Matrix Factorization (NMF)

NMF(Non-Negative Matrix Factorization, 비음수 행렬 분해)는 토픽 모델링에 사용되는 또 다른 주요 기법입니다. NMF는 LDA와 달리 확률 모델이 아니라, **행렬 분해(Matrix Decomposition)**라는 대수적(Algebraic) 접근 방식을 사용합니다.

4.1 NMF의 핵심 아이디어

NMF는 원본 문서-단어 행렬(V)을 두 개의 더 작은 행렬(W, H)의 곱으로 근사(Approximate)하는 것을 목표로 합니다.

$$V \approx W \times H$$

- V **(원본 행렬): [문서 × 단어]** 크기의 행렬. (예: m번째 문서의 n번째 단어 빈도수)
- W **(문서-토픽 행렬): [문서 × 토픽 개수** K**]** 크기의 행렬. (각 문서가 K개의 토픽을 얼마나 포함하는지 나타내는 가중치)
- H **(토픽-단어 행렬): [토픽 개수** K **× 단어]** 크기의 행렬. (각 토픽이 어떤 단어들로 구성되는지 나타내는 가중치)

여기서 "비음수(Non-Negative)"라는 이름이 붙은 이유는, 행렬 V, W, H의 **모든 원소가 0 이상**(≥ 0)이어야 한다는 제약 조건 때문입니다. 이는 '단어의 빈도'나 '토픽의 가중치'가 음수가 될 수 없다는 현실적인 직관과 잘 부합하여, 결과를 해석하기 쉽게 만듭니다.

4.2 NMF 예시

간단한 텍스트 데이터로 NMF가 어떻게 작동하는지 살펴봅시다.

- **텍스트 데이터:**
  - Doc 1: "cats meow"
  - Doc 2: "dogs bark"
  - Doc 3: "cats purr, dogs growl"
- **어휘 (Vocabulary):** "cats", "dogs", "meow", "bark", "purr", "growl"

이를 바탕으로 원본 행렬 V (3개 문서 × 6개 단어)를 구성합니다.

$$V = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

NMF는 이 V를 $K = 2$ (토픽 2개)로 분해하여 W (3×2)와 H (2×6)를 찾습니다.

$$V \approx W \times H$$

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \approx \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0.5 & 0.5 \end{bmatrix}}_{W\text{: 문서-토픽}} \times \underbrace{\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}}_{H\text{: 토픽-단어}}$$

결과 해석:

- H **(토픽-단어):**
  - 토픽 1 (H의 첫 행) = [1, 0, 1, 0, 1, 0] → "cats", "meow", "purr"와 강하게 연관 → **"고양이" 토픽**
  - 토픽 2 (H의 두 행) = [0, 1, 0, 1, 0, 1] → "dogs", "bark", "growl"와 강하게 연관 → **"강아지" 토픽**
- W **(문서-토픽):**
  - Doc 1 (W의 첫 행) = [1, 0] → "고양이" 토픽 100%, "강아지" 토픽 0%
  - Doc 2 (W의 두 행) = [0, 1] → "고양이" 토픽 0%, "강아지" 토픽 100%
  - Doc 3 (W의 세 행) = [0.5, 0.5] → "고양이" 토픽 50%, "강아지" 토픽 50%

NMF는 $V \approx WH$ 가 되도록 하는 W, H를 (수치 최적화를 통해) 근사적으로 찾아냄으로써, LDA와 유사하게 문서의 토픽 구성을 발견해냅니다.

4.3 LDA vs. NMF 비교

두 알고리즘은 토픽 모델링이라는 동일한 목표를 갖지만, 근본적인 접근 방식이 다릅니다.

Table 2: *LDA*와 *NMF*의 비교

특징	LDA (Latent Dirichlet Allocation)	NMF (Non-Negative Matrix Factorization)
접근 방식	**확률적 (Probabilistic)**	**대수적 / 행렬 분해 (Algebraic)**
기본 가정	문서 생성 과정에 대한 복잡한 확률 분포 (Dirichlet, Multinomial)를 가정함.	$V \approx WH$ 라는 비교적 단순한 행렬 분해(차원 축소)를 가정함.
결과 일관성	실행할 때마다 결과가 약간씩 다를 수 있음. (Stochastic)	동일한 조건에서는 (대체로) 동일한 결과. (Deterministic)
결과 해석	토픽의 **'확률'** 분포로 해석됨.	토픽의 **'가중치'**로 해석됨.
계산	상대적으로 복잡하고 느릴 수 있음. (MCMC, Variational Inference)	상대적으로 단순하고 빠를 수 있음. (단, K가 커지면 복잡)

5 실습: Python 및 R 구현

LDA와 NMF는 Python의 `scikit-learn` 라이브러리나 R의 `topicmodels` 라이브러리를 통해 쉽게 구현할 수 있습니다.

5.1 Python (scikit-learn) 구현 예제

먼저 문서를 단어 빈도 행렬(Document-Term Matrix, DTM)로 변환해야 합니다. `CountVectorizer`가 이 역할을 합니다.

```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

# 샘플문서
documents = [
    "Cats purr gently and climb high trees. Chasing a mouse is fun for cats.",
    "Independent creatures, cats enjoy solitude and love their nap time.",
    "Dogs bark loudly at strangers and fetch sticks with enthusiasm.",
    "Loyal dogs accompany humans on hikes and love to chase balls.",
    "When the energetic dog spotted a squirrel, it barked energetically.",
    "A purring cat climbed the bookshelf, watching over the room.",
    "Regularly, dogs enjoy long walks, sniffing and exploring their
       environment.",
    "Cats meow softly and purr when content, loving to stretch in the sun."
]

# 1. 문서단어- 행렬(DTM) 생성불용어 ( 제거)
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)

# 2. LDA 모델학습토픽  ( 개수 K로=2 설정)
# 는random_state 재현성을위한시드값
lda = LatentDirichletAllocation(n_components=2, random_state=0)
lda.fit(X)

# 3. 결과출력 : 토픽별상위개  5 단어
feature_names = vectorizer.get_feature_names_out()
for index, topic in enumerate(lda.components_):
    print(f"Topic {index + 1}:")
    # topic.argsort()[:-6:-1] : 상위개 5 단어의인덱스를뽑는구문
    top_words = [feature_names[i] for i in topic.argsort()[:-6:-1]]
    print(top_words)

# --- 결과예시  ---
# Topic 1:
# ['dogs', 'enjoy', 'long', 'walks', 'exploring']
# Topic 2:
# ['cats', 'purr', 'love', 'trees', 'mouse']
```

Listing 1: Python scikit-learn을 이용한 LDA 구현

```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import NMF

# 샘플문서위와 ( 동일)
documents = [
    "Cats purr gently and climb high trees. Chasing a mouse is fun for cats.",
    # ... 중략() ...
    "Cats meow softly and purr when content, loving to stretch in the sun."
]

# 1. DTM 생성
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)

# 2. NMF 모델학습토픽 ( 개수 K로=2 설정)
nmf_model = NMF(n_components=2, random_state=0, init='nndsvd')
# W: 문서토픽- 행렬, H: 토픽단어- 행렬
W = nmf_model.fit_transform(X)
H = nmf_model.components_

# 3. 결과출력 : 토픽별상위개 5 단어
feature_names = vectorizer.get_feature_names_out()
for index, topic in enumerate(H):
    print(f"Topic {index + 1}:")
    top_words = [feature_names[i] for i in topic.argsort()[:-6:-1]]
    print(top_words)

# --- 결과예시 ---
# Topic 1:
# ['cats', 'purr', 'high', 'trees', 'climb']
# Topic 2:
# ['dogs', 'love', 'enjoy', 'humans', 'loyal']
```

Listing 2: Python scikit-learn을 이용한 NMF 구현

5.2 R (topicmodels) 구현 예제

R에서는 `tm` 패키지로 텍스트를 전처리하고, `topicmodels` 패키지의 `LDA()` 함수를 사용합니다.

```r
# install.packages("tm")
# install.packages("topicmodels")
library(tm)
library(topicmodels)

# 샘플문서
documents <- c(
  "Cats purr gently and climb high trees. Chasing a mouse is fun for cats.",
  "Independent creatures, cats enjoy solitude and love their nap time.",
  "Dogs bark loudly at strangers and fetch sticks with enthusiasm.",
  "Loyal dogs accompany humans on hikes and love to chase balls.",
```

```r
  "When the energetic dog spotted a squirrel, it barked energetically.",
  "A purring cat climbed the bookshelf, watching over the room.",
  "Regularly, dogs enjoy long walks, sniffing and exploring their environment.",
  "Cats meow softly and purr when content, loving to stretch in the sun."
)

# 1. 텍스트전처리 (Corpus 생성)
corpus <- Corpus(VectorSource(documents))
corpus <- tm_map(corpus, content_transformer(tolower)) # 소문자
corpus <- tm_map(corpus, removePunctuation) # 구두점
corpus <- tm_map(corpus, removeNumbers) # 숫자
corpus <- tm_map(corpus, removeWords, stopwords("english")) # 불용어
corpus <- tm_map(corpus, stripWhitespace) # 공백

# 2. DTM 생성
dtm <- DocumentTermMatrix(corpus)

# 3. LDA 모델학습 (K=2)
lda_model <- LDA(dtm, k = 2, control = list(seed = 1234))

# 4. 결과출력 : 토픽별상위개 5 단어
terms_lda <- terms(lda_model, 5)
print(terms_lda)

# --- 결과예시 ---
#      Topic 1      Topic 2
# [1,] "dogs"       "cats"
# [2,] "enjoy"      "purr"
# [3,] "love"       "chasing"
# [4,] "bark"       "climb"
# [5,] "enthusiasm" "fun"
```

Listing 3: R을 이용한 LDA 구현

참고: R Markdown (.RMD) R 환경에서는 코드, 출력 결과, 설명을 하나의 문서로 통합 관리할 수 있는 **R Markdown (.RMD)** 형식을 자주 사용합니다. 이는 Python 환경의 Jupyter Notebook과 매우 유사하며, `knit` 버튼을 눌러 HTML, PDF, Word 등 원하는 형식의 보고서를 손쉽게 생성할 수 있습니다.

6 토픽 모델링 결과 해석 및 평가

토픽 모델링은 비지도 학습이므로, 모델이 '알아서' 토픽을 찾아줍니다. 하지만 이 결과가 유용한지, 그리고 가장 중요한 질문인 **"그래서 토픽 개수(K)를 몇 개로 정해야 하는가?"**는 전적으로 분석가의 몫입니다.

6.1 토픽 해석하기 (Labeling)

모델은 '토픽 1', '토픽 2'와 같이 번호만 부여합니다. 이 토픽이 실제로 무엇을 의미하는지(예: "강아지 토픽") 라벨을 붙이는 것은 사람이 해야 합니다. 두 가지 방법이 주로 사용됩니다.

방법 1: 토픽별 상위 단어 확인 (Top Words per Topic) 가장 직관적인 방법입니다. 위 코드 예제처럼 각 토픽을 구성하는 확률(가중치)이 높은 상위 단어들을 살펴봅니다.

예시:

- **Topic 1:** 'dogs', 'walks', 'enjoy', 'bark', 'loyal' ... → 해석: '강아지' 관련 토픽
- **Topic 2:** 'cats', 'purr', 'climb', 'mouse', 'meow' ... → 해석: '고양이' 관련 토픽

단점: 만약 불용어 처리가 미흡하거나 여러 주제에 공통적으로 등장하는 단어(예: 'love', 'enjoy')가 상위권에 나오면 토픽의 의미를 파악하기 모호해질 수 있습니다.

방법 2: 토픽별 상위 문서 확인 (Top Documents per Topic) 더욱 강력하고 정확한 방법입니다. 각 토픽이 **가장 높은 비율(Prevalence)**로 나타난 문서들을 직접 읽어보는 것입니다.

Python에서는 `lda.transform(X)`를 통해 각 문서의 토픽 분포를 확인할 수 있습니다.

예시:

- (문서 3) → [Topic 1: 93%, Topic 2: 7%]
- (문서 7) → [Topic 1: 94%, Topic 2: 6%]

해석: '토픽 1'의 비율이 90% 이상으로 압도적인 '문서 3'과 '문서 7'을 실제로 읽어봅니다.

- 문서 *3* 내용: "Dogs bark loudly at strangers..."
- 문서 *7* 내용: "Regularly, dogs enjoy long walks..."

→ 최종 결론: 두 문서 모두 '강아지'에 대해 이야기하므로, '토픽 1'은 **'강아지' 토픽**이라고 확신할 수 있습니다.

6.2 최적의 토픽 개수 (K) 결정하기

토픽 개수 K는 모델링 성능에 가장 큰 영향을 미치는 **하이퍼파라미터**입니다. K가 너무 작으면 여러 주제가 하나로 뭉개지고, K가 너무 크면 하나의 주제가 여러 개로 불필요하게 쪼개집니다.

최적의 K를 찾기 위해 **응집도(Coherence)**와 **배타성(Exclusivity)**이라는 두 가지 핵심 지표를 사용합니다.

핵심 평가 지표: 응집도(Coherence)와 배타성(Exclusivity) **1. 응집도 (Coherence)**

- **의미:** "하나의 토픽 내에 있는 상위 단어들이, 실제 원본 문서에서도 **자주 함께** 등장하는가?"
- **직관:** 좋은 토픽이라면 (예: '농구' 토픽), 상위 단어인 '농구', '선수', '코트', '슛'이 실제 문서에서도 자주 같이 등장해야 합니다.
- **판단:** 높을수록 좋습니다.

2. 배타성 (Exclusivity)

- **의미:** "하나의 토픽에 속한 상위 단어들이, **다른 토픽**들과 얼마나 겹치지 않고 **고유**하게 존재하는가?"
- **직관:** '농구' 토픽의 상위 단어가 '축구' 토픽에도 똑같이 나타난다면(예: '선수', '경기'), 두 토픽은 변별력이 없습니다.
- **판단:** 높을수록 좋습니다.

The Trade-off (트레이드오프): 안타깝게도 두 지표는 종종 반비례 관계에 있습니다.

- K가 너무 작으면 (예: $K = 2$), 토픽이 너무 광범위해져 응집도는 높지만 배타성이 낮아집니다. (예: '스포츠' 토픽 하나)
- K가 너무 크면 (예: $K = 100$), 토픽이 매우 세분화되어 배타성은 높지만 응집도가 낮아집니다. (예: 'A선수' 토픽, 'B선수' 토픽)

최적의 K 찾기: 일반적으로 K의 값을 2부터 50까지(예: 2, 5, 10, 15...) 변화시키면서 여러 모델을 실행한 뒤, **응집도(x축)와 배타성(y축)을 2D 그래프**로 그립니다.

두 지표가 모두 '적절하게' 높으면서(그래프의 우측 상단) 안정화되는 지점, 즉 **'엘보우 포인트(Elbow Point)'**에 해당하는 K를 최적의 값으로 선택합니다.

7 다음 학습: 구조적 토픽 모델링 (STM)

LDA는 강력하지만, 오직 '텍스트 본문'만을 사용하여 토픽을 추출합니다. 하지만 실제로는 텍스트 외에 **메타데이터(Metadata)**가 함께 주어지는 경우가 많습니다.

- 기사 (본문 + 기자, 작성일, 언론사)
- 상품평 (본문 + 사용자 성별, 연령대, 평점)
- 교수 평가 (본문 + 교수 성별, 개설 학과, 과목)

STM(Structural Topic Modeling, 구조적 토픽 모델링)은 LDA를 확장하여, 이러한 **메타데이터(Covariates, 공변량)**까지 모델링에 함께 포함시키는 기법입니다. (주로 R의 `stm` 패키지를 사용)

STM 활용 예시: 교수 평가 데이터 분석 약 100만 건의 교수 평가 텍스트와 메타데이터(교수 성별, 학과 등)를 STM으로 분석한 연구 사례가 있습니다.

분석: LDA처럼 텍스트만으로 토픽(예: '흥미로운 강의', '공정한 피드백', '배려심')을 추출할 뿐만 아니라, 이 토픽들이 **메타데이터와 어떤 상관관계**가 있는지 함께 분석합니다.

발견 (예시):

- **여성 교수**의 평가는 '배려심(caring)', '효과적인 토론 유도', '시기적절한 피드백'과 같은 토픽의 비중이 더 높게 나타났습니다.
- **남성 교수**의 평가는 '유머', '흥미롭고 관련성 높은 강의'와 같은 토픽의 비중이 더 높게 나타났습니다.

의의: 이러한 결과는 학생들이 교수의 성별에 따라 기대하거나 평가하는 방식에 잠재적인 편향(bias)이 존재할 수 있음을 시사합니다. STM은 이처럼 텍스트와 구조적 데이터를 결합하여 훨씬 더 깊이 있는 사회과학적 분석을 가능하게 합니다.

A 부록 A: 주요 용어 정리

Table 3: 주요 용어 정리표

용어	원어	쉬운 설명	비고
원-핫 인코딩	One-Hot Encoding	단어 사전에 있는 단 하나의 단어만 1로, 나머지는 0으로 표시하는 벡터.	희소(Sparse)하고 차원이 높음.
오토인코더	Autoencoder	입력을 저차원으로 압축(Encoder)했다가 다시 원본으로 복원(Decoder)하는 신경망.	비지도 학습.
병목	Bottleneck	오토인코더에서 차원이 가장 낮은 은닉층. 입력의 핵심 특징이 압축됨.	Encoding Vector가 생성되는 곳.
언더컴플리트	Undercomplete	입력 차원보다 병목(표현) 차원이 더 낮은 상태. (예: 100차원 → 10차원)	데이터 압축이 목적.
적층 오토인코더	Stacked Autoencoder	은닉층을 여러 개 깊게 쌓은(Stacked) 오토인코더.	Deep Network.
토픽 모델링	Topic Modeling	문서 집합에서 숨겨진(Latent) 주제(Topic)를 찾아내는 비지도 학습.	
잠재/숨겨진	Latent	데이터에 직접 드러나지 않고 숨어있는 변수나 구조. (예: Latent Topic)	
LDA	Latent Dirichlet Allocation	'문서는 토픽의 혼합, 토픽은 단어의 혼합'이라 가정하는 생성 확률 모델.	토픽 모델링의 대표 주자.
NMF	Non-Negative Matrix Factorization	원본 행렬(V)을 두 개의 비음수 행렬(W, H)의 곱으로 분해하는 기법.	$V \approx WH$. 대수적 접근.
응집도	Coherence	토픽 내 상위 단어들이 실제 문서에서 얼마나 자주 함께 등장하는지에 대한 지표.	높을수록 좋음.
배타성	Exclusivity	토픽 내 상위 단어들이 다른 토픽과 얼마나 겹치지 않는지에 대한 지표.	높을수록 좋음.
STM	Structural Topic Modeling	LDA에 메타데이터(성별, 날짜 등)를 결합하여 토픽을 분석하는 확장 모델.	
MLE	Maximum Likelihood Estimation	관찰된 데이터를 가장 잘 설명하는(등장 확률이 최대가 되는) 파라미터를 찾는 방법.	

B 부록 B: R 및 RStudio 설치 가이드

STM(구조적 토픽 모델링) 등 일부 고급 NLP 패키지는 Python보다 R에서 더 안정적으로 지원됩니다. R을 사용하기 위한 기본 환경 설치 단계는 다음과 같습니다.

1. **1단계: R (언어 본체) 설치**
   - R은 통계 계산과 그래픽을 위한 프로그래밍 '언어'이자 '환경'입니다.
   - 먼저 R 공식 웹사이트(CRAN)에 접속하여 본인의 운영체제(Windows, Mac, Linux)에 맞는 R을 다운로드하여 설치합니다.
   - https://www.r-project.org/
2. **2단계: RStudio (IDE) 설치**
   - RStudio는 R 언어를 더 쉽고 편리하게 사용할 수 있도록 도와주는 **통합 개발 환경(IDE)**입니다. (Python의 VS Code나 PyCharm과 유사)
   - RStudio를 사용하려면 반드시 1단계의 R이 **먼저** 설치되어 있어야 합니다.
   - Posit (구 RStudio) 웹사이트에서 RStudio Desktop (무료 버전)을 다운로드하여 설치합니다.
   - https://posit.co/download/rstudio-desktop/
3. **3단계: 패키지 설치**
   - RStudio를 실행한 뒤, 콘솔 창에 `install.packages("패키지명")` 명령어를 입력하여 필요한 패키지를 설치합니다.
   - 예: `install.packages("topicmodels")`, `install.packages("stm")`
   - 설치된 패키지는 `library(패키지명)` 명령어로 세션에 로드하여 사용합니다.

Cloud 버전 사용 설치가 번거롭다면, 웹 브라우저에서 바로 R을 사용할 수 있는 Posit Cloud (구 RStudio Cloud) 버전을 사용하는 것도 좋은 대안입니다.

C 부록 C: 1페이지 요약 (Quick Overview)

Autoencoder 복습

- **One-Hot Encoding**은 희소성(Sparsity)과 고차원 문제로 비효율적.

- **Autoencoder**는 입력을 저차원(Bottleneck)으로 압축(→ Encoder)했다가 복원(→ Decoder)하는 신경망.
- **핵심 목적:** 완벽한 복원이 아니라, 유용한 **저차원 표현(Encoding)**을 얻는 것.
- **Undercomplete:** 입력 차원 > 병목 차원. (가장 일반적인 형태)

Topic Modeling 이란?

- **정의:** 문서 집합에서 숨겨진(Latent) 주제를 찾는 비지도 학습.
- **Clustering과 차이:** 문서를 하나의 주제로 분류하는 대신 (Hard Assignment), **여러 토픽의 혼합(Mixture)**으로 표현함 (Soft Assignment).
- **예:** "이 문서는 [정치 70%, 경제 30%]이다."

LDA (Latent Dirichlet Allocation)

- **접근: 생성 확률 모델 (Probabilistic).**
- **가정 1:** 문서는 토픽의 확률 분포 ($\theta \sim Dirichlet$)
- **가정 2:** 토픽은 단어의 확률 분포 ($\phi \sim Dirichlet$)
- **추정:** MLE 또는 EM 알고리즘을 사용.
- **단점:** 텍스트 외의 메타데이터(작성자, 날짜 등)를 고려하지 못함.

NMF (Non-Negative Matrix Factorization)

- **접근: 행렬 분해 (Algebraic).**
- **가정:** $V \approx W \times H$
- V: [문서 × 단어] 원본 행렬
- W: [문서 × 토픽] 가중치 행렬
- H: [토픽 × 단어] 가중치 행렬
- **특징:** 모든 행렬이 비음수(Non-Negative)라서 해석이 직관적임.

모델 평가 및 선택 (K 정하기)

- **난관:** 최적의 토픽 개수 K는 하이퍼파라미터임.
- **지표 1: 응집도 (Coherence):** 토픽 내 단어들이 실제로 함께 자주 등장하는가? (높을수록 좋음)
- **지표 2: 배타성 (Exclusivity):** 토픽 내 단어들이 다른 토픽과 겹치지 않는가? (높을수록 좋음)
- **선택:** 두 지표가 모두 '적절히' 높아지는 K 값을 선택 (Trade-off 존재).

December 10, 2025

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 08

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 08의 핵심 개념 학습

Abstract

본 문서는 자연어 처리(NLP) 8강의 핵심 내용을 정리합니다. 주요 주제는 기존 LDA의 한계를 극복하는 **구조적 토픽 모델링(Structural Topic Modeling, STM)**입니다. STM이 어떻게 문서의 **메타데이터(metadata)**를 모델에 통합하여 더 풍부하고 정확한 분석을 가능하게 하는지 수학적 원리와 R 실습을 통해 배웁니다. 또한, 과제 수행 시 발생하는 텍스트 오토인코더의 어려움과 올바른 제출 가이드라인을 다루며, LDA와 NMF에 대한 퀴즈 복습을 포함합니다.

Contents

1 주요 공지 및 과제 가이드라인

본격적인 강의 내용에 앞서, 과제 제출과 관련된 주요 가이드라인을 명확히 합니다.

1.1 제출물 요구사항

과제 제출 시 다음 두 가지 구성 요소를 모두 제출해야 합니다.

1. **보고서 (Report): 단일 PDF 파일**
   - 최종 보고서는 **하나의(single) PDF 파일**이어야 합니다.
   - 각 문제 풀이를 별도의 Jupyter Notebook에서 수행했더라도, 각 노트북에서 생성된 PDF 파일들을 **하나로 병합하여** 제출해야 합니다.
   - Jupyter Notebook에서 직접 PDF를 생성하거나, 다른 워드 프로세서(예: MS Word)를 사용해 수동으로 보고서를 작성한 후 PDF로 변환할 수 있습니다.
2. **원본 코드 (Source Code)**
   - 보고서를 생성하는 데 사용된 모든 원본 코드를 제출해야 합니다.
   - (예: `.ipynb` Jupyter Notebook 파일, `.py` Python 스크립트 파일 등)
   - 여러 개의 코드 파일이 있는 경우, **하나의 `.zip` 파일**로 압축하여 제출하는 것을 강력히 권장합니다. (필수는 아니지만 관리에 용이함)

□ 핵심 요약

제출 요약:

- **파일 1 (필수):** 모든 결과가 포함된 **하나의 PDF 보고서**.
- **파일 2 (필수):** 모든 소스 코드를 담은 **하나의 `.zip` 파일** (또는 개별 코드 파일들).

Jupyter Notebook을 반드시 사용해야 하는 것은 아니지만, 코드와 리포트를 효율적으로 관리하는 데 유용할 수 있습니다.

2 과제 Q&A: 텍스트 오토인코더(Autoencoder)

많은 학생이 텍스트(예: 200단어 시퀀스)를 처리하는 오토인코더 과제에서 어려움을 겪고 있습니다.

2.1 Q: 텍스트 오토인코더로 원본 문장을 완벽하게 복원하기 매우 어렵습니다.

A: 네, 그것은 매우 예상된(expected) 결과입니다.

완벽하게 복원되지 않는다고 해서 모델이 잘못된 것은 아닙니다. 이 과제의 목적은 텍스트 오토인코더가 이미지 오토인코더보다 훨씬 더 어려운 과제임을 직접 경험하는 것입니다.

2.1.1 왜 텍스트 복원이 더 어려운가?

1. **극심한 병목 현상 (Severe Bottleneck)**
   - 표준적인 시퀀스-투-시퀀스(Seq2Seq) 오토인코더는 인코더(Encoder)가 전체 텍스트 시퀀스(예: 200단어)를 **하나의 단일 벡터(single vector)** (예: 인코더 RNN의 마지막 은닉 상태)로 압축합니다.
   - 이 단일 벡터가 전체 문장의 문맥, 순서, 의미를 모두 담아야 합니다.

- 디코더(Decoder)는 오직 이 단일 벡터 정보에만 의존하여 원본 시퀀스를 복원해야 하므로, 정보 손실이 막대하게 발생합니다.

2. **이미지 vs. 텍스트**
   - **이미지 (쉬움):** 컨볼루셔널 오토인코더(Convolutional Autoencoder)는 이미지를 압축할 때 공간적 구조(spatial structure)를 유지하며 점진적으로 특징을 추출합니다. 복원 시에도 이 구조를 역으로 따라가면 되므로 상대적으로 복원력이 뛰어납니다.
   - **텍스트 (어려움):** 텍스트는 순서와 문맥이 매우 중요한 데이터입니다. 이를 단일 벡터로 '뭉개버리면' 원본의 복잡한 순차 정보를 복원하기가 극도로 어려워집니다.

□ 핵심 요약

과제의 핵심은 "완벽한 복원"이 아니라, "텍스트 시퀀스를 단일 벡터로 압축하고 복원하는 것이 왜 어려운지, 그 한계가 무엇인지"를 이해하는 것입니다.

2.2 개선 시도 및 접근법

- **학생의 접근 (Smart Approach):** 디코더의 `softmax` 출력에서 확률이 가장 높은 단어(top-1)만 선택하는 대신, **온도 함수(temperature function)**를 사용하여 확률 분포를 조절하고, 확률이 높은 *주변*의 단어들을 샘플링(sampling)하는 방식입니다. → 교수의 평가: 매우 똑똑한 접근입니다. 원본과 정확히 같지는 않더라도 문법적으로나 의미적으로 일관된 영어 문장을 생성해낼 수 있습니다.
- **데이터 증강 (Data Augmentation):** 모델의 성능이 낮은 이유 중 하나는 훈련 데이터가 부족하기 때문일 수 있습니다. 이 경우 데이터 증강 기법을 사용할 수 있습니다.
  - **유의어(Synonym) 대체:** 문장 내 일부 단어를 유의어로 교체합니다.
  - **단어 삽입/삭제:** 문장의 일부 단어를 무작위로 삭제(drop)하거나 다른 단어를 삽입합니다.
  - **문장 순서 변경:** 문맥에 큰 영향이 없는 선에서 문장 순서를 바꿉니다.

□ 예제: title

놀랍게도, 이 '시퀀스를 단일 벡터로 압축하는' 오토인코더 구조는 초기 신경망 기계 번역(NMT)에서 실제로 시도되었던 방식입니다. (예: 영어 문장을 단일 벡터로 압축 → 프랑스어 문장으로 복원)
하지만 이 역시 동일한 병목 현상(bottleneck) 문제로 인해 긴 문장에서 성능이 급격히 저하되었고, 이후 '어텐션(Attention) 메커니즘'이 등장하면서 이 접근법은 사장되었습니다. 여러분이 겪은 어려움은 과거 연구자들도 동일하게 겪었던 문제입니다.

3 퀴즈 7 복습: LDA 및 NMF

퀴즈 7의 주요 개념인 LDA(잠재 디리클레 할당)와 NMF(음수 미포함 행렬 분해)를 복습합니다.

3.1 퀴즈 문항 및 정답

- **Q1 (LDA의 역할):** LDA가 무엇을 하는가? *A:* (C) 각 문서를 여러 토픽의 **혼합(mixture)**으로 간주합니다. (예: 이 문서는 정치 60
- **Q2 (LDA의 과제):** LDA 사용 시 흔히 겪는 어려움은? *A:* (B) 최적의 **토픽 수(K)**를 결정하는 것. (K는 하이퍼파라미터임)
- **Q3 (디리클레 분포의 역할):** LDA에서 디리클레(Dirichlet) 분포의 역할은? *A:* (A) 각 문서 내의 **토픽 비율(proportions)**(θ)을 생성하기 위한 사전 확률 분포(prior distribution)입니다.
- **Q4 (NMF의 역할):** NMF(음수 미포함 행렬 분해)에 대한 가장 좋은 설명은? *A:* (B) (음수가 아닌 항목을 가진) 행렬을 더 작은 행렬들로 분해합니다. (예: 문서-단어 행렬 $V \approx W \times H$)
- **Q5 (LDA vs. NMF 차이):** 두 접근법의 차이는? *A:* (A) LDA는 가능도(likelihood)를 최대화하는 **확률적 모델**인 반면, NMF는 행렬을 분해하는 **행렬 대수(matrix algebra)** 기법입니다.

3.2 LDA와 NMF 심층 비교

퀴즈 5번 항목에서 혼동이 있을 수 있습니다. LDA와 NMF는 모두 토픽 모델링에 사용될 수 있지만, 그 철학이 다릅니다.

아래 표는 LDA와 NMF의 핵심적인 차이를 요약합니다.

특징	LDA (Latent Dirichlet Allocation)	NMF (Non-negative Matrix Factorization)
기본 원리	**확률적 생성 모델** (Probabilistic)	**행렬 대수 분해** (Linear Algebra)
목표	문서가 생성되는 확률적 과정을 모델링합니다. (최대 가능도 추정, MLE)	원본 행렬(V)을 두 개의 작은 음수 미포함 행렬(W, H)의 곱($V \approx WH$)으로 근사합니다.
결과물	1. 문서별 토픽 분포 (θ): 문서 A는 토픽 1(40%), 토픽2(60%) 2. 토픽별 단어 분포 (β): 토픽1은 '고양이'(30%), '강아지'(20%)...	1. 토픽-단어 행렬 (W) 2. 문서-토픽 행렬 (H)
가정	문서는 토픽의 혼합, 토픽은 단어의 분포라는 명확한 생성 가정이 있습니다. (디리클레 사전 분포)	모든 행렬의 성분이 0 이상이어야 한다는 제약 조건만 있습니다.
해석	결과가 '확률' 또는 '비율'로 나와 해석이 매우 직관적입니다.	분해된 행렬을 '토픽'으로 해석합니다. 이때 관계는 덧셈(additive)이 아닌 곱셈(multiplicative)입니다.

Table 1: *LDA*와 *NMF*의 핵심 원리 비교

3.3 LDA의 비결정론적(Stochastic) 특성

LDA는 선형 회귀처럼 단 하나의 정답이 나오는 결정론적(deterministic) 모델이 아닙니다. LDA는 **확률적(stochastic) 알고리즘**입니다.

- **실행 시마다 결과가 다름:** LDA를 실행할 때마다 (특히 초기값이 다를 경우) 결과가 미묘하게 달라질 수 있습니다.
  - 토픽 1과 토픽 2의 순서가 바뀔 수 있습니다.
  - 토픽을 구성하는 단어의 클러스터가 약간 달라질 수 있습니다.
- **이유: 유일한 해가 없음 (No Unique Solution)** LDA는 최대 가능도 추정(MLE)을 사용하는데, 이 가능도(likelihood)를 최대화하는 해가 유일하지 않을 수 있습니다.

□ 예제: title

최대 가능도 추정(MLE)은 "관측된 데이터를 가장 잘 설명하는 원인(모델 파라미터)은 무엇인가?"를 찾는 과정입니다.

- **관측 (Data):** 고양이가 밖에서 **젖은 채로** 집에 돌아왔습니다.
- **가설 (Model):**
  1. 가설 A: 밖에 **비가 왔다.**
  2. 가설 B: 누군가 고양이에게 **물을 뿌렸다.**
- **MLE 추론:** "비가 왔을 때 고양이가 젖을 확률"과 "누군가 물을 뿌렸을 때 고양이가 젖을 확률"을 계산하여, 현재의 관측(젖은 고양이)을 가장 그럴듯하게 만드는 가설(예: 비가 왔다)을 선택합니다.

LDA도 마찬가지로, 우리가 관측한 '문서들(단어들의 집합)'을 가장 그럴듯하게 생성했을 토픽 분포(θ)와 단어 분포(β)를 역으로 추정하는 것입니다.

LDA 실행 시 실전 팁

LDA는 확률적이며 안정적이지 않기 때문에, 실제 분석에서는 다음과 같은 방법을 사용합니다.

1. **여러 번 실행 (Multiple Runs):** 동일한 K(토픽 수)에 대해 모델을 여러 번 실행합니다.
2. **최적 모델 선택:** 실행된 모델들 중에서 가장 "좋은" 모델을 선택합니다.
3. **평가 지표:** "좋은" 모델을 판단하는 기준은 다음과 같습니다.
   - **의미론적 일관성 (Semantic Coherence):** 토픽 내의 상위 단어들이 의미적으로 얼마나 관련성이 높은가? (예: '고양이', '강아지', '애완동물' ... → 높음)
   - **배타성 (Exclusivity):** 토픽들이 서로 얼마나 겹치지 않고 고유한 단어들을 갖는가?

4 구조적 토픽 모델링 (STM) 소개

4.1 LDA의 근본적인 한계: 메타데이터의 부재

지금까지 배운 LDA는 문서를 '단어의 가방(Bag-of-Words)'으로만 취급합니다. 즉, 문서에 포함된 단어 외의 **모든 맥락 정보를 무시**합니다.
하지만 실제 세계의 문서는 풍부한 **메타데이터(Metadata)**와 함께 제공됩니다.

- **뉴스 기사:** 저자, 출판 날짜, 언론사 (예: New York Times vs. Financial Times)
- **학술 논문:** 저자, 출판 연도, 학회지(Journal)
- **고객 리뷰:** 작성자, 평점(Rating), 작성 날짜
- **학생 강의평가:** 교수 성별, 개설 학과, 수강 시기 (예: 가을 vs. 봄 학기)

LDA는 이 중요한 메타데이터를 활용할 방법이 없습니다.

4.2 STM (Structural Topic Modeling)의 정의

구조적 토픽 모델링 (Structural Topic Modeling, STM)은 LDA를 확장하여, 이러한 문서 수준의 **메타데이터**를 모델에 직접 통합하는 프레임워크입니다.
메타데이터는 통계학 용어로 **공변량(Covariates)**이라고도 부릅니다.

□ 핵심 요약

STM의 핵심 아이디어:

- **LDA:** 문서의 토픽 비율(θ)은 모든 문서에 동일한 디리클레 분포($\text{Dir}(\alpha)$)에서 나온다.
- **STM:** 문서의 토픽 비율(θ_d)은 해당 문서의 메타데이터(X_d)에 따라 **달라진다.**

(예: '교수 성별'이라는 메타데이터가 '돌봄(caring)' 토픽의 비율에 영향을 줄 수 있다.)

5 STM의 수학적 원리

STM이 LDA와 어떻게 다른지 수학적 공식을 통해 비교합니다. 가장 큰 차이는 문서별 토픽 비율(θ)을 생성하는 부분입니다.

5.0.1 1. LDA의 토픽 유병률 (Topic Prevalence)

문서 m의 토픽 비율 θ_m은 디리클레 분포에서 직접 샘플링됩니다.

$$\theta_m \sim \text{Dirichlet}(\alpha)$$

여기서 α는 모든 문서에 동일하게 적용되는 하이퍼파라미터입니다.

5.0.2 2. STM의 토픽 유병률 (Topic Prevalence)

문서 d의 토픽 비율 θ_d는 해당 문서의 메타데이터 X_d에 의존하는 **로지스틱 정규 분포(Logistic-Normal Distribution)**를 따릅니다.

$$\theta_d | X_d\gamma, \Sigma \sim \text{LogisticNormal}(\mu = X_d\gamma, \Sigma)$$

이 공식이 생소해 보일 수 있지만, 두 단계로 나누어 생각하면 쉽습니다.

1단계 (회귀 분석): 먼저, 문서 d의 메타데이터(X_d)를 사용하여 토픽의 평균적인 방향(μ)을 계산합니다.

$$\mu = X_d\gamma$$

- X_d: 문서 d의 메타데이터 벡터 (예: '[1, date, author$_g ender, ...$]')
- γ (감마): 메타데이터가 각 토픽에 미치는 영향을 나타내는 **회귀 계수** (모델이 학습해야 할 파라미터)
- 이는 선형 회귀($y = X\beta$)와 매우 유사한 형태입니다.

2단계 (Softmax): 1단계에서 계산된 μ를 평균으로, Σ를 공분산으로 하는 **다변량 정규 분포(Multivariate Normal)**에서 벡터를 하나 샘플링합니다. 이 벡터를 **소프트맥스(Softmax) 함수**에 통과시켜 합이 1이 되는 확률 벡터(즉, 토픽 비율 θ_d)를 생성합니다. (이 과정을 합쳐 '로지스틱 정규 분포'라고 부릅니다.)

5.0.3 STM의 주요 파라미터

- γ **(감마):** 메타데이터(공변량)가 토픽 유병률에 얼마나 영향을 미치는지 나타내는 **회귀 계수**입니다. 이 값을 분석하는 것이 STM의 핵심입니다.
- Σ **(시그마):** 토픽 간의 **공분산 행렬**입니다. LDA와 달리, STM은 토픽들이 서로 **상관관계**를 가질 수 있다고 가정합니다. (예: '정치' 토픽과 '경제' 토픽은 자주 함께 등장 → 상관관계가 높음). 이 Σ 역시 모델이 데이터로부터 학습합니다.

6 STM의 장점 및 활용

- **메타데이터 통합:** 문맥 정보를 활용하여 더 정확하고 의미 있는 토픽을 추출합니다.
- **가설 검증 (Hypothesis Testing):** STM의 진정한 강점입니다. 모델이 추정한 γ 계수를 분석하여 사회과학적, 경영학적 가설을 검증할 수 있습니다.
  - 예1: "시간이 지남에 따라(date) '기후 변화' 토픽의 언급이 증가했는가?"
  - 예2: "여성 교수(gender)가 남성 교수보다 '학생 지원' 관련 토픽을 더 많이 언급받는가?"
  - 예3: "보수 언론(source)이 진보 언론보다 '세금 감면' 토픽을 더 많이 다루는가?"
- **활용 분야:** 미디어 프레이밍 분석, 여론 조사, 소셜 미디어 트렌드 추적, 고객 피드백 분석, 학술 연구 동향 파악 등 메타데이터가 존재하는 모든 텍스트 분석에 활용됩니다.

7 R 실습: stm 패키지

STM은 Python보다 R의 stm 패키지가 가장 표준적이고 강력한 구현체로 인정받고 있습니다. 본 강의에서는 R을 사용하여 STM을 실습합니다.

7.1 실습 1: "고양이 vs. 강아지" (저자 분석)

두 명의 저자가 각각 다른 주제(고양이, 강아지)에 대해서만 글을 쓴 간단한 예제입니다.

- **데이터:** 저자 1 (고양이 텍스트 8개), 저자 2 (강아지 텍스트 8개)
- **메타데이터 (X_d):** author (범주형 변수: "Author1", "Author2")

```r
# 1. 라이브러리로드
library(stm)

# 2. 데이터및메타데이터준비생략     ()
# documents <- c("Cats purr gently...", "Dogs bark loudly...", ...)
# metadata <- data.frame(author = rep(c("Author1", "Author2"), each = 8))

# 3. 텍스트전처리
# 는textProcessor 문서와메타데이터를함께처리하여
# 문서가삭제될경우메타데이터도함께동기화시킵니다      .
processed <- textProcessor(documents = documents, metadata = metadata)
out <- prepDocuments(processed$documents, processed$vocab, processed$meta)

# 4. STM 모델피팅
# K=2 토픽( 개2), prevalence = ~ author 저자( 변수를공변량으로사용   )
stm_model <- stm(documents = out$documents,
                 vocab = out$vocab,
                 K = 2,
                 prevalence = ~ author,  # 핵심: 메타데이터지정
                 data = out$meta,
                 max.em.its = 100,
                 init.type = "Spectral")

# 5. 결과요약
summary(stm_model)
# Topic 1 Top Words: dog, energet, bark, enjoy, chase...
# Topic 2 Top Words: cat, love, climb, purr, spot...

# 6. 메타데이터효과추정
effects <- estimateEffect(1:2 ~ author,
                          stmobj = stm_model,
                          metadata = out$meta,
                          uncertainty = "Global")

# 7. 효과시각화
plot(effects,
     covariate = "author",
     method = "difference",
     cov.value1 = "Author1",
```

```
    cov.value2 = "Author2",
    main = "Effect of Author Across Topics")
```

Listing 1: *R 코드: 간단한 STM 모델 피팅 (저자 분석)*

- **결과 해석:** summary 결과, 토픽 1은 '강아지' 관련, 토픽 2는 '고양이' 관련 단어로 명확히 분리됩니다. plot(effects, ...) 결과는 "Author1이 Author2에 비해 토픽 2(고양이)를 더 많이 사용하고, 토픽 1(강아지)을 덜 사용한다"는 것을 시각적으로 보여줍니다.

7.2 실습 2: "Kaggle 뉴스 헤드라인" (시계열 분석)

100만 개 이상의 호주 뉴스 헤드라인 데이터를 사용하여 시간에 따른 토픽 트렌드를 분석합니다.

- **데이터:** 뉴스 헤드라인 텍스트
- **메타데이터 (X_d):** publish_date (날짜, 예: 20030219)

7.2.1 데이터 전처리: 날짜(Date) 공변량 다루기

날짜 데이터를 STM에서 공변량으로 사용하는 방법은 두 가지가 있습니다.

1. **범주형(Categorical) 변수:** 모든 날짜(예: '2003-02-19', '2003-02-20'...)를 별개의 범주로 취급합니다. (단점: 변수의 수가 너무 많아져(수천 수만 개) 계산 비용이 매우 비싸지고(*expensive*) 분석이 어려워짐.)
2. **연속형(Continuous) 수치 변수:** 날짜를 **하나의 숫자**로 변환합니다. (예: '2003.0 + (month-1)/12') (장점: 변수가 하나이므로 계산이 효율적임.) (가정: 이 방식을 사용하면, 토픽의 유병률이 시간에 따라 **선형적(*linear*)으로** 증가하거나 감소한다고 **가정**하는 것입니다.)
   본 실습에서는 **2번 (연속형 수치 변수)** 방식을 사용합니다.

7.2.2 전처리 시 중요사항: 빈 문서 자동 제거

textProcessor와 prepDocuments 함수는 텍스트를 정리(불용어, 숫자, 구두점 제거)합니다.

데이터 정합성(Consistency) 유지

- 텍스트 정리 과정에서, 어떤 문서는 내용이 **완전히 비게(empty)** 될 수 있습니다 (실습 예제에서 50개 문서).
- prepDocuments는 이 빈 문서들을 **자동으로 제거**합니다.
- **치명적 오류 방지:** 만약 원본 메타데이터를 그대로 stm 모델에 전달하면, 문서 수(줄어듦)와 메타데이터 행 수(그대로)가 일치하지 않아 오류가 발생합니다.
- **해결책:** prepDocuments가 반환한 **out$meta**를 새로운 메타데이터로 사용해야 합니다. out$meta는 빈 문서에 해당하는 행이 이미 제거된, 정제된 메타데이터입니다.

```
# 잘못된사용
meta <- original_metadata
out <- prepDocuments(...)
stm_model <- stm(..., data = meta) # 오류! 행개수가안맞음

# 올바른사용
out <- prepDocuments(..., metadata = original_metadata)
meta_synced <- out$meta # <--- 반드시 out$를meta 사용
stm_model <- stm(..., data = meta_synced) # 정상작동
```

7.2.3 시계열 효과 추정 및 시각화

```r
# 1. STM 모델피팅 (K=5, 날짜를공변량으로 )
# (date_는numeric 2003.083... 같이변환된수치형날짜 )
stm_model_news <- stm(...,
                      K = 5,
                      prevalence = ~ date_numeric,
                      data = meta_synced)

# 2. 효과추정
# 는estimateEffect 모델의후방분포 (posterior distribution)에서
# 샘플링하여 date_에numeric 따른를 prevalence 회귀분석합니다 .
effects_news <- estimateEffect(1:5 ~ date_numeric,
                               stmobj = stm_model_news,
                               metadata = meta_synced,
                               uncertainty = "Global")

# 3. 시계열트렌드시각화
plot(effects_news,
     covariate = "date_numeric",
     method = "continuous", # <--- 연속형변수지정
     topics = 1:5,
     xlab = "Year",
     main = "Topic Prevalence Over Time")
```

Listing 2: *R* 코드*: STM* 시계열 효과 추정

- **결과 해석:** 생성된 플롯은 시간에 따른 각 토픽의 유병률 **추세(trend)**와 **신뢰 구간(confidence interval)**을 보여줍니다.
  - (예: 토픽 1 (Council, Govt)은 시간이 지남에 따라 유병률이 **증가**하는 선형 추세를 보임.)
  - (예: 토픽 3 (Politics)은 시간이 지남에 따라 유병률이 **감소**하는 선형 추세를 보임.)

8 최적의 토픽 수 (K) 찾기

지금까지 K(토픽 수)를 임의로 (K=2, K=5) 지정했습니다. 하지만 LDA와 마찬가지로 STM에서도 최적의 K를 찾는 것은 매우 중요한 하이퍼파라미터 튜닝 과정입니다.

8.1 방법: searchK() 함수

stm 패키지는 searchK() 함수를 제공하여, 여러 K 값에 대한 모델 성능을 한 번에 비교할 수 있게 합니다.

```r
# K부터=2 까지10 모델을모두실행하고성능을비교
k_search_results <- searchK(documents = out$documents,
                            vocab = out$vocab,
                            K = c(2, 3, 4, 5, 6, 7, 8, 9, 10),
                            prevalence = ~ date_numeric,
                            data = meta_synced)

# 결과시각화
plot(k_search_results)
```

Listing 3: *R 코드: 최적의 K 탐색*

주의사항

searchK()는 지정된 모든 K 값에 대해 STM 모델을 (여러 번) 실행하고 평가합니다. 데이터가 크면 **매우 오랜 시간** (수 시간 ~ 수 일)이 소요될 수 있습니다.

8.2 평가 지표: 일관성(Coherence) vs. 배타성(Exclusivity)

searchK()는 여러 지표를 보여주지만, 토픽의 품질을 평가하는 데 가장 중요한 두 가지 지표는 다음과 같습니다.

의미론적 일관성 (Semantic Coherence)
- **의미:** 토픽 내의 상위 단어들이 의미적으로 얼마나 관련성이 높은가?
- **예시:** '고양이', '강아지', '애완동물', '먹이' ... → **일관성 높음**
- (높을수록 좋음)

배타성 (Exclusivity)
- **의미:** 토픽들이 서로 얼마나 겹치지 않고 고유한(unique) 단어들을 갖는가?
- **예시:** 토픽 A: '고양이', '야옹', '집사' / 토픽 B: '강아지', '멍멍', '산책' → **배타성 높음**
- (높을수록 좋음)

최적의 K 선택 전략

1. **트레이드오프 (Trade-off):** 일반적으로 K가 너무 작으면 일관성은 높지만 배타성이 낮고, K가 너무 크면 배타성은 높지만 일관성이 낮아지는 경향이 있습니다.
2. **시각화 및 선택:** X축을 '배타성', Y축을 '의미론적 일관성'으로 하는 2D 플롯을 그립니다. **가장 오른쪽 위 (Top-Right)**에 위치하는, 즉 두 지표가 모두 가장 높은 K 값을 선택하는 것이 이상적입니다.
3. **정규화 (Rescaling) 팁:** 두 지표는 스케일이 매우 다를 수 있습니다 (예: 일관성 9 vs. 배타성 -260). 따라서 각 지표를 [0, 1] 범위로 정규화(rescale)한 뒤, **두 정규화된 값의 평균**을 최대화하는 K를 선택하는 것이 합리적인 방법입니다.

9 토픽 해석하기: "토픽 1"은 무엇인가?

모델을 훈련하고 최적의 K를 찾아도, "토픽 1", "토픽 2" 등은 그저 숫자에 불과합니다. 이 토픽들이 실제 어떤 의미를 갖는지 해석하는 과정이 필요합니다.

9.1 방법 1: 상위 단어 확인 (`summary()` 또는 `labelTopics()`)

- **내용:** 각 토픽에서 등장 확률이 가장 높은 단어들을 봅니다.
- **예시:** (토픽 1: cat, purr, climb...), (토픽 2: dog, bark, fetch...)
- **단점:** (실습 2의 경우) 'australia', 'council', 'govt' 등 모든 토픽에 공통적으로 등장하거나 의미 파악에 도움이 안 되는 단어들이 상위를 차지할 수 있습니다.

9.2 방법 2: 대표 문서 확인 (`findThoughts()`) - 권장

가장 권장되는 방법입니다. `findThoughts()` 함수는 각 토픽에 대해 **가장 높은 유병률(θ)을 갖는 원본 문서(들)**을 직접 보여줍니다.

```
# 토픽에 3 대해가장대표적인문서헤드라인    () 개를2 보여줌
findThoughts(stm_model_news,
             texts = original_headlines, # 원본텍스트
             n = 2,
             topics = 3)
```

Listing 4: *R 코드: 대표 문서로 토픽 해석*

- **해석 과정:**
  1. `findThoughts()`를 실행하여 토픽 3의 대표 헤드라인을 읽습니다.
  2. (예: "Police investigate crash on highway", "Man charged over stabbing incident")
  3. 이 문서들을 읽어본 분석가는 "아, 토픽 3은 **'사건/사고 및 범죄'**에 관한 토픽이구나"라고 **수동으로 레이블(label)을 붙일 수 있습니다.**

□ 예제: title

`findThoughts`는 모델의 `theta` (θ) 행렬을 기반으로 작동합니다.

- `theta`는 (문서 수 × 토픽 수) 크기의 행렬입니다.
- `theta[d, k]` 값은 d번째 문서에서 k번째 토픽이 차지하는 비율(유병률)을 의미합니다.
- `findThoughts(..., topics=3)`는 `theta` 행렬의 3번째 열(토픽 3)에서 값이 가장 큰 행(문서)을 찾아, 그 문서의 원본 텍스트를 보여주는 것입니다.

10 용어 정리

용어 (원어)	쉬운 설명	비고 (관련 개념)
LDA (Latent Dirichlet Allocation)	문서가 여러 토픽의 혼합으로 이루어져 있다고 가정하는 확률적 토픽 모델.	메타데이터를 사용하지 못함.
STM (Structural Topic Modeling)	LDA를 확장하여, 문서의 **메타데이터**가 토픽 비율에 영향을 미친다고 보는 고급 토픽 모델.	R `stm` 패키지.
메타데이터 (Metadata)	텍스트 자체는 아니지만 텍스트에 대한 정보. (예: 저자, 날짜, 출처)	STM에서는 '공변량(Covariate)'이라고도 부름.
공변량 (Covariate)	분석 대상(토픽 비율)에 영향을 줄 수 있는 외부 변수. (예: X_d)	통계학 용어. 메타데이터와 거의 동일한 의미로 사용됨.
토픽 유병률 (Topic Prevalence)	특정 문서 또는 문서 집합에서 특정 토픽이 차지하는 비율 또는 중요도.	θ (세타) 값.
로지스틱 정규 분포 (Logistic-Normal)	다변량 정규 분포에서 샘플링한 벡터를 Softmax 함수에 통과시켜 합이 1인 확률 벡터를 얻는 분포.	STM에서 θ_d를 생성하는 방식.
최대 가능도 추정 (MLE, Max. Likelihood Est.)	관측된 데이터를 가장 그럴듯하게 설명하는(생성할 확률이 가장 높은) 모델 파라미터를 찾는 통계적 방법.	"젖은 고양이" 비유.
확률적/비결정론적 (Stochastic)	무작위성을 포함하므로 실행할 때마다 결과가 달라질 수 있음.	LDA, STM 모두 해당.
의미론적 일관성 (Semantic Coherence)	토픽 내의 단어들이 의미적으로 얼마나 일관된지를 나타내는 지표.	(높을수록 좋음)
배타성 (Exclusivity)	토픽들이 서로 얼마나 겹치지 않고 고유한 단어들로 구성되었는지를 나타내는 지표.	(높을수록 좋음)
NMF (Non-negative Matrix Fact.)	음수 미포함 행렬(V)을 두 개의 작은 행렬($W \times H$)의 곱으로 분해하는 기법.	LDA와 달리 확률 모델이 아님.

Table 2: 주요 용어 정리

11 FAQ (주요 질문 및 답변)

Q: STM을 꼭 R로만 해야 하나요? Python은 없나요?

A: `stm` 패키지는 R에서 개발되었고, 가장 많은 기능과 안정성을 제공하며 학계에서도 표준으로 사용됩니다. Python에 일부 구현체가 존재하긴 하지만(GitHub 등), R 패키지만큼 신뢰할 수 있거나 기능이 풍부하지 않은 경우가 많습니다.
본 강의에서는 안정적인 분석과 `estimateEffect` 같은 강력한 효과 추정 기능을 위해 R 사용을 권장합니다.

Q: `searchK()`를 실행하는 데 시간이 너무 오래 걸립니다.

A: 정상입니다. `searchK()`는 (K의 개수 × 실행 횟수)만큼 STM 모델을 처음부터 끝까지 훈련시킵니다.
실제 분석에서는 (1) 데이터의 일부를 샘플링하여 빠르게 K의 범위를 좁히거나, (2) 밤새도록 또는 며칠간 실행할 것을 예상해야 합니다.

Q: 메타데이터로 날짜를 사용할 때, '연속형'과 '범주형' 중 무엇이 더 좋은가요?

A: 정답은 없습니다.

- **연속형 (예: `year + (m-1)/12`):** "토픽이 시간에 따라 선형적으로 증가/감소한다"는 강한 가정을 합니다. 트렌드를 부드럽게 볼 수 있지만, 계절성이나 특정 이벤트(예: 선거)로 인한 급격한 변화를 놓칠 수 있습니다.
- **범주형 (예: `as.factor(year)`):** "연도별로 토픽 비율이 자유롭게 다를 수 있다"고 가정합니다. 더 유연하지만, 해석이 복잡해지고 더 많은 데이터가 필요합니다.

분석의 목적에 맞게 선택해야 합니다. "장기적 트렌드"를 보려면 연속형, "특정 연도의 차이"를 보려면 범주형이 적합할 수 있습니다.

Q: 전처리 후 문서가 50개나 사라졌습니다. 괜찮은가요?

A: 네, 괜찮습니다. 뉴스 헤드라인처럼 매우 짧은 텍스트는 불용어, 숫자, 구두점 등을 제거하고 나면 내용이 완전히 비게 되는 경우가 흔합니다.
`textProcessor`와 `prepDocuments`는 이런 빈 문서들을 자동으로 제거해 주므로 편리합니다.
단, `out$meta`를 사용해야 한다는 점을 절대 잊지 마세요. (데이터 정합성)

December 10, 2025

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 09

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 09의 핵심 개념 학습

Contents

1 개요 (Overview)

□ 핵심 요약

이 문서는 자연어 처리(NLP) 작업, 특히 텍스트 분류를 수행하기 위한 **고전적 기계학습(Classical Machine Learning) 모델**들을 다룹니다.
데이터가 아주 많지 않을 때(e.g., 수천 수만 건) 딥러닝보다 효과적일 수 있는 **Naive Bayes, KNN, Logistic Regression, SVM, Random Forest**의 핵심 원리와 장단점을 초심자의 시선에서 설명합니다.
특히, 현실의 **불균형 데이터(imbalanced data)**를 다룰 때 단순 '정확도'의 함정을 피하고, **K-겹 교차 검증(K-Fold CV), 파이프라인(Pipeline), 혼동 행렬(Confusion Matrix)**을 활용하여 모델을 올바르게 평가하고 최적화하는 실전 방법론에 초점을 맞춥니다.

2 학습 로드맵 및 주요 용어 정리

2.1 학습 로드맵

이 노트를 효과적으로 학습하기 위한 권장 순서입니다.

1. **텍스트의 숫자 변환**: 왜 TF-IDF가 필요한지 이해합니다.
2. **모델 검증의 중요성**: 왜 단순 8:2 분할 대신 K-겹 교차 검증(K-Fold CV)을 쓰는지 이해합니다.
3. **평가 지표의 함정**: 왜 '정확도(Accuracy)'가 아닌 '정밀도(Precision)'와 '재현율(Recall)'이 중요한지 이해합니다.
4. **개별 모델 학습**: 각 모델(Naive Bayes, KNN, SVM 등)의 기본 작동 원리를 비유를 통해 이해합니다.
5. **실습 사례 분석**: BBC 뉴스 분류 예제에서 어떤 모델이 왜 더 좋은 성능을 보였는지 확인합니다.

2.2 주요 용어 정리표

이 문서에서 자주 등장하는 핵심 용어들입니다.

Table 1: 핵심 용어 정리

용어 (한글)	용어 (원어)	쉬운 설명	비고
TF-IDF	Term Frequency-Inverse Document Frequency	"이 문서에선 자주 나오지만, 다른 데선 잘 안 나오는 단어"에 높은 점수를 줌	텍스트의 숫자화
K-겹 교차 검증	K-Fold Cross-Validation	데이터를 K개로 쪼개서, 한 조각씩 돌아가며 테스트하고 평균내는 검증법	데이터가 적을 때 유용
파이프라인	Pipeline	데이터 전처리(e.g., TF-IDF)와 모델 학습을 묶어주는 통로	데이터 누수 방지
혼동 행렬	Confusion Matrix	모델이 무엇을 맞혔고(TP, TN), 무엇을 틀렸는지(FP, FN) 보여주는 표	평가의 시작
정확도	Accuracy	(TP + TN) / 전체. **불균형 데이터에선 함정이 될 수 있음**	100개 중 99개 맞힘
정밀도	Precision	TP / (TP + FP). "모델이 '양성'이라고 한 것 중, **진짜 '양성'**일 확률"	스팸 필터 (신뢰도)
재현율 / 민감도	Recall / Sensitivity	TP / (TP + FN). "**실제 '양성'**인 것 중, 모델이 **찾아낸** 비율"	암 진단 (누락 방지)
특이도	Specificity	TN / (TN + FP). "**실제 '음성'**인 것 중, 모델이 **'음성'**이라고 맞춘 비율"	–
하이퍼파라미터	Hyperparameter	모델이 학습하기 전에 **사람이 직접 설정**해줘야 하는 값	e.g., KNN의 'k' 값
GridSearchCV	Grid Search Cross-Validation	하이퍼파라미터 후보들을 **모두 조합**해보고 CV로 최고를 찾는 도구	자동 최적화

3 왜 딥러닝 대신 고전적 모델을 사용하나요?

최근 NLP 분야는 딥러닝(Deep Learning) 모델들이 주도하고 있지만, 고전적 기계학습 모델은 여전히 중요하며 특정 상황에서 더 유리합니다.

- **데이터 크기**: 딥러닝 모델은 성능을 내기 위해 **아주 많은 데이터**가 필요합니다. 데이터가 수천 수만 건 정도로 비교적 적다면, 딥러닝 모델은 과적합(Overfitting)되기 쉬운 반면, 고전적 모델들(특히 SVM)이 더 안정적이고 우수한 성능을 보일 때가 많습니다.
- **학습 속도 및 비용**: 고전적 모델은 딥러닝 모델보다 훨씬 **빠르게 훈련**됩니다. 복잡한 GPU 환경 설정 없이 일반 CPU로도 충분히 빠릅니다.
- **해석 용이성**: Logistic Regression이나 Decision Tree 같은 모델은 어떤 단어(Feature)가 분류에 큰 영향을 미쳤는지 비교적 쉽게 해석할 수 있습니다. (물론 SVM(커널 사용 시)이나 Random Forest는 딥러닝처럼 해석이 어려운 '블랙박스'에 가깝습니다.)

4 핵심 절차 1: 텍스트 벡터화 (TF-IDF)

기계학습 모델은 '뉴스 기사'라는 텍스트 자체를 이해할 수 없습니다. 오직 숫자, 즉 **벡터(Vector)**만을 입력으로 받습니다. 따라서 텍스트를 숫자로 변환하는 과정이 반드시 필요합니다.

가장 널리 쓰이는 방법이 **TF-IDF**입니다.

- **TF (Term Frequency, 단어 빈도)**: 한 문서 내에서 특정 단어가 얼마나 **자주** 등장했는지를 나타냅니다. (e.g., 이 문서에서 '모델'이라는 단어의 TF 값은 높습니다.)
- **IDF (Inverse Document Frequency, 역문서 빈도)**: 특정 단어가 전체 문서 집합에서 얼마나 **희귀**한지를 나타냅니다. 모든 문서에 다 나오는 단어(e.g., 'the', 'a', '이다')는 희귀성이 낮아 IDF 값이 0에 가깝습니다. 반면, 특정 주제의 문서에서만 나오는 단어(e.g., 'SVM', '커널')는 희귀성이 높아 IDF 값이 큽니다.

TF-IDF 값 = TF × IDF

결과적으로, 해당 문서에서 **자주 등장**하면서 (높은 TF) 다른 문서에서는 **잘 안 나오는** (높은 IDF) 단어일수록 그 문서를 대표하는 **중요한 키워드**로 인식되어 높은 TF-IDF 점수를 받게 됩니다.

5 핵심 절차 2: 모델 훈련 및 검증

5.1 단순 분할 vs. K-겹 교차 검증 (K-Fold CV)

문제점: 2,225개의 뉴스 기사 데이터가 있다고 가정해봅시다. (e.g., 훈련 80%, 테스트 20%로 분할)

이 방식은 **어떤** 데이터를 테스트용으로 뽑았는지에 따라 모델 성능이 **우연히** 좋거나 나쁘게 나올 수 있습니다. (결과가 불안정함) 또한, 전체 데이터의 20%(약 445개)를 테스트용으로 "낭비"하게 됩니다. 데이터가 많지 않을수록 이 "낭비"는 뼈아픕니다.

□ 예제:

해결책: K-겹 교차 검증 (K-Fold Cross-Validation, k=5 예시)
"낭비"되는 데이터 없이, 모든 데이터를 훈련과 검증에 골고루 사용하는 방법입니다.

1. 전체 데이터를 K개(e.g., 5개)의 "폴드(Fold)" 또는 "조각"으로 나눕니다.
2. **실행 1:** 1 4번 조각으로 **훈련** → 5번 조각으로 **테스트** (성능 기록)
3. **실행 2:** 1 3, 5번 조각으로 **훈련** → 4번 조각으로 **테스트** (성능 기록)
4. **실행 3:** 1 2, 4 5번 조각으로 **훈련** → 3번 조각으로 **테스트** (성능 기록)
5. **실행 4:** 1, 3 5번 조각으로 **훈련** → 2번 조각으로 **테스트** (성능 기록)
6. **실행 5:** 2 5번 조각으로 **훈련** → 1번 조각으로 **테스트** (성능 기록)

최종 결과: 5번의 테스트 성능 점수(e.g., [95, 97, 96, 98, 96])의 **평균**을 냅니다. 이 평균값은 단 한 번의 8:2 분할보다 훨씬 **안정적이고 신뢰할 수 있는** 모델 성능 추정치입니다.

5.2 `Pipeline`의 중요성: 데이터 누수(Data Leakage) 방지

K-Fold CV의 치명적 함정: 데이터 누수 (Data Leakage)

만약 K-Fold CV를 **하기 전에, 전체 2,225개**의 데이터로 `TfidfVectorizer`를 `fit` (학습) 시켰다고 가정해봅시다.

이는 훈련(1 4번 조각)을 할 때, **미리 테스트 데이터(5번 조각)의 정보(IDF 값)를 엿본 셈**이 됩니다. 이는 현실에서 일어날 수 없는 일이며, 모델 성능이 비정상적으로 높게 나오는 원인이 됩니다. (반칙!)

올바른 절차: 매 실행마다(총 5번), **오직 훈련용 조각(e.g., 1 4번)만**으로 `TfidfVectorizer`를 **새롭게** `fit` 하고, 그것으로 테스트 조각(e.g., 5번)을 `transform` 해야 합니다.

이 복잡하고 반복적인 "올바른 절차"를 아주 쉽게 자동화해주는 도구가 바로 `sklearn`의 **`Pipeline`**입니다.

`Pipeline`은 (1) TF-IDF 벡터화, (2) 분류 모델 학습 이 두 단계를 하나의 "파이프"로 묶어줍니다. 이 `Pipeline` 객체를 `cross_val_score` 같은 교차 검증 함수에 전달하면, 함수가 **알아서** 매 실행마다 훈련용 조각으로만 벡터화(fit)를 수행하여 데이터 누수를 완벽하게 방지합니다.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score

# 1. Pipeline 정의: 단계를2 하나로묶음
```

```
# 'vec': TF-IDF 벡터화단계
# 'clf': Naive Bayes 분류기단계
text_clf_pipeline = Pipeline([
    ('vec', TfidfVectorizer()),
    ('clf', MultinomialNB()),
])

# 2. 을Pipeline 교차검증기에전달
# X_data: 전체텍스트데이터   (e.g., 개2225 기사)
# y_data: 전체레이블  (e.g., 개2225 0 또는 1)
# cv=5: 겹5- 교차검증을수행
# 이과정에서데이터누수없이번의      5 fit/이transform 자동으로일어남
scores = cross_val_score(text_clf_pipeline, X_data, y_data, cv=5, scoring='
    accuracy')

print(f"교차 검증평균정확도  : {scores.mean():.4f}")
```

Listing 1: sklearn의 Pipeline을 사용한 교차 검증 예시

5.3 GridSearchCV를 이용한 하이퍼파라미터 최적화

- **하이퍼파라미터란?** 모델이 데이터로부터 "학습"하는 값(e.g., Logistic Regression의 가중치 w)이 아니라, 우리가 모델에게 **미리 알려줘야 하는 설정값**입니다. (e.g., KNN의 k 값, SVM의 C 값과 'kernel' 종류) 이 설정값에 따라 모델 성능이 크게 달라집니다.
- **GridSearchCV란?** "Grid Search" + "Cross-Validation"의 합성어입니다. 우리가 시도해보고 싶은 하이퍼파라미터 후보들을 **모두 조합**하여 (Grid Search), 각 조합마다 **K-Fold CV**를 수행하여 가장 성능이 좋았던 **최고의 조합**을 찾아줍니다.

□ 예제:

KNN 모델의 최적 k 찾기 예시
GridSearchCV에게 "k 값으로 1, 3, 5, 10, 15, 19를 테스트해보고, 각 k 마다 5-Fold CV를 돌려서 평균 점수가 가장 높은 k를 알려줘"라고 명령하는 것과 같습니다.

□ 예제:

SVM 모델의 최적 C와 'kernel' 찾기 예시
후보군: 'C = [0.1, 1, 10]', 'kernel = ['linear', 'rbf']' GridSearchCV는 아래 6가지 조합을 모두 테스트합니다.

- (C=0.1, kernel='linear') → 5-Fold CV → 평균 점수 1
- (C=0.1, kernel='rbf') → 5-Fold CV → 평균 점수 2
- (C=1, kernel='linear') → 5-Fold CV → 평균 점수 3
- (C=1, kernel='rbf') → 5-Fold CV → 평균 점수 4
- (C=10, kernel='linear') → 5-Fold CV → **평균 점수 5 (최고!)**
- (C=10, kernel='rbf') → 5-Fold CV → 평균 점수 6

최종 결과: "최고의 조합은 'C=10', 'kernel='linear'' 입니다."

최종 모델 훈련

`GridSearchCV`는 최적의 하이퍼파라미터를 **찾아주는** 도구입니다. 최고의 조합(e.g., 'C=10, kernel='linear' ')을 찾았다면, 그 설정으로 **전체 훈련 데이터**를 다시 학습시켜 **단 하나의 최종 모델**을 만들어야 합니다.

6 핵심 절차 3: 불균형 데이터 평가하기

6.1 정확도(Accuracy)의 함정

문제 상황: "테크" 뉴스(401개)와 "비-테크" 뉴스(1824개)를 분류하는 문제. 데이터 비율이 약 1:4.5로 **불균형(Imbalanced)**합니다.

함정: 만약 어떤 모델이 **모든 기사를 무조건 "비-테크"라고만 예측**한다면, 이 "멍청한" 모델은 1824개는 맞히고 401개는 틀리게 됩니다. 이 모델의 **정확도(Accuracy)**는 $1824/(1824+401) = 81.9\%$ 입니다.

81.9%라는 높은 정확도 숫자만 보면 이 모델이 꽤 쓸만하다고 **오해**할 수 있지만, 우리의 원래 목적인 "테크" 기사를 **단 하나도** 찾아내지 못하는 완전히 쓸모없는 모델입니다.

6.2 해결책: 혼동 행렬 (Confusion Matrix)

불균형 데이터를 평가할 때는, 모델이 "무엇을" 맞혔고 "무엇을" 틀렸는지 세부적으로 파악해야 합니다. 이때 사용하는 것이 **혼동 행렬**입니다.

("테크" 뉴스를 '양성(Positive)', "비-테크" 뉴스를 '음성(Negative)'이라고 가정)

Table 2: 혼동 행렬 (*Confusion Matrix*)

		모델의 예측 (Predicted)	
		'테크' (Positive)	'비-테크' (Negative)
실제 값 (Actual)	'테크' (Positive)	**TP (True Positive)** (진짜 '테크'를 '테크'로 맞힘)	**FN (False Negative)** (진짜 '테크'를 '비-테크'로 놓침)
	'비-테크' (Negative)	**FP (False Positive)** ('비-테크'를 '테크'로 잘못 예측)	**TN (True Negative)** ('비-테크'를 '비-테크'로 맞힘)

6.3 핵심 평가 지표: 정밀도, 재현율, 특이도

혼동 행렬의 4가지 값(TP, TN, FP, FN)을 조합하여 정확도보다 훨씬 유용한 3가지 핵심 지표를 만듭니다.

Table 3: 불균형 데이터의 핵심 평가 지표

지표	공식	**직관적 의미 (일상 언어 번역)**
정밀도 (Precision)	$\frac{TP}{TP+FP}$	"모델이 '테크'라고 예측한 것들 중, **진짜 '테크'**일 확률" **"이 모델의 예측은 얼마나 믿을만한가?"** (신뢰도)
재현율 (Recall) (민감도, Sensitivity)	$\frac{TP}{TP+FN}$	"**실제 '테크'** 기사들 중, 모델이 **얼마나 많이 찾아냈는가?**" **"이 모델이 놓친 것은 얼마나 적은가?"** (누락도)
특이도 (Specificity)	$\frac{TN}{TN+FP}$	"**실제 '비-테크'** 기사들 중, 모델이 **'비-테크'라고 맞춘** 비율" (재현율의 '비-테크' 버전)

□ 예제:

정밀도 vs. 재현율의 트레이드오프(Trade-off)

우리는 두 지표가 모두 높기를 원하지만, 현실에서는 종종 한쪽이 높아지면 다른 쪽이 낮아지는 **트레이드오프** 관계가 발생합니다.

- **상황 1: 스팸 메일 필터 (정밀도가 중요)** 필터가 **아주 확실한** 스팸(e.g., 99.9% 확신)만 걸러낸다고 가정합시다.
  - **정밀도 (Precision) = 매우 높음**: 필터가 "스팸"이라고 한 것은 100% 스팸입니다. (신뢰도 높음)
  - **재현율 (Recall) = 낮음**: 애매한 스팸(e.g., 80% 확신)은 **놓치고** 받은 편지함으로 통과시킵니다.
  - **결론**: 중요한 메일이 스팸함으로 가는 것(FP)을 막는 것이, 스팸을 몇 개 놓치는 것(FN)보다 중요합니다.
- **상황 2: 암 진단 모델 (재현율이 중요)** 모델이 암일 **가능성이 조금이라도** 있으면 "양성"이라고 예측한다고 가정합시다.
  - **재현율 (Recall) = 매우 높음**: **실제 암 환자(TP)를 놓치는 일(FN)이 거의 없습니다.**
  - **정밀도 (Precision) = 낮음**: "양성" 예측(TP+FP) 중에 **정상인(FP)이 많이 섞여있습니다.**
  - **결론**: 정상인을 암으로 오진(FP)하여 추가 검사를 하는 한이 있더라도, 실제 암 환자를 정상이라고 놓치는(FN) 치명적인 실수는 절대 안 됩니다.

"테크" 뉴스 분류 문제는 정답이 없습니다. "나는 테크 기사를 **하나도 놓치기 싫어!**"라면 **재현율**이 높은 모델을, "내가 추천받은 기사는 **무조건 테크 기사여야 해!**"라면 **정밀도**가 높은 모델을 선택해야 합니다.

7 고전적 분류 모델 상세

7.1 Naive Bayes (나이브 베이즈)

- **한 줄 요약**: 베이즈 정리를 기반으로, "모든 단어는 서로 독립적"이라고 **순진하게(Naive)** 가정하는 분류기.
- **직관적 예시 (스팸 필터)**: 어떤 메일에 'free'와 'Viagra'라는 단어가 동시에 등장했습니다.
  - **현실**: 'free'와 'Viagra'는 스팸 메일에서 **함께** 등장할 확률이 높습니다. (서로 의존적)
  - **Naive Bayes의 가정**: "나는 똑똑하지 않아서(Naive) 그런 관계는 모르겠고, 'free'가 등장할 확률과 'Viagra'가 등장할 확률을 그냥 **곱할래**." 즉, $P(\text{'free' and 'Viagra'}|\text{스팸}) = P(\text{'free'}|\text{스팸}) \times P(\text{'Viagra'}|\text{스팸})$ 이라고 가정합니다.

  이 가정은 명백히 틀렸지만, 놀랍게도 스팸 필터링 같은 많은 문제에서 빠르고 준수한 성능을 보여줍니다.
- **기술적 설명**: 베이즈 정리에 따라, 우리는 $P(C_k|\mathbf{x})$ (특징 $\mathbf{x}$가 주어졌을 때 클래스 C_k 일 확률)를 최대화하는 클래스 C_k를 찾습니다.

$$\hat{C} = \arg\max_{C_k} P(C_k|\mathbf{x}) = \arg\max_{C_k} \frac{P(\mathbf{x}|C_k)P(C_k)}{P(\mathbf{x})}$$

  여기서 $\mathbf{x}$는 (TF-IDF 벡터) $(x_1, x_2, ..., x_n)$입니다. $P(\mathbf{x})$는 모든 클래스에서 동일하므로 무시하고, $P(C_k)$는 단순히 훈련 데이터의 클래스 비율(사전 확률)입니다.
  핵심은 $P(\mathbf{x}|C_k)$인데, **나이브 가정**에 따라 다음과 같이 계산합니다.

$$P(\mathbf{x}|C_k) = P(x_1, ..., x_n|C_k) \approx \prod_{i=1}^{n} P(x_i|C_k)$$

  실제로는 값들이 너무 작아져서 0이 되는 것을 막기 위해(numerical underflow), 로그(log)를 씌워서 곱셈을 덧셈으로 바꿔 계산합니다.

$$\log P(C_k|\mathbf{x}) \propto \log P(C_k) + \sum_{i=1}^{n} \log P(x_i|C_k)$$

- **장점**: 매우 빠르고, 구현이 간단하며, 적은 데이터로도 잘 작동합니다. 훌륭한 "베이스라인" 모델입니다.
- **단점**: "모든 특징이 독립"이라는 가정이 현실과 너무 다르면 성능이 떨어집니다.

7.2 k-Nearest Neighbors (KNN, k-최근접 이웃)

- **한 줄 요약**: 별다른 "학습" 없이, 그냥 데이터를 다 외워버린 다음, 새로운 데이터가 들어오면 **가장 가까운 k개의 이웃**을 보고 **다수결 투표**로 결정하는 모델.
- **직관적 예시 ("동네 투표")** 새로 이사 온 집(새 데이터)이 'A동'인지 'B동'인지 분류해야 합니다.
  1. k값을 정합니다. (e.g., $k = 5$)
  2. 새 집에서 **가장 가까운 집 5개(이웃)**를 찾습니다.
  3. 5개 이웃을 보니, 3채는 'A동', 2채는 'B동'이었습니다.
  4. **다수결의 원칙**에 따라, "새 집은 **'A동'**일 것이다"라고 분류합니다.

KNN은 이 과정이 전부입니다. 별도의 "훈련" 단계가 없고, 데이터 자체를 모델로 사용합니다. (Instance-based Learning)

- **기술적 설명**:
  - **하이퍼파라미터 1: k 값**: k가 너무 작으면(e.g., $k = 1$) 이웃 한 명의 "이상한" 의견(outlier)에 휘둘리기 쉽고 (과적합), k가 너무 크면 너무 먼 동네 의견까지 들어서 분류 경계가 둔해집니다 (과소적합). 보통 3, 5, 7 등 **홀수**를 사용합니다. (동점 투표를 피하기 위해)
  - **하이퍼파라미터 2: 거리 측정 (Distance Metric)**: "가깝다"를 어떻게 정의할지 정해야 합니다.
    - **유클리드 거리 (Euclidean)**: 두 점 사이의 직선 거리. (가장 일반적)
    - **코사인 유사도 (Cosine Similarity)**: 텍스트 분류에서 자주 사용됨. 두 TF-IDF 벡터가 가리키는 **방향**이 얼마나 유사한지 측정. (문서 길이와 상관없이 주제의 유사성을 봄)
- **장점**: 모델이 매우 단순하고 직관적입니다. 훈련이 필요 없습니다.
- **단점**:
  - **데이터 정규화(Normalization) 필수**: 만약 [나이(1 100), 연봉(1만 100만)]으로 이웃을 찾는다면, '나이' 차이(e.g., 5)는 '연봉' 차이(e.g., 50,000)에 비해 완전히 무시됩니다. 모델은 **오직 연봉**만 보게 됩니다. 따라서 모든 특징(feature)의 범위를 [0, 1] 등으로 **스케일링**하는 작업이 **반드시** 필요합니다.
  - **느린 예측**: 예측 시마다 **모든 훈련 데이터**와의 거리를 계산해야 하므로, 데이터가 수백만 개가 되면 예측이 매우 느려집니다.

7.3 Logistic Regression (로지스틱 회귀)

- **한 줄 요약**: 선형 회귀(직선)의 결과를 **시그모이드(Sigmoid) 함수**에 넣어 0 1 사이의 **"확률"** 값으로 변환하는 분류기.
- **직관적 예시 (단 하나의 뉴런)**: 로지스틱 회귀는 딥러닝의 **뉴런 1개**와 거의 동일합니다.
  1. **입력 (Inputs)**: 각 단어의 TF-IDF 점수들 $(x_1, x_2, ...)$.
  2. **가중치 (Weights)**: 각 단어가 '테크' 분류에 얼마나 중요한지 나타내는 가중치 $(w_1, w_2, ...)$. (e.g., 'apple'의 가중치는 높고, 'sport'의 가중치는 음수일 것)
  3. **선형 결합 (z)**: 모든 (점수 × 가중치)를 더합니다. $\rightarrow z = w_1x_1 + w_2x_2 + ... + b$ (z 값은 $-\infty$ $+\infty$ 사이의 어떤 값이 됩니다.)
  4. **활성화 함수 (Sigmoid)**: z 값을 시그모이드 함수에 넣어 0 1 사이의 값으로 "찌그러뜨립니다". $\rightarrow \sigma(z) = \frac{1}{1+e^{-z}}$ (e.g., $z = 10 \rightarrow \sigma(z) \approx 1.0$, $z = -5 \rightarrow \sigma(z) \approx 0.0$, $z = 0 \rightarrow \sigma(z) = 0.5$)

  이 0 1 사이의 최종 값이 "테크" 기사일 확률(e.g., 0.9)이 되며, 보통 0.5를 기준으로 '테크'(>0.5) / '비-테크'(<0.5)를 결정합니다.
- **기술적 설명**:
  - **선형 결정 경계 (Linear Boundary)**: 로지스틱 회귀는 기본적으로 **직선(또는 초평면)**으로만 데이터를 나눌 수 있습니다.
  - **비선형 문제 해결법**: 만약 데이터가 원형으로 분포(e.g., 중앙은 'A', 바깥은 'B')한다면 직선으로 나눌 수 없습니다. 이때는 **특성 공학(Feature Engineering)**을 통해 해결합니다. 기존 입력 x_1, x_2에 더해 x_1^2, x_2^2 같은 **새로운 특성**을 우리가 **직접** 만들어서 모델에 넣어주면, 모델이 $w_3(x_1^2) + w_4(x_2^2)$ 같은 항을 학습하여 **원형의 경계**를 만들 수 있게 됩니다.
- **장점**: 출력이 0 1 사이의 확률이라 해석하기 좋습니다. 딥러닝의 기초 개념을 이해하는 데 도움이 됩니다.

- **단점**: 기본적으로 선형 분류기이므로, 비선형 경계를 가지는 복잡한 문제는 잘 풀지 못합니다. (SVM의 커널 트릭과 대비됨)

7.4 Support Vector Machines (SVM, 서포트 벡터 머신)

- **한 줄 요약**: 2012년 딥러닝이 부상하기 전까지 "왕좌"에 있던 모델. 두 클래스 사이의 **간격(Margin)을 최대화**하는 결정 경계(선)를 찾는 모델.
- **직관적 예시 ("가장 넓은 도로 찾기")** 서로 다른 두 마을('A' 마을, 'B' 마을) 사이에 국경선을 긋는다고 생각해봅시다.
  - **다른 모델**: 그냥 'A'와 'B'를 나누는 **선(Line)** 하나만 그으려고 합니다.
  - **SVM**: 선 하나가 아니라, 두 마을 사이에 **가장 넓은 중립 도로(Margin)**를 낸다고 생각합니다.

  이때 도로의 **양쪽 경계선**에 정확히 **닿는** 집들(데이터 포인트)이 생기는데, 이 집들이 바로 도로의 위치를 결정하는 가장 중요한 집들, 즉 **서포트 벡터(Support Vectors)**입니다. SVM은 이 "도로 폭(Margin)"을 **최대화**하는 선을 찾습니다.
- **기술적 설명 (단계별 진화)**

  1. 하드 마진 (Hard Margin)
  - **개념**: 단 하나의 데이터도 도로(마진) 안으로 들어오는 것을 허용하지 않는 "완벽한" 도로.
  - **문제 1 (과적합)**: 도로 폭이 오직 양쪽의 가장 가까운 집 2 3개(서포트 벡터)에만 의존합니다. 이 집들이 약간의 노이즈(Outlier)라면, 도로 전체가 심하게 휘어집니다. (과적합)
  - **문제 2 (불가)**: 두 마을의 집들이 약간 섞여있다면(Linearly non-separable), 하드 마진 도로는 아예 만들 수조차 없습니다.

  2. 소프트 마진 (Soft Margin) - 하이퍼파라미터 C
  - **개념**: 현실적인 타협. "약간의 집(데이터)이 도로 안(마진 침범)에 있거나, 심지어 **국경 너머(잘못 분류)**에 있는 것을 **허용**하자. 대신 **벌금(C)**을 내자."
  - **하이퍼파라미터 C (벌금의 세기)**:
  - **C가 크다 (High C)**: **벌금이 아주 비싸다.** → SVM은 벌금을 내기 싫어서 마진을 침범하지 않으려고 **좁은** 도로를 만듭니다. (하드 마진과 유사해짐, 과적합 위험)
  - **C가 작다 (Low C)**: **벌금이 싸다.** → SVM은 "벌금 좀 내지 뭐" 하면서 마진을 침범하는 데이터들을 너그럽게 봐주고, 대신 **도로 폭(마진)을 넓게** 확보합니다. (일반화 성능 향상)
  - **효과**: 소프트 마진 덕분에 데이터가 섞여있어도 분류가 가능해지고, 과적합이 줄어듭니다.

  3. 커널 트릭 (The Kernel Trick) - 'kernel='rbf'' / ''poly''

  □ **예제:**

  문제: 데이터가 ['A' 마을이 'B' 마을을 **원형으로 둘러싼**] 형태입니다. 이건 2D 평면에서는 **절대 직선(도로)으로** 나눌 수 없습니다.
  잘못된 직관: "SVM이 원형 경계선을 그린다."
  올바른 직관 (커널 트릭):
  1. 2D 지도를 3D로 바꿉니다. $z = x^2 + y^2$ (중심에서 먼 만큼 높이 솟아오름) 라는 3번째 차원을 **추가**합니다.
  2. 2D 지도에서는 'A'가 'B'를 둘러쌌지만, 3D 공간에서는 바깥쪽 'B' 마을이 **더 높이 솟아오르고 (높은 z)**, 안쪽 'A' 마을은 **아래 (낮은 z)**에 있게 됩니다.
  3. 이제 이 3D 공간에서는 **단순한 평면(Hyperplane)** 하나로 'A'와 'B'를 완벽하게 분리(e.g., $z = 0.5$ 평면)할 수 있게 됩니다!

  커널(Kernel)이란, $z = x^2 + y^2$ 같은 새로운 차원을 **실제로 계산하지 않고도** (계산량이 너무 많아짐), 마치 그 고차원에서 계산한 것과 **동일한 결과**를 2D에서 **값싸게** 얻어내는 수학적 "트릭"

입니다.
- `kernel='linear'`: 커널 트릭 안 씀. (데이터가 이미 고차원이거나 선형 분리 가능할 때 좋음)
- `kernel='rbf'`: (Radial Basis Function) 데이터를 무한 차원으로 매핑. (가장 범용적이고 강력함)
- `kernel='poly'`: 다항식(x^2, x_1x_2 등) 차원을 추가.

- **장점**: **매우 강력합니다.** 특히 텍스트 데이터(TF-IDF)처럼 **매우 고차원**(수만 차원)인 공간에서는, 데이터가 선형으로 분리 가능할 확률이 높기 때문에 'kernel='linear''인 SVM이 압도적으로 빠르고 좋은 성능을 보일 때가 많습니다. 비선형 문제도 커널 트릭으로 잘 해결합니다.
- **단점**: 모델이 블랙박스에 가깝고, 데이터가 아주 많아지면 훈련 속도가 느려집니다.

7.5 Random Forests (랜덤 포레스트)

- **한 줄 요약**: 과적합되기 쉬운 **결정 트리(Decision Tree)** 모델 **수백 개**를 만들되, 각각을 **일부러 조금씩 다르게** 만들어서(Random), 그 결과들을 **평균(회귀) 또는 투표(분류)**하는 앙상블(Ensemble) 모델.
- **직관적 예시 ("현명한 군중")**
  - **결정 트리 1개**: "한 명의 **천재 전문가**"와 같습니다. 이 전문가는 아주 복잡한 규칙(e.g., "연봉이 5만 이상이고, 나이가 30 미만이며, 거주지가 서울이 아니면...")을 만들어 데이터를 완벽하게 외워버립니다. **문제**: 이 전문가는 훈련 데이터는 100% 맞히지만, 자신의 지식(규칙)에 너무 빠져있어서(과적합) 새로운 데이터를 만나면 엉뚱한 예측을 할 수 있습니다.
  - **랜덤 포레스트**: "수백 명의 **다양한 전문가**로 이루어진 **군중**"과 같습니다. 한 명의 천재보다 **조금 덜 똑똑한 다수의 군중**이 더 현명한 결정을 내린다는 원리입니다. **어떻게 "다양한" 전문가를 만드나요? (이것이 "Random"의 핵심)**
    1. **부트스트랩 (Bootstrapping, 다른 데이터)**: 1000개의 데이터가 있다면, **복원 추출** (샘플 뽑고 다시 집어넣기)로 1000개를 뽑습니다. (어떤 데이터는 2 3번 뽑히고, 어떤 데이터는 아예 안 뽑힘) 이 "조금씩 다른 훈련 세트"를 500개 만들어서 500명의 전문가(트리)에게 각각 나눠줍니다. → 모든 전문가가 **서로 다른 데이터**로 학습합니다.
    2. **특성 랜덤성 (Feature Randomness, 다른 관점)**: 각 전문가(트리)가 결정을 내릴 때(split), **모든 정보(특성)를 보지 못하게 합니다.** (e.g., 전체 특성이 20개라면, "당신은 5개 특성 중에서만 골라서 결정하세요"라고 제한함) → 모든 전문가가 **서로 다른 관점**으로 문제를 보게 됩니다.

    최종 결정: 500명의 전문가가 **다수결 투표**를 합니다. 개별 전문가는 과적합되었을지 몰라도, 그들의 실수가 **서로 다르기 때문에** 투표 과정에서 **상쇄**됩니다.
- **기술적 설명**: 앙상블 기법 중 **배깅(Bagging)** = **B**ootstrap **Agg**regat**ing**에 특성 랜덤성을 추가한 모델입니다. 결정 트리는 데이터를 나눌 때 **불순도(Impurity)**를 가장 낮추는 방향으로 나눕니다. (Gini Index 또는 Entropy 사용) 랜덤 포레스트는 이 과정을 수백 번 반복하여 그 결과를 취합합니다.
- **장점**: **매우 강력하고 안정적입니다.** 데이터 스케일링(정규화)이 필요 없습니다. 과적합에 매우 강건하여, 하이퍼파라미터 튜닝에 크게 신경 쓰지 않아도 (e.g., 기본값 사용) 거의 항상 준수한 성능을 냅니다. ("대충 써도 잘 맞는" 모델의 대명사)
- **단점**: SVM보다 훈련 속도가 느릴 수 있고, 모델이 완전히 블랙박스라 해석이 어렵습니다.

8 BBC 뉴스 분류 실습 사례 연구

강의에서 다룬 BBC 뉴스 분류 실습을 통해, 이 모델들이 불균형 데이터에서 어떻게 작동하는지 비교 분석합니다.

- **문제**: 2,225개의 BBC 뉴스 기사 분류
- **데이터**: 5개 카테고리 (스포츠, 비즈니스, 정치, 테크, 연예)
- **실습 목표**: "테크" 기사(401개)와 "비-테크" 기사(1824개)로 이진 분류 (약 1:4.5 불균형)
- **검증 방법**: 5-겹 교차 검증 (K-Fold CV)을 사용한 `GridSearchCV`
- **평가 지표**: 정확도(Accuracy), **재현율(Recall/Sensitivity)**, **특이도(Specificity)**

Table 4: *BBC* 뉴스 분류 모델별 성능 비교 (*5-Fold CV*)

모델 (Model)	최적 하이퍼파라미터	정확도(Acc.)	재현율(Recall)	특이도(Spec.)	**분석 및 평가**
Naive Bayes	(기본값)	89.6%	43% (낮음)	**100%**	**함정에 빠짐**. 정확도는 높아 보이나, 재현율이 43%라는 것은 **테크 기사의 절반 이상을 놓쳤다**는 의미. 특이도가 100%인 것은, 모델이 그냥 "비-테크"로만 예측했음을 시사.
KNN	`k=5` (기본값)	97%	94%	98%	매우 우수한 성능. 재현율과 특이도 모두 높음.
KNN (Optimized)	`k=19`	98%	94%	99%	`k=19` (더 넓은 이웃)가 기본값보다 약간 더 안정적인 성능을 보임.
Logistic Reg.	(기본값)	90.7%	69% (낮음)	**100%**	Naive Bayes와 유사. 재현율은 조금 낮지만 여전히 다수 클래스인 "비-테크"로 예측하는 경향이 매우 강함.
SVM (Default)	'kernel='rbf', 'C=1'	92.5%	70%	**100%**	Logistic Regression과 거의 동일한 문제 발생.
SVM (Optimized)	`C=10, kernel='linear'`	**98.8%**	**97%** **(Best)**	**99%**	**압도적인 1위.** 높은 `C` 값과 `linear` 커널이 이 고차원 TF-IDF 데이터에 가장 적합했음. 테크 기사를 **거의 놓치지 않으면서(재현율 97%)** 비-테크 기사도 완벽하게(특이도 99%) 분리해냄.
Random Forest	(기본값)	98.4%	89% (Slightly low)	**100%**	매우 좋은 성능. 하지만 SVM(Optimized)에 비해 재현율이 다소 낮아(89%), 일부 테크 기사를 놓침.

□ 핵심 요약

실습 결론

- **평가 지표의 중요성**: 단순 정확도만 봤다면 90%대의 Naive Bayes나 Logistic Regression도 좋아 보였겠지만, **재현율(Recall)**을 확인해보니 이 모델들은 "테크" 기사를 거의 찾지 못하는 쓸모없는 모델이었음을 알 수 있었습니다.
- **하이퍼파라미터 튜닝의 중요성**: SVM은 **기본값(rbf, C=1)**으로는 엉뚱한 결과(재현율 70%)를 냈지만, `GridSearchCV`를 통해 **최적의 파라미터(linear, C=10)**를 찾자 모든 모델 중 가장 완벽한 성능(재현율 97%)을 보여주었습니다.
- **고차원 텍스트와 Linear SVM**: TF-IDF로 변환된 텍스트 데이터는 수만 차원의 **초고차원(High-dimensional)** 데이터입니다. 이런 초고차원 공간에서는 데이터가 **선형(linear)으로 분리 가능**할 확률이 매우 높습니다. 이것이 복잡한 'rbf' 커널보다 단순한 **'linear' 커널 SVM**이 이 문제에서 최고의 성능을 낸 이유입니다.

9 학습 체크리스트 및 FAQ

9.1 학습 체크리스트

모델을 만들기 전, 아래 항목들을 점검해보세요.

텍스트 데이터를 숫자로 변환했는가? (e.g., `TfidfVectorizer`)

훈련 데이터와 테스트 데이터를 분리했는가?

데이터셋이 크지 않다면 (e.g., 10만 건 이하), K-Fold CV 사용을 고려했는가?

K-Fold CV 사용 시, `Pipeline`을 구성하여 데이터 누수(Leakage)를 방지했는가?

분류하려는 클래스 간의 비율이 불균형한가? (e.g., 9:1, 8:2)

(불균형하다면) '정확도(Accuracy)' 대신 '정밀도(Precision)'와 '재현율(Recall)'을 확인했는가?

KNN, SVM, Random Forest 같은 모델의 하이퍼파라미터를 튜닝했는가? (e.g., `GridSearchCV`)

9.2 FAQ (자주 묻는 질문)

Q: Naive Bayes와 Logistic Regression이 왜 이 문제에서 실패했나요?

A: 데이터가 "비-테크"(1824개) 쪽으로 심하게 **불균형**했기 때문입니다. 이 모델들은 손실(loss)을 최소화하는 과정에서, "일단 다수 클래스인 '비-테크'라고 예측하면 82%는 맞힌다"는 **쉬운 길(local minima)**에 빠지기 쉽습니다. 특이도(Specificity)가 100%라는 것은, 이 모델들이 "테크" 기사를 거의 무시하고 "비-테크"로만 예측했음을 보여줍니다.

Q: KNN에서 `k=19`가 `k=5`보다 좋은 이유는 무엇인가요?

A: `k=5`는 너무 **지역적(local)**인 정보만 봅니다. 만약 내 주변 5명만 보고 투표한다면, 그 5명이 우연히 이상한 의견을 가졌을 때(노이즈) 나의 예측도 흔들리기 쉽습니다. `k=19`는 더 **넓은(global)** 동네의 의견을 반영합니다. (e.g., 19명 중 2 3명이 이상해도 다수결에 큰 영향 없음) 이 데이터에서는 `k=19` 정도가 노이즈의 영향을 받지 않고 데이터의 전반적인 패턴을 더 잘 반영하는 "최적의 동네 크기"였던 것입니다.

Q: SVM에서 왜 `linear` 커널이 `rbf` (비선형) 커널보다 좋았나요?

A: **차원의 저주(Curse of Dimensionality)의 역설**입니다. TF-IDF 벡터는 **수만 차원**의 초고차원 공간에 존재합니다. 저차원(2D, 3D)에서는 데이터가 복잡하게 얽혀있어 선으로 나누기 어렵지만, **초고차원 공간**으로 가면 데이터 포인트 사이의 거리가 매우 멀어져서, 마치 텅 빈 우주에 점들이 흩뿌려진 것처럼 됩니다. 이런 초고차원 공간에서는 **단순한 선(초평면) 하나**만으로도 두 클래스를 분리할 수 있게 될 확률이 **매우 높아집니다.**
따라서 굳이 `rbf` 같은 복잡한 비선형 커널로 경계선을 꼬아줄 필요 없이, 가장 단순하고 빠른 **`linear` 커널**이 오히려 더 좋은 성능을 낸 것입니다.

Q: 최종 모델은 5개 모델의 평균인가요?

A: **아닙니다.** K-Fold CV는 **평가 및 하이퍼파라미터 탐색**을 위한 과정입니다. 일단 `GridSearchCV`를 통해 "SVM + C=10 + kernel='linear' 조합이 1등이다"라는 사실을 알아냈다면, K-Fold CV

는 거기서 임무가 끝납니다.
우리가 고객에게 배포할 **최종 모델**은, 이 1등 조합(SVM, C=10, `linear`)을 설정으로 하여 **전체 훈련 데이터(Train+Validation)를 모두 사용해 단 한 번** 훈련시킨, **단 하나의 모델**입니다.

10 빠른 훑어보기 (1-Page Quick Review)

□ 핵심 요약

텍스트 분류의 3단계 핵심 프로세스

1. **Vectorize (벡터화)**: 텍스트를 숫자로 변환 (`TfidfVectorizer`)
2. **Validate (검증)**: 모델의 "진짜 실력"을 측정. 데이터가 적으면 **K-Fold CV** 사용. 이때 **Pipeline**으로 데이터 누수 방지는 필수.
3. **Evaluate (평가)**: 모델의 성적표 확인.

불균형 데이터 평가의 제1원칙

클래스 비율이 9:1처럼 불균형하면, **절대 '정확도(Accuracy)'를 믿지 마세요.** "멍청한" 모델도 90%의 정확도를 달성할 수 있습니다.

반드시 **혼동 행렬(Confusion Matrix)**을 열고, 목적에 따라 **정밀도(Precision)**와 **재현율(Recall)**을 확인해야 합니다.

□ 고전적 모델 치트 시트 (Model Cheat Sheet)

모델	핵심 아이디어	언제 사용하는가?
Naive Bayes	"모든 단어는 독립"이라는 순진한 가정. ($P(A \cap B) = P(A) \times P(B)$)	빠르고 간단한 **베이스라인**이 필요할 때. (e.g., 스팸 필터)
KNN	"가장 가까운 k명의 이웃에게 물어보고 **다수결**로 결정."	모델이 직관적이어야 할 때. (e.g., 추천 시스템) **정규화 필수!**
Logistic Reg.	뉴런 1개. 선형 결합(z)을 시그모이드 함수로 압축해 **0 1 확률** 출력.	분류 결과가 "확률"로 나와야 할 때. 선형 경계가 잘 먹힐 때.
SVM	두 클래스 사이의 **"도로 폭(Margin)"을 최대화**하는 선을 찾음. + **커널 트릭** (e.g., `'rbf'`)	**텍스트(TF-IDF) 분류** 같이 고차원 데이터에 매우 강력함. 비선형 데이터도 차원 트릭으로 해결 가능. (범용성 높음)
Random Forest	과적합된 "전문가(트리)" 수백 명을 **무작위**로 만들어 **투표**시킴.	**과적합을 피하는** 가장 안정적인 방법. (성능이 웬만해선 잘 나옴)

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.

December 10, 2025

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 11

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 11의 핵심 개념 학습

□ 핵심 요약

이 문서는 자연어 처리(NLP)의 고급 모델들을 다룹니다. 문장의 순서와 구조를 이해하는 **순차 레이블링 모델**(HMM, CRF, BiLSTM-CRF)부터, 새로운 데이터(텍스트나 이미지)를 생성하는 **생성 모델**(VAE, GAN)까지의 핵심 원리를 다룹니다.

각 모델이 **왜 등장했는지**(이전 모델의 한계), **어떻게 작동하는지**(핵심 아이디어), 그리고 **어떤 단점이 있는지**(다음 모델의 등장 배경)를 중심으로 설명합니다. 이 문서 하나만으로 각 모델의 개념을 잡고 서로 비교할 수 있도록 구성되었습니다.

Contents

1 용어 정리

본격적인 학습에 앞서, 이 문서에서 다룰 주요 용어들을 정리합니다.

Table 1: 주요 *NLP* 모델 및 개념 용어

용어	쉬운 설명	원어	비고 (핵심 키워드)
마코프 속성	"미래는 오직 현재에만 의존한다." (과거는 불필요)	Markov Property	HMM의 기본 가정
HMM	'숨겨진 상태'가 '관찰된 결과'를 만든다고 보는 모델	Hidden Markov Model	생성 모델, POS 태깅
CRF	'전체 관찰 결과'를 보고 '가장 확률 높은 상태'를 맞히는 모델	Conditional Random Fields	판별 모델, NER, HMM의 한계 극복
특징 함수	CRF가 특정 패턴(예: 대문자, 이전 단어)을 감지하는 규칙	Feature Function	CRF의 핵심. (수동 정의 필요)
BiLSTM	문장을 양방향으로 읽어 문맥을 파악하는 똑똑한 RNN	Bidirectional LSTM	자동 특징 추출기
오토인코더	데이터를 압축(인코더)했다가 복원(디코더)하는 모델	Autoencoder (AE)	차원 축소, 특징 학습
VAE	잠재 공간을 '확률 분포'로 만들어 부드럽게 연결한 AE	Variational AE	생성 모델, 잠재 공간 구조화
잠재 손실	VAE가 잠재 공간을 구조화하도록 강제하는 패널티	Latent Loss (KL Div.)	μ는 0으로, σ는 1로 유도
GAN	위조범(생성자)과 경찰(판별자)이 경쟁하며 학습하는 모델	Generative Adversarial Network	생성 모델, 고품질 이미지 생성

2 핵심 개념 1: Hidden Markov Models (HMM)

HMM(은닉 마코프 모델)은 순차적인 데이터(예: 문장)를 이해하기 위한 고전적이면서도 중요한 통계 모델입니다.

2.1 시작하기: 마코프 체인 (Markov Chains)

HMM을 이해하려면 먼저 마코프 체인을 알아야 합니다.

□ 핵심 정보

마코프 체인(Markov Chain)은 여러 '상태'(State)가 존재하고, 한 상태에서 다음 상태로 이동할 확률(전이 확률)이 정해져 있는 모델입니다.
핵심 가정: 마코프 속성 (Markov Property)

- "미래의 상태는 오직 **현재 상태**에만 의존한다."
- 즉, 내가 '상태 2'에 도달하기까지 '상태 0 → 상태 1'을 거쳤든, '상태 3 → 상태 1'을 거쳤든 상관없이, '상태 1'에 있다는 **현재 사실**만이 다음 상태(예: '상태 2')로 갈 확률에 영향을 줍니다.
- (비유) 주사위 굴리기: 이전에 1이 10번 연속 나왔다고 해서, 다음번에 1이 나올 확률(1/6)이 변하지 않는 것과 비슷합니다.

□ 예제:

예시: 날씨 마코프 체인 세 가지 날씨 상태가 있다고 가정합니다: (1) 맑음, (2) 흐림, (3) 비

- 오늘 '맑음'일 때, 내일 '맑음'일 확률 ($P(\text{맑음}|\text{맑음}) = 0.7$)
- 오늘 '맑음'일 때, 내일 '흐림'일 확률 ($P(\text{흐림}|\text{맑음}) = 0.2$)
- 오늘 '맑음'일 때, 내일 '비'일 확률 ($P(\text{비}|\text{맑음}) = 0.1$)
- (이 확률들의 합은 1이 되어야 합니다: $0.7 + 0.2 + 0.1 = 1.0$)

이처럼 '흐림'일 때와 '비'일 때의 다음 날 날씨 확률도 모두 정의해 놓은 것이 마코프 체인입니다.

2.2 HMM: 숨겨진 상태와 관찰된 결과

HMM은 마코프 체인에서 한 단계 더 나아갑니다. 우리가 '상태'를 직접 볼 수 없고, '상태'가 만들어낸 '결과물'만 볼 수 있다고 가정합니다.

□ 핵심 정보

HMM(Hidden Markov Model)은 두 가지 층위로 구성됩니다.

1. **숨겨진 상태 (Hidden States, Y)**:
   - 우리는 직접 볼 수 없습니다. (예: 실제 날씨, 단어의 품사)
   - 이 숨겨진 상태들은 마코프 속성을 따르며 서로 이동합니다. (예: 명사 다음에는 동사가 올 확률이 높다)
   - 이 이동 확률을 **전이 확률 (Transition Probability)**이라고 부릅니다.
2. **관찰된 결과 (Observations, X)**:
   - 숨겨진 상태가 만들어낸 결과물이며, 우리가 실제로 보는 데이터입니다.

- (예: 아이스크림 판매량, 실제 단어 'study')
- 특정 숨겨진 상태가 특정 관찰 결과를 만들어낼 확률을 **방출 확률 (Emission Probability)**이라고 부릅니다.

이 외에, 문장이 어떤 상태에서 시작할지 정하는 **초기 상태 확률**이 있습니다.

□ 예제:

예시: 품사 판별 (Part-of-Speech Tagging) 우리의 목표는 "I study"라는 문장(관찰 X)을 보고, "대명사, 동사"라는 품사(숨겨진 상태 Y)를 맞히는 것입니다.

- **관찰 (X)**: "I", "study" (우리가 보는 단어)
- **숨겨진 상태 (Y)**: "대명사(PRP)", "동사(VBP)" (우리가 맞혀야 하는 품사)
- **초기 확률**: 문장은 '대명사'로 시작할 확률이 높다. ($P(Y_1 = PRP)$)
- **전이 확률**: '대명사' 상태(Y_t) 다음에는 '동사' 상태(Y_t+1)가 올 확률이 높다. (P(VBP|PRP))
- **방출 확률**: '대명사' 상태(Y)는 "I"라는 단어(X)를 방출(생성)할 확률이 높다. (P("I"|PRP)) / '동사' 상태(Y)는 "study"라는 단어(X)를 방출할 확률이 높다. (P("study"|VBP))

HMM은 이 확률들을 조합하여 "I study"라는 관찰이 주어졌을 때, 가장 그럴듯한(확률 높은) 숨겨진 상태의 연속(즉, 품사 태그)이 "대명사 → 동사"임을 계산해냅니다.

2.3 HMM의 한계

HMM은 강력하지만 치명적인 한계를 가집니다.

주의사항

한계: 너무 단순한 의존성 가정

- **마코프 속성의 한계**: HMM은 $t+1$ 시점의 상태(Y_t+1)가 **오직 t 시점의 상태(Y_t)에만** 의존한다고 가정합니다.
- (예시) "I study"에서 "study"의 품사는 "I"(대명사)에만 영향을 받습니다.
- **관찰 독립성의 한계**: HMM은 t 시점의 관찰(X_t)이 **오직 t 시점의 상태(Y_t)에만** 의존한다고 가정합니다.
- (예시) "study"라는 단어는 "동사"라는 현재 품사에만 영향을 받습니다.

문제점: 실제 언어는 그렇지 않습니다! "study"가 동사인지 명사인지 판단하려면 "I"라는 단어뿐만 아니라 "I **will** study..."나 "A **recent** study..."처럼 문장 **전체**의 다른 단어들(X)을 모두 참고해야 합니다.

HMM은 현재 상태(Y_t) 외에는 **그 어떤 정보(다른 시점의 Y나 X)도 보지 못하는** 근시안적인 모델입니다.

3 핵심 개념 2: Conditional Random Fields (CRFs)

CRF(조건부 랜덤 필드)는 HMM의 '근시안적인' 한계를 극복하기 위해 등장한 강력한 순차 레이블링 모델입니다.

3.1 HMM의 한계를 극복하다

HMM이 "현재 상태(Y_t)는 오직 이전 상태(Y_t-1)에만 의존한다"고 가정한 반면, CRF는 이 가정을 완전히 버립니다.

□ 핵심 정보

CRF(Conditional Random Field)의 핵심 아이디어:
"레이블(Y)을 맞힐 때, HMM처럼 쪼개서 보지 말고, **관찰(X) 시퀀스 전체를 한꺼번에** 조건으로 사용하자!"

- **HMM (생성 모델)**: $P(X, Y)$를 모델링. (상태가 관찰을 '생성'한다고 봄)
  - "어떤 품사(Y)가 어떤 단어(X)를 생성할까?" (날씨가 아이스크림 판매량을 생성)
  - $P(X, Y) = P(Y) \times P(X|Y)$
- **CRF (판별 모델)**: $P(Y|X)$를 직접 모델링. (관찰을 보고 상태를 '판별'함)
  - "이 단어들(X)이 주어졌을 때, 가장 적절한 품사(Y)는 무엇일까?"
  - (비유) 로지스틱 회귀가 선형 회귀보다 분류에 더 직접적이듯, CRF는 HMM보다 순차 레이블링에 더 직접적입니다. (CRF는 로지스틱 회귀의 시퀀스 버전이라 불림)

이 '판별' 접근 방식 덕분에, CRF는 HMM이 할 수 없었던 **문장 전체의 다양한 특징**을 자유롭게 활용할 수 있습니다.

3.2 CRF의 핵심: 특징 함수 (Feature Functions)

CRF가 문장 전체의 특징을 활용하는 방법은 '특징 함수'를 사용하는 것입니다.

□ 핵심 정보

특징 함수 (f_k)는 우리가 모델에게 알려주는 "규칙" 또는 "패턴"입니다. 이 함수는 (이전 레이블 Y_{t-1}, 현재 레이블 Y_t, 관찰 시퀀스 X, 현재 시점 t)를 입력받아 0 또는 1을 반환합니다.
가중치 (λ_k)는 각 특징 함수(f_k)가 얼마나 중요한지 나타내는 점수입니다. (이 값은 모델이 학습을 통해 스스로 찾습니다.)
CRF는 이 (특징 함수 × 가중치)의 합을 기반으로 가장 점수가 높은 레이블 시퀀스를 선택합니다.

□ 예제:

예시: 이름 인식 (Named Entity Recognition, NER) 문장 "Mr. Smith..."에서 "Smith"가 사람 이름(B-PERSON)임을 맞히고 싶습니다.

1. 우리가 직접 '특징 함수'(f_k)를 설계합니다: (수동 공학)

- $f_1 = 1$ **if** (현재 단어(X_t)가 "Smith" AND 현재 레이블(Y_t)이 "B-PERSON") **else** 0
- $f_2 = 1$ **if** (현재 단어(X_t)가 대문자로 시작 AND 현재 레이블(Y_t)이 "B-PERSON") **else** 0
- $f_3 = 1$ **if** (**이전 단어**(X_{t-1})**가 "Mr."** AND 현재 레이블(Y_t)이 "B-PERSON") **else** 0

- $f_4 = 1$ **if** (이전 레이블(Y_{t-1})이 "O" AND 현재 레이블(Y_t)이 "B-PERSON") **else** 0
- $f_5 = 1$ **if** (**다음 단어**(X_{t+1})**가 쉼표**(,) AND 현재 레이블(Y_t)이 "B-PERSON") **else** 0

2. 모델이 λ_k (가중치)를 학습합니다:

- 학습 데이터에서 f_2, f_3, f_4 같은 패턴이 이름 인식에 매우 유용하다는 것을 발견하면,
- 모델은 $\lambda_2, \lambda_3, \lambda_4$에 높은 양수 값을 할당합니다.

3. 예측: "Mr. Smith"가 입력되면, f_2, f_3, f_4가 모두 활성화(1)되면서, "Smith"의 레이블로 "B-PERSON"을 선택할 때 전체 점수가 가장 높아지게 됩니다. HMM과 달리 **과거("Mr.")와 미래(",")** - **만약 있다면)**의 관찰(X)을 모두 활용할 수 있습니다.

3.3 CRF의 장점과 단점

주의사항

장점:

- **유연한 특징 활용**: HMM과 달리 문장 전체의 어떤 특징(예: 현재 단어, 이전/다음 단어, 접두사, 접미사, 대소문자 여부 등)이든 자유롭게 가져와 사용할 수 있습니다.
- **판별 모델**: $P(Y|X)$를 직접 모델링하여 순차 레이블링 작업에 더 적합합니다.

치명적인 단점:

- **수동 특징 공학 (Intensive Feature Engineering)**:
- 위 예시처럼, 어떤 특징(f_k)이 유용한지는 **사람이 직접** 생각해서 코드로 구현해야 합니다.
- 이는 엄청난 시간과 노력이 필요하며, 도메인 전문가의 지식이 요구됩니다.
- 만약 우리가 f_3 (이전 단어가 "Mr.") 같은 중요한 특징을 빠뜨리면, 모델의 성능은 급격히 저하됩니다.

4 핵심 개념 3: BiLSTM-CRF (자동 특징 공학)

CRF는 강력했지만 '수동 특징 공학'이라는 거대한 벽에 부딪혔습니다. 이 문제를 해결하기 위해 딥러닝, 즉 BiLSTM이 CRF와 결합되었습니다.

4.1 CRF의 단점을 해결하다: BiLSTM의 등장

CRF의 문제는 '좋은 특징'을 사람이 만들어야 한다는 것이었습니다. 만약 '좋은 특징'을 기계가 알아서 문맥을 보고 뽑아주면 어떨까요?

□ 핵심 정보

BiLSTM-CRF 아키텍처는 두 개의 강력한 모델이 각자의 장점을 살려 결합한 형태입니다.

1. **BiLSTM (Bidirectional LSTM) 층: "자동 특징 추출기"**
   - 입력: 단어들의 시퀀스 (보통 임베딩 벡터로 변환됨)
   - 역할: CRF에 필요했던 '특징 함수'(f_k)를 **자동으로 학습**합니다.
   - BiLSTM은 문장을 정방향(왼쪽 → 오른쪽)과 역방향(오른쪽 → 왼쪽)으로 동시에 읽습니다.
   - (예시) "Smith"라는 단어를 처리할 때, 정방향 LSTM은 "Mr."라는 과거 문맥을, 역방향 LSTM은 "lives in..."이라는 미래 문맥을 모두 고려합니다.
   - 출력: 각 단어 위치(t)에서, 과거와 미래 문맥이 모두 풍부하게 반영된 **"특징 벡터"** ($\mathbf{H}_t$) . CRF ′ ′ .
2. **CRF 층: "최종 레이블 결정자"**
   - 입력: BiLSTM이 뽑아준 고품질 특징 벡터들 (H_1, H_2, ...)
   - 역할: 이 특징들을 입력받아, **레이블 시퀀스 간의 의존성**을 학습하고 최종 레이블을 결정합니다.
   - (예시) "Smith"가 B-PERSON(이름 시작)일 확률이 높다는 특징 벡터를 받아도, CRF는 "I-PER(이름 중간) 뒤에는 B-LOC(장소 시작)이 올 수 없다"와 같은 **레이블 간의 규칙**을 학습하여, "B-PERSON → I-PERSON"처럼 문법적으로 올바른 레이블 시퀀스를 출력하도록 보장합니다.

왜 BiLSTM 위에 Softmax만 쓰지 않고 굳이 CRF를 붙이나요? BiLSTM의 각 시점 출력(H_t)에 Softmax를 적용하여 바로 레이블을 예측할 수도 있습니다 (이것을 '독립적인 분류'라고 합니다).

문제점: Softmax는 각 단어의 레이블을 **독립적으로** 예측합니다.

- (예시) "Mr. John Smith"를 예측할 때, "Mr."(B-PER), "John"(I-PER)까지는 잘 맞히다가, "Smith"에서 실수로 B-LOC(장소 시작)를 예측할 수 있습니다.
- 그 결과 "B-PER → I-PER → **B-LOC**"라는, 문법적으로 말이 안 되는(I-PER 다음에는 I-PER나 O가 와야 함) 레이블 시퀀스가 나올 수 있습니다.

CRF의 역할: CRF 층은 $P(Y|X)$를 '전역적(Globally)'으로 최적화합니다.

- 즉, BiLSTM이 "Smith"를 B-LOC로 예측하려는 성향(Emission 점수)을 보여도, CRF 층이 학습한 "I-PER → B-LOC"라는 **전이(Transition) 점수**가 매우 낮다면,
- CRF는 이 경로를 패널티를 주어 선택하지 않고, 대신 "I-PER → I-PER"라는 더 그럴듯한(점수가 높은) 전체 시퀀스를 선택합니다.

결론: BiLSTM이 '문맥을 읽어 특징을 뽑는' 두뇌라면, CRF는 '레이블 간의 문법 규칙을 적용하는' 문법 교정기 역할을 합니다.

4.2 장점과 단점

□ 핵심 정보

장점:

- **자동 특징 공학**: BiLSTM이 문맥을 읽어 CRF가 필요로 하는 특징을 자동으로 학습합니다. (수동 공학 불필요)
- **높은 정확도**: 문맥(BiLSTM)과 레이블 의존성(CRF)을 모두 고려하여, NER, POS 태깅 등에서 SOTA(최고 수준) 성능을 보였습니다.

단점:

- **높은 계산 비용**: BiLSTM과 CRF를 모두 학습해야 하므로 HMM이나 CRF 단독 모델보다 복잡하고 느립니다.
- **복잡한 모델 튜닝**: 하이퍼파라미터가 많아 튜닝이 어렵습니다.

5 실습/코드: BiLSTM-CRF 구현 (Python)

BiLSTM-CRF 모델은 `torch`와 `pytorch-crf` 같은 라이브러리를 사용하여 구현할 수 있습니다. (코드는 개념 이해를 위한 의사 코드(pseudo-code)에 가깝게 단순화되었습니다.)

```python
import torch
import torch.nn as nn
from TorchCRF import CRF

class BiLSTM_CRF(nn.Module):
    def __init__(self, vocab_size, tagset_size, embedding_dim, hidden_dim):
        super(BiLSTM_CRF, self).__init__()

        # 1. 임베딩층단어 ( -> 벡터)
        self.embedding = nn.Embedding(vocab_size, embedding_dim)

        # 2. BiLSTM 층특징 ( 추출기)
        #    hidden_dim // 2 는양방향이므로합치면이   hidden_dim 됨
        self.lstm = nn.LSTM(embedding_dim, hidden_dim // 2,
                            num_layers=1, bidirectional=True,
                            batch_first=True)

        # 3. Linear 층 (LSTM 출력을가 CRF 받을점수로변환 )
        # 이것이의    CRF 'Emission 점수가' 됨
        self.hidden2tag = nn.Linear(hidden_dim, tagset_size)

        # 4. CRF 층레이블 ( 결정자)
        self.crf = CRF(tagset_size, batch_first=True)

    def forward(self, sentences, tags=None, mask=None):
        # 1. 임베딩
        embeddings = self.embedding(sentences)

        # 2. BiLSTM 통과
        lstm_out, _ = self.lstm(embeddings)

        # 3. Emission 점수계산
        emissions = self.hidden2tag(lstm_out)

        if tags is not None:
            # 학습모드 : (emissions, tags) 간의로그가능도 (likelihood)를 계산
            # 이것이( 가Loss 됨. 우리는이값을최소화    = 음수로그가능도최소화   )
            log_likelihood = self.crf(emissions, tags, mask=mask)
            return -log_likelihood
        else:
            # 예측모드 : emissions 점수와의 CRF 전이점수를고려하여
            # 가장점수가높은최적의태그시퀀스를디코딩      (Viterbi)
            tag_seq = self.crf.decode(emissions, mask=mask)
            return tag_seq
```

Listing 1: BiLSTM-CRF 모델 클래스 정의 (PyTorch 예시)

코드 해설

- `nn.Embedding`: 입력된 단어 인덱스(예: 1, 3, 10)를 밀집 벡터(예: 100차원)로 변환합니다.
- `nn.LSTM`: 임베딩된 벡터 시퀀스를 입력받아, 양방향 문맥을 고려한 특징 벡터 시퀀스(`lstm_out`)를 출력합니다.
- `nn.Linear`: `lstm_out`(예: 256차원)을 `tagset_size`(예: 9개 태그) 차원의 '점수(Emission Score)'로 변환합니다. 이 점수는 "이 단어가 이 태그일 확률이 얼마나 높은가"에 대한 BiLSTM의 의견입니다.
- `self.crf`: 이 'Emission 점수'와 자신이 학습한 '전이 점수(Tramsition Score)'를 함께 고려합니다.
- **학습 시 (`forward`의 `if`문)**: 정답 `tags`를 알려주고, 해당 정답 경로의 확률(`log_likelihood`)을 높이도록 모델(BiLSTM의 가중치, CRF의 전이 행렬)을 업데이트합니다.
- **예측 시 (`forward`의 `else`문)**: 정답이 없으므로, `crf.decode`(보통 비터비 알고리즘 사용)를 통해 현재 Emission 점수와 전이 점수를 조합할 때 총점이 가장 높은 '최적의 경로'를 찾아 반환합니다.

6 핵심 개념 4: Variational Autoencoders (VAEs)

지금까지는 주어진 입력(X)에서 레이블(Y)을 맞히는 '판별 모델' 또는 '순차 모델'을 다뤘습니다. 이제부터는 모델이 스스로 무언가를 '생성'해내는 **생성 모델(Generative Models)**을 다룹니다.

6.1 시작하기: 일반 오토인코더 (Autoencoder, AE)

VAE를 이해하려면 먼저 일반 AE를 알아야 합니다.

□ 핵심 정보

오토인코더(AE)는 "입력을 압축했다가 다시 복원하는" 신경망입니다.

- **인코더 (Encoder)**: 입력 데이터(예: 고해상도 이미지)를 저차원의 벡터(예: 30차원 벡터)로 **압축**합니다. 이 압축된 벡터를 **잠재 변수(Latent Variable)** 또는 **코딩(Coding)**이라고 부릅니다.
- **병목 (Bottleneck)**: 이 잠재 변수가 있는 가장 좁은 구간입니다.
- **디코더 (Decoder)**: 압축된 잠재 변수를 입력받아 다시 원본 데이터(이미지)로 **복원**합니다.

학습 목표: (입력)과 (복원된 출력)이 최대한 같아지도록 (즉, 재구성 손실(Reconstruction Loss)을 최소화하도록) 학습합니다. 이 과정에서 인코더는 데이터의 핵심 특징(예: 얼굴 이미지의 눈, 코, 입 특징)만 잠재 변수에 압축하는 법을 배우게 됩니다.

6.2 일반 AE의 한계: 비구조화된 잠재 공간

AE는 데이터 압축에는 유용하지만, '생성 모델'로 쓰기에는 치명적인 한계가 있습니다.

주의사항

문제: 잠재 공간에 구멍(Hole)이 많다.

- AE는 학습 데이터(예: 입력 이미지 1000장)를 잠재 공간(예: 2차원 평면)의 특정 점(1000개의 점)으로 매핑합니다.
- 디코더는 **정확히 그 1000개의 점** 위치에서만 원본을 잘 복원하도록 학습됩니다.
- 만약 우리가 1번 이미지의 잠재 변수(A점)와 2번 이미지의 잠재 변수(B점)의 **중간 지점(C점)**에 있는 값을 디코더에 넣으면 어떻게 될까요?
- **결과:** C점은 학습된 적이 없는 '구멍(Hole)' 영역이므로, 디코더는 완전히 깨지거나 의미 없는 이미지(노이즈)를 생성합니다.

이처럼 잠재 공간이 '점'들로만 이루어져 있고 그 사이가 비어있는 것을 **"비구조화된(Unstructured) 잠재 공간"**이라고 부릅니다.

6.3 VAE의 혁신: 잠재 공간을 확률 분포로

VAE는 "잠재 공간을 점이 아닌, 확률 분포(영역)로 만들어서 구멍을 메우자!"라는 아이디어에서 시작합니다.

□ 핵심 정보

VAE(Variational Autoencoder)의 작동 방식:

1. **인코더**: 입력을 받아 **하나의 점(벡터)**을 출력하는 대신, 이 점 주변의 **확률 분포**를 나타내는 두 개의 벡터를 출력합니다.
   - **평균 (μ, mu)**: 분포의 중심점
   - **로그 분산 (log σ^2, sigma)**: 분포가 퍼진 정도 (분산)
2. **샘플링 (Sampling)**: 이 평균과 분산을 따르는 정규 분포(가우시안 노이즈)에서 **랜덤하게 잠재 변수(z)를 샘플링**합니다. (이것이 '랜덤성 주입'입니다.)
3. **디코더**: 이 랜덤하게 샘플링된 z를 입력받아 원본을 복원합니다.

결과: 인코더는 입력을 받을 때마다 매번 조금씩 다른(랜덤한) z를 디코더에게 줍니다. 디코더는 이 '살짝 흔들린' z값들로도 원본을 잘 복원해야 하므로, 특정 '점'이 아닌 그 '주변 영역' 전체에서 복원하는 법을 배우게 됩니다.

6.4 핵심: 잠재 손실 (Latent Loss / KL Divergence)

여기서 매우 중요한 질문이 생깁니다.

만약 VAE가 재구성에만 집중하면 어떻게 될까요? 모델(인코더)은 '랜덤성'이 재구성을 방해한다는 것을 알고 있습니다.

모델의 꼼수: 인코더가 분산(σ)을 0으로 만들어 버립니다.

- 분산이 0이 되면 확률 분포는 '점'이 되고, 랜덤 샘플링은 항상 평균(μ) 값만 뽑게 됩니다.
- 랜덤성이 사라지고, VAE는 일반 AE와 똑같이 행동하게 됩니다.
- 잠재 공간은 다시 '비구조화된' 상태가 됩니다.

이 꼼수를 막기 위해 VAE는 두 번째 손실 함수, 즉 **잠재 손실**을 도입합니다.

□ 핵심 정보

잠재 손실 (Latent Loss): (정확히는 쿨백-라이블러 발산, D_{KL})
이 손실은 인코더가 만든 모든 확률 분포(μ, σ)가 "특정 기준 분포" (보통 $\mu = 0, \sigma = 1$인 표준 정규 분포)와 비슷해지도록 강제하는 패널티입니다.
잠재 손실의 두 가지 역할:

1. **분산(σ)이 0이 되는 것을 방지**: σ를 0이 아닌 1에 가깝게 유지하도록 강제하여, 인코더가 **적절한 랜덤성(노이즈)**을 유지하게 만듭니다. (잠재 공간에 '영역'을 만들도록 강제)
2. **평균(μ)을 0 근처로 집결**: 모든 분포의 중심(μ)을 원점(0) 근처로 모읍니다.

최종 효과 (가장 중요): 모든 분포가 원점 근처로 모이고(by $\mu \to 0$), 적절한 분산(by $\sigma \to 1$)을 가지게 되면, 서로 다른 이미지(예: A이미지, B이미지)의 잠재 공간 분포가 서로 **오버랩 (Overlap)**하게 됩니다.
이 '오버랩' 덕분에 잠재 공간에는 더 이상 '구멍'이 존재하지 않습니다. (이를 **"구조화된 (Structured) 잠재 공간"**이라 부름)
이제 A이미지의 분포와 B이미지의 분포 사이의 임의의 점 C를 뽑아 디코더에 넣어도, 그럴듯한 (A와 B를 섞은 듯한) 새로운 이미지가 생성됩니다.

7 핵심 개념 5: Generative Adversarial Networks (GANs)

GAN(생성적 적대 신경망)은 VAE와는 완전히 다른 철학을 가진 생성 모델입니다. VAE가 확률과 분포를 이용했다면, GAN은 두 신경망의 '경쟁'을 이용합니다.

7.1 핵심 비유: 위조지폐범 vs 경찰

GAN의 핵심 아이디어는 두 개의 신경망이 서로를 속이고 잡아내기 위해 경쟁(Adversarial)하는 것입니다.

□ 핵심 정보

GAN의 두 가지 구성 요소:

1. **생성자 (Generator, G): "위조지폐범"**
   - **입력:** 무작위 노이즈 (Noise, z) (예: 100차원의 랜덤 벡터)
   - **역할:** 이 노이즈를 입력받아 '가짜' 데이터(예: 가짜 이미지)를 생성합니다.
   - **목표:** 판별자가 "진짜"라고 속을 만큼 정교한 가짜 데이터를 만드는 것.
2. **판별자 (Discriminator, D): "경찰"**
   - **입력:** '진짜' 데이터 (학습 데이터셋) 또는 '가짜' 데이터 (생성자가 만든 것)
   - **역할:** 입력된 데이터가 진짜인지 가짜인지 판별합니다. (이진 분류: 0=Fake, 1=Real)
   - **목표:** 생성자가 만든 가짜를 '가짜(Fake)'라고 정확히 잡아내고, 진짜는 '진짜(Real)'라고 정확히 맞히는 것.

7.2 학습 과정 (Minimax Game)

두 네트워크는 서로의 이익이 반대되는 '제로섬 게임(Minimax Game)'을 벌이며 함께 똑똑해집니다.

GAN 학습 단계

학습은 두 단계를 번갈아 가며 진행됩니다.

1단계: 판별자(D) 학습 (생성자(G)는 고정)

- 생성자(G)가 노이즈로부터 '가짜 이미지'를 생성합니다.
- 판별자(D)에게 (1) '진짜 이미지'와 (2) '가짜 이미지'를 보여줍니다.
- 판별자(D)는 (1)에는 1(Real)이라고 답하고, (2)에는 0(Fake)이라고 답하도록 학습(업데이트)됩니다.
- (비유) 경찰이 진짜 지폐와 위조지폐를 보며 감별법을 익힙니다.

2단계: 생성자(G) 학습 (판별자(D)는 고정)

- 생성자(G)가 노이즈로부터 '가짜 이미지'를 생성합니다.
- 이 '가짜 이미지'를 **고정된** 판별자(D)에게 보여줍니다.
- 생성자(G)는 판별자(D)가 이 이미지를 보고 0(Fake)이 아닌 **1(Real)이라고 답하도록** 자신의 가중치를 학습(업데이트)합니다.
- (비유) 위조지폐범이 경찰(판별자)을 속일 수 있는(즉, 경찰이 '진짜'라고 착각할 만한) 더 정교한 위조지폐를 만드는 법을 배웁니다.

이 과정을 수없이 반복하면, 생성자(G)는 실제 데이터와 구별이 불가능할 정도로 고품질의 가짜 데이터를 생성하게 되고, 판별자(D)는 더 이상 진짜와 가짜를 구별하지 못하게 됩니다 (판별 확률 0.5에 수렴).

7.3 GAN의 진화와 응용

초기 GAN은 학습이 불안정했지만, 이후 DCGAN (합성곱 신경망 적용), StyleGAN 등 고도화된 모델이 등장하며 놀라운 성능을 보였습니다.

□ 예제:

예시: StyleGAN "This Person Does Not Exist" / "This Cat Does Not Exist"와 같은 웹사이트가 바로 StyleGAN을 이용한 예시입니다.

- 이 웹사이트들이 보여주는 사람 얼굴이나 고양이 이미지는 **세상에 존재하지 않는** 이미지입니다.
- GAN(생성자)이 수많은 실제 사진을 학습한 뒤, 데이터의 '특징'(예: 머리 스타일, 눈 색깔, 배경)을 이해하고 이를 조합하여 완전히 새로운(하지만 현실적인) 이미지를 생성해낸 것입니다.

장점: VAE보다 훨씬 더 선명하고 고품질의 이미지를 생성하는 경향이 있습니다. **단점:** 학습이 매우 불안정하고(예: 생성자가 한 가지 이미지만 계속 생성하는 'Mode Collapse'), 많은 데이터와 계산 자원이 필요합니다.

8 체크리스트: 학습 점검표

이 문서를 읽고 다음 질문에 스스로 답할 수 있는지 확인해 보세요.

최종 점검 리스트

- ☐ 마코프 속성이 무엇이며, 왜 HMM의 한계와 연결되는지 설명할 수 있는가?
- ☐ HMM과 CRF의 결정적인 차이는 무엇인가? (생성 모델 vs 판별 모델)
- ☐ CRF가 HMM보다 유연한 이유는 무엇인가? (특징 함수)
- ☐ CRF의 가장 큰 단점(수동 특징 공학)이 무엇을 의미하는지 예시를 들어 설명할 수 있는가?
- ☐ BiLSTM-CRF 아키텍처에서 BiLSTM과 CRF는 각각 어떤 역할을 담당하는가?
- ☐ 왜 BiLSTM의 출력에 Softmax 대신 CRF를 사용하는 것이 더 좋은가? (레이블 의존성)
- ☐ 일반 오토인코더(AE)로 고품질의 '새로운' 이미지를 생성하기 어려운 이유는 무엇인가? (비구조화된 잠재 공간)
- ☐ VAE는 AE와 달리 왜 인코더가 μ와 σ를 출력하는가? (랜덤성 주입)
- ☐ VAE에서 '잠재 손실(KL Divergence)'이 없다면 어떤 문제가 발생하는가? ($\sigma \to 0$)
- ☐ VAE의 잠재 손실이 어떻게 잠재 공간을 '구조화'하는가? (오버랩 강제)
- ☐ GAN의 생성자(G)와 판별자(D)의 목표는 각각 무엇인가?
- ☐ GAN의 학습 과정(2단계)을 '경찰과 위조지폐범' 비유를 사용하여 설명할 수 있는가?

9 FAQ: 초심자 주요 질문

HMM은 이제 쓸모가 없나요? 그렇지 않습니다. HMM은 모델이 단순하고 계산이 빠르며, 데이터가 매우 적을 때도 비교적 안정적으로 작동합니다. 딥러닝 모델(BiLSTM 등)이 성능이 좋지만 학습을 위해 훨씬 많은 데이터와 자원이 필요합니다. 간단한 시퀀스 문제나 초기 베이스라인 모델로 여전히 유용합니다.

CRF와 로지스틱 회귀(Logistic Regression)는 무슨 관계인가요? CRF는 종종 "로지스틱 회귀의 시퀀스(순차) 버전"이라고 불립니다.

- **로지스틱 회귀**: 여러 특징(X)을 입력받아 하나의 레이블(Y)을 예측하는 판별 모델입니다. (예: $P(Y = 1|X)$)
- **CRF**: 여러 특징(X 시퀀스)을 입력받아 **레이블 시퀀스(Y 시퀀스)**를 예측하는 판별 모델입니다. (예: $P(Y\_1, Y\_2, ...|X\_1, X\_2, ...)$)

둘 다 특징을 유연하게 사용하고 $P(Y|X)$를 직접 모델링한다는 공통점이 있습니다.

생성 모델로 VAE와 GAN 중 어느 것이 더 좋은가요? 목적에 따라 다릅니다.

- **VAE**:
  - **장점:** 학습이 안정적이며, 잠재 공간이 잘 구조화되어 데이터의 '분포'를 학습하기 좋습니다.
  - **단점:** 생성된 이미지가 GAN에 비해 다소 흐릿(Blurry)한 경향이 있습니다.
- **GAN**:
  - **장점:** 매우 선명하고 현실적인 고품질 이미지를 생성하는 데 탁월합니다.
  - **단점:** 학습이 매우 불안정하고(Minimax 최적점을 찾기 어려움) 하이퍼파라미터에 민감합니다.

10 빠르게 훑어보기 (1페이지 요약)

HMM (Hidden Markov Model)

- **개념:** 숨겨진 상태(Y)가 마코프 속성을 따르며 전이하고, 각 상태가 관찰(X)을 방출(생성)함.
- **모델:** 생성 모델 ($P(X, Y)$).
- **핵심:** 전이 확률 ($P(Y_t|Y_{t-1})$), 방출 확률 ($P(X_t|Y_t)$).
- **한계:** 미래는 오직 현재 상태(Y_t)에만 의존. 다른 관찰(X)을 참고하지 못함.

CRF (Conditional Random Field)

- **개념:** HMM의 한계 극복. 관찰 시퀀스(X) 전체를 조건으로 레이블 시퀀스(Y)의 확률($P(Y|X)$)을 직접 모델링.
- **모델:** 판별 모델 ($P(Y|X)$).
- **핵심:** 유연한 특징 함수(f_k)와 가중치(λ_k)의 조합.
- **한계:** 유용한 특징(f_k)을 **사람이 직접** 설계해야 함 (수동 특징 공학).

BiLSTM-CRF

- **개념:** CRF의 수동 특징 공학 문제를 BiLSTM으로 자동화.
- **BiLSTM (특징 추출기):** 양방향 문맥을 읽어 고품질 특징(Emission 점수)을 자동 생성.
- **CRF (레이블 결정자):** BiLSTM의 특징을 받아, 레이블 간의 전이(Transition) 규칙을 적용하여 최적의 시퀀스 결정.
- **장점:** 자동 특징 공학 + 높은 정확도.

VAE (Variational Autoencoder)

- **개념:** 일반 AE의 '비구조화된 잠재 공간' 문제를 해결한 생성 모델.
- **핵심:** 인코더가 잠재 변수를 '점'이 아닌 '확률 분포(μ, σ)'로 출력. 이 분포에서 샘플링하여 디코더에 주입 (랜덤성).
- **잠재 손실(D_{KL}):** 모델이 랜덤성을 포기하는 꼼수($\sigma \to 0$)를 막는 패널티. 분포들을 원점 근처로 모아 '오버랩'시켜 잠재 공간을 빽빽하게 채움.

GAN (Generative Adversarial Network)

- **개념:** 생성자(G)와 판별자(D)가 경쟁하며 학습하는 생성 모델.
- **G (생성자/위조범):** 노이즈(z)를 받아 가짜 데이터를 생성. D를 속이는 것이 목표.
- **D (판별자/경찰):** 진짜와 가짜를 구별. G에게 속지 않는 것이 목표.
- **장점:** 지도 데이터 없이 학습 가능. 매우 현실적인 고품질 이미지 생성.

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 12

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 12의 핵심 개념 학습

Contents

Lecture 12: Attention Mechanism & Transformers

자연어 처리의 혁명: 어텐션 메커니즘과 트랜스포머 아키텍처 완전 정복

개요 (Overview)

이번 강의는 현대 자연어 처리(NLP)의 가장 중요한 분기점인 **Attention Mechanism(어텐션 메커니즘)**과 이를 기반으로 한 **Transformer(트랜스포머)** 모델을 다룹니다.

기존 순환 신경망(RNN/LSTM)이 가진 한계점(병목 현상, 기울기 소실, 병렬화 불가)을 극복하기 위해 어텐션이 어떻게 등장했는지 살펴보고, 나아가 "순환(Recurrence)을 완전히 제거한" 트랜스포머 아키텍처의 작동 원리(Self-Attention, Multi-Head Attention, Positional Encoding)를 학습합니다. 마지막으로 이 구조가 어떻게 BERT와 GPT로 발전했는지 이해합니다.

□ 핵심 요약

핵심 목표

- RNN 기반 Seq2Seq 모델의 한계점(정보 병목, 장기 의존성 문제) 이해
- 어텐션(Attention)의 기본 원리: "중요한 부분에 집중한다"는 개념
- 트랜스포머(Transformer)의 구조: Encoder-Decoder, Self-Attention (Q, K, V)
- BERT(인코더 기반)와 GPT(디코더 기반)의 차이점

1 필수 용어 정리

본격적인 학습에 앞서, 낯설 수 있는 핵심 용어를 미리 정리합니다.

Table 1: *Lecture 12* 핵심 용어 사전

용어 (Term)	한국어	쉬운 설명
Seq2Seq	시퀀스 투 시퀀스	입력 문장을 받아 출력 문장을 만드는 구조 (예: 번역). 인코더와 디코더로 구성됨.
Bottleneck	병목 현상	긴 문장의 모든 정보를 하나의 고정된 크기 벡터에 억지로 구겨 넣을 때 발생하는 정보 손실.
Context Vector	문맥 벡터	입력 문장의 정보를 요약한 벡터. 어텐션에서는 매 시점마다 동적으로 변함.
Attention	어텐션(주목)	출력 단어를 만들 때, 입력 문장의 어느 단어를 더 '주의 깊게' 볼지 결정하는 기술.
Self-Attention	셀프 어텐션	문장 내의 단어들이 서로 어떤 관계가 있는지 파악하는 것 (예: '그것'이 가리키는 단어 찾기).
Query (Q)	쿼리	"내가 지금 찾고자 하는 정보는?" (질문자 역할)
Key (K)	키	"나는 누구인가?" (정보의 식별자 역할)
Value (V)	밸류	"내가 가진 실제 내용은 무엇인가?" (정보의 내용물 역할)
Transformer	트랜스포머	RNN 없이 오직 어텐션만으로 구성된 딥러닝 모델 아키텍처.

2 RNN의 한계와 어텐션의 등장

2.1 기존 RNN/LSTM의 문제점

과거 번역 모델(Encoder-Decoder)은 RNN을 사용했습니다. 입력 문장을 인코더가 읽어서 하나의 **고정된 크기의 벡터(Context Vector)**로 압축하고, 디코더가 이를 풀어서 번역문을 만들었습니다.

- **정보 병목(Information Bottleneck):** 100단어짜리 긴 문장을 겨우 128차원 벡터 하나에 압축해야 합니다. 정보 손실이 발생할 수밖에 없습니다.
- **장기 의존성(Long-Term Dependency):** 문장이 길어지면 앞부분의 내용이 뒷부분까지 전달되지 못하고 희석됩니다(Vanishing Gradient).
- **순차적 처리(Sequential Processing):** 단어를 하나씩 순서대로 처리해야 하므로 병렬 계산(GPU 활용)이 어렵고 속도가 느립니다.

2.2 어텐션(Attention)의 아이디어

직관적 이해: 통역사의 비유 RNN이 "문장 전체를 외운 뒤 한 번에 번역하는 통역사"라면,
어텐션은 **"번역할 때마다 원문을 다시 힐끔힐끔 쳐다보는 통역사"**입니다.

어텐션 메커니즘은 디코더가 단어를 출력할 때마다 인코더의 **모든 입력 단어**를 다시 참고합니다. 단, 그냥 보는 것이 아니라 **"지금 번역할 단어와 관련된 부분"**에 가중치(Attention Weight)를 두어 봅니다.

2.3 수식 없는 어텐션 작동 원리

1. **스코어 계산(Alignment Score):** 현재 디코더의 상태와 인코더의 각 단어가 얼마나 관련 있는지 계산합니다. 2. **확률 변환(Softmax):** 스코어를 0 1 사이의 확률값(합이 1)으로 바꿉니다. 이것이 '어텐션 가중치'입니다.

- 예: "Zone(프랑스어)"을 번역할 때, "Area(영어)"에 0.7, "Economic"에 0.2의 가중치를 둠.

3. **가중합(Weighted Sum):** 인코더의 정보들에 가중치를 곱해 더합니다. 이것이 새로운 동적 문맥 벡터(Dynamic Context Vector)가 됩니다.

[시각화: 어텐션 맵]
프랑스어 "Zone"이 출력될 때 → 영어 "Area" 부분의 픽셀이 밝게 빛남.
즉, 모델이 번역 시점에 'Area'라는 단어에 집중(Attend)하고 있음을 시각적으로 확인 가능.

Figure 1: 어텐션 가중치를 시각화하면 단어 간의 연관성을 알 수 있습니다.

3 트랜스포머 (Transformer)

2017년 구글은 *"Attention Is All You Need"*라는 논문을 통해 **RNN을 완전히 제거한** 모델인 트랜스포머를 제안합니다.

3.1 왜 RNN을 버렸는가?

RNN은 순서대로 계산해야 하므로 느립니다. 트랜스포머는 문장 전체를 한 번에 행렬로 입력받아 병렬 처리가 가능합니다. 이를 통해 학습 속도를 비약적으로 높이고, 더 많은 데이터를 학습할 수 있게 되었습니다(GPT, BERT의 탄생 배경).

3.2 핵심 1: 포지셔널 인코딩 (Positional Encoding)

RNN이 없으면 단어의 **순서 정보**가 사라집니다. "I love you"와 "You love I"를 구별할 수 없게 됩니다. 따라서, 단어 벡터에 **위치 정보(Positional Encoding)**를 더해줍니다.

- 사인(sin)과 코사인(cos) 함수를 이용해 각 위치마다 고유한 패턴의 값을 더해줍니다.
- 이를 통해 모델은 단어의 상대적/절대적 위치를 파악할 수 있습니다.

3.3 핵심 2: 셀프 어텐션 (Self-Attention)과 Q, K, V

트랜스포머의 심장입니다. 문장 내의 단어들이 서로 어떤 관계인지 파악합니다. 이를 위해 각 단어를 세 가지 역할로 나눕니다.

Q, K, V 비유: 파일 검색 시스템

- **Query (Q):** 검색창에 입력한 검색어 ("Sky에 대해 알고 싶어")
- **Key (K):** 파일의 제목/태그 ("이 파일은 Cloud에 관한 것임")
- **Value (V):** 파일의 실제 내용 ("구름은 흰색이고 하늘에 떠 있다...")

작동 과정: 1. 나의 Q와 상대방들의 K를 내적(Dot Product)하여 유사도를 구합니다. (Sky와 Cloud는 관련성이 높으므로 점수가 높음) 2. 점수를 $\sqrt{d_k}$로 나누고(스케일링), Softmax를 취해 확률로 만듭니다. 3. 이 확률을 상대방들의 V에 곱해서 더합니다. 4. 결과: "Sky"라는 단어 벡터는 "Cloud", "Blue" 등의 문맥 정보를 흡수하여 더 풍부한 의미를 갖게 됩니다.

Scaled Dot-Product Attention 공식

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- $\sqrt{d_k}$로 나누는 이유: 벡터 차원(d_k)이 커지면 내적 값이 너무 커져서, Softmax의 기울기(Gradient)가 소실되는 것을 방지하기 위함(학습 안정화).

3.4 핵심 3: 멀티 헤드 어텐션 (Multi-Head Attention)

한 번만 어텐션을 수행하는 것이 아니라, 여러 개의 헤드(Head)로 나누어 병렬로 수행합니다.

- **비유:** 같은 문장을 읽을 때, 한 명은 문법을 보고, 한 명은 의미를 보고, 한 명은 대명사 지칭을

봅니다. 나중에 이들의 통찰을 합칩니다.

- 이를 통해 모델은 다양한 관점(Representation Subspaces)에서 문장을 이해할 수 있습니다.

3.5 핵심 4: 마스킹 (Masking)

트랜스포머의 디코더(Decoder) 학습 시 사용됩니다.

- **문제:** 디코더는 정답 문장을 생성해야 하는데, 학습 시 정답 전체를 한 번에 입력받습니다. 모델이 뒤에 올 정답 단어를 미리 "컨닝(Cheating)"하면 안 됩니다.
- **해결:** 현재 위치보다 뒤에 있는 단어들의 점수를 $-\infty$(음의 무한대)로 만들어 Softmax 결과가 0이 되게 합니다. 이를 **Look-ahead Mask**라고 합니다.

4 실습 코드 분석 (Keras)

다음은 Keras를 이용한 어텐션 레이어 구현의 핵심 로직입니다.

```
class AttentionLayer(Layer):
    def call(self, inputs):
        # 1. 스코어계산 (tanh 사용)
        # 와inputs 가중치를 W 내적하고를 bias 더한뒤 tanh 통과
        v = tf.tanh(tf.tensordot(inputs, self.W, axes=1) + self.b)

        # 2. Alignment Score 계산
        # 벡터와 u 내적하여스칼라점수 (vu) 생성
        vu = tf.tensordot(v, self.u, axes=1, name='vu')

        # 3. 어텐션가중치 (alpha) 계산 (Softmax)
        alphas = tf.nn.softmax(vu, axis=1)

        # 4. 문맥벡터 (Context Vector) 생성
        # 입력(inputs)에 가중치(alphas)를 곱해서합침 (reduce_sum)
        output = tf.reduce_sum(inputs * tf.expand_dims(alphas, -1), axis=1)

        return output
```

Listing 1: 사용자 정의 Attention Layer 핵심 로직

- 위 코드는 RNN 기반 어텐션 구조입니다. 트랜스포머의 QK^T 방식과는 약간 다르지만(여기선 $tanh$ 사용), "가중치를 계산해서 입력의 가중합을 구한다"는 핵심 원리는 같습니다.

5 BERT와 GPT: 트랜스포머의 자식들

트랜스포머 구조를 반으로 쪼개서 각각 발전시킨 것이 현재의 최신 모델들입니다.

- **BERT (Bidirectional Encoder Representations from Transformers):** 문맥을 양쪽에서 파악하므로 "이해(Understanding)" 능력이 뛰어납니다.
- **GPT (Generative Pre-trained Transformer):** 다음 단어를 예측하는 방식으로 학습하므로 "생성(Generation)" 능력이 뛰어납니다.

Table 2: *BERT*와 *GPT*의 비교

구분	BERT (Encoder 기반)	GPT (Decoder 기반)
기반 구조	트랜스포머의 **인코더(Encoder)**	트랜스포머의 **디코더(Decoder)**
방향성	양방향 (Bidirectional)	단방향 (Unidirectional, 왼쪽→오른쪽)
주특기	문장의 의미 파악, 빈칸 채우기, 분류	문장 생성, 다음 단어 예측
비유	문장 전체를 보고 해석하는 독해 전문가	앞 단어만 보고 뒷말 잇는 작가
활용 예	감성 분석, 질의응답(QA), 개체명 인식	챗봇, 소설 쓰기, 코드 생성

6 학습 점검 체크리스트 & FAQ

6.1 체크리스트

[ ] RNN의 장기 의존성 문제와 병목 현상이 무엇인지 설명할 수 있는가?

[ ] 어텐션이 입력 데이터의 가중합(Weighted Sum)을 구하는 과정임을 이해했는가?

[ ] 트랜스포머가 RNN을 사용하지 않고도 순서 정보를 아는 방법(Positional Encoding)을 아는가?

[ ] Self-Attention에서 Q, K, V가 각각 어떤 역할을 하는지 비유를 들어 설명할 수 있는가?

[ ] 트랜스포머의 인코더 부분(BERT)과 디코더 부분(GPT)의 차이를 구분할 수 있는가?

6.2 자주 묻는 질문 (FAQ)

Q1. 어텐션(Attention)과 셀프 어텐션(Self-Attention)은 다른 건가요?
A. 원리는 같지만 **대상**이 다릅니다. 일반 어텐션(Seq2Seq)은 디코더가 인코더를 보는 것이고(서로 다른 문장 간 관계), 셀프 어텐션은 문장 내에서 자기 자신 안의 단어들끼리의 관계를 보는 것입니다(예: 'I'와 'am'의 관계).

Q2. 왜 Q, K, V를 따로 만드나요? 그냥 단어 벡터 그대로 쓰면 안 되나요?
A. 가능은 하지만, 유연성(Flexibility)이 떨어집니다. 같은 단어라도 "질문할 때의 나(Query)", "비교 대상으로서의 나(Key)", "정보 제공자로서의 나(Value)"의 역할에 따라 다르게 표현될 수 있도록 별도의 가중치 행렬(W^Q, W^K, W^V)을 학습시키는 것이 성능이 훨씬 좋습니다.

Q3. 마스킹(Masking)은 왜 디코더에만 있나요?
A. 인코더는 이미 주어진 문장을 분석하는 것이라 미래 단어를 봐도 상관없습니다(오히려 다 봐야 문맥을 압니다). 하지만 디코더는 문장을 생성하는 과정이므로, 아직 생성하지 않은 미래의 단어를 미리 보고 예측하면 학습이 제대로 되지 않기 때문입니다.

Q4. 강의 초반 퀴즈 내용 중 CRF가 HMM보다 좋은 점은?
A. HMM은 바로 이전 상태(State)에만 의존한다고 가정하지만, CRF(Conditional Random Field)는 문맥 전체(과거와 미래)의 특성을 동시에 고려할 수 있어 더 복잡한 의존성을 모델링할 수 있습니다.

빠르게 훑어보기 (1분 요약)

Summary

- **문제:** RNN은 느리고, 긴 문장을 까먹음.
- **해결 1 (Attention):** 문장을 요약하지 말고, 필요할 때마다 원문을 다시 보자.
- **해결 2 (Transformer):** RNN을 버리고 어텐션만 쓰자. 병렬 처리로 속도 UP.
- **Self-Attention:** $Attention(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$
- **BERT:** 인코더 사용, 문맥 이해(양방향).
- **GPT:** 디코더 사용, 문장 생성(단방향).

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 13

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 13의 핵심 개념 학습

기계 번역(Machine Translation) 심층 분석

자연어 처리(NLP) 입문 - 13주차 강의 통합 요약

December 10, 2025

□ 핵심 요약

문서 개요 (Executive Summary)

- **기계 번역(MT)**은 소프트웨어를 통해 인간 수준의 번역 품질을 목표로 하는 기술입니다.
- 초기 **규칙 기반(RBMT)**에서 **통계 기반(SMT)**을 거쳐, 현재는 **신경망 기반(NMT)**이 주류입니다.
- **Seq2Seq** 모델은 입력과 출력을 연결하는 혁신을 가져왔으나 병목 현상이 있었고, 이를 **어텐션(Attention)**과 **트랜스포머(Transformer)**가 해결했습니다.
- 번역 품질 평가는 주로 **BLEU 점수**를 사용하며, 최근에는 규칙과 신경망을 합친 **하이브리드 모델**도 특정 분야에서 강세를 보입니다.

Contents

1 기본 개념 및 용어 정리

기계 번역을 처음 접할 때 반드시 알아야 할 핵심 용어들입니다. 이 용어들은 문서 전체에서 반복되므로 미리 숙지하면 이해가 빠릅니다.

Table 1: 기계 번역 핵심 용어 정의

용어 (약어)	원어	쉬운 설명 및 비고
MT	Machine Translation	컴퓨터가 한 언어를 다른 언어로 번역하는 기술 전반.
RBMT	Rule-Based MT	1950 80년대 방식. 사람이 직접 문법 규칙과 사전을 입력해줌. (예: "I am" → "나는 이다")
SMT	Statistical MT	1990 2010년대 방식. 수많은 번역 데이터에서 '확률'을 계산해 번역. (예: 이 단어 뒤엔 저 단어가 올 확률이 80%)
NMT	Neural MT	2010년 이후 방식. 딥러닝(인공신경망)을 사용해 문장 전체의 맥락을 이해하고 번역. 현재의 주류.
Seq2Seq	Sequence-to-Sequence	입력 문장(Sequence)을 받아 출력 문장(Sequence)을 만들어내는 딥러닝 모델 구조.
Attention	Attention Mechanism	긴 문장을 번역할 때, "지금 번역하는 단어와 관련된 입력 단어"에 집중(Attention)하게 만드는 기술.
Transformer	Transformer	RNN을 버리고 'Attention'만으로 구성하여 속도와 성능을 획기적으로 높인 현대 AI의 기반 모델 (BERT, GPT의 조상).
BLEU	Bilingual Evaluation Understudy	기계 번역이 얼마나 잘 됐는지 채점하는 점수. 인간 번역과 얼마나 겹치는지 확인.

2 기계 번역(MT)의 개요와 발전사

2.1 기계 번역이란 무엇인가?

기계 번역은 단순히 단어를 1:1로 바꾸는 것이 아닙니다. 목표는 **"의미와 문화적 뉘앙스를 보존하면서 문법적으로 정확한 문장을 만드는 것"**입니다.

- **목표:** 인간 번역가의 품질을 복제하는 것.
- **용도:** 국제 비즈니스, 외교 문서, 실시간 회의 통역, 관용구(예: "hit the nail on the head" → "정곡을 찌르다")의 적절한 의역 등.

2.2 발전의 3단계 흐름

기계 번역은 크게 세 가지 시대를 거쳐 발전했습니다. "왜 이전 기술이 도태되었는가?"를 이해하는 것이 중요합니다.

1. 규칙 기반 (RBMT, 1950s–1980s)

- **방식:** 언어학자가 문법 규칙과 사전을 일일이 코딩했습니다.
- **장점:** 날씨 예보나 법률 문서처럼 형식이 정해진 문서는 아주 정확합니다.
- **단점:** 예외가 너무 많아 확장이 어렵고, 자연스러운 구어체 번역에 실패합니다.
- **사례:** 1954년 조지타운-IBM 실험 (러시아어 → 영어).

2. 통계 기반 (SMT, 1990s–2010s)

- **방식:** 수많은 이중 언어 데이터(예: UN 회의록)를 통계적으로 분석해 "이 단어 옆엔 저 단어가 온다"는 확률을 학습합니다.
- **장점:** 규칙을 일일이 짤 필요가 없고, 데이터가 많으면 성능이 좋아집니다.
- **단점:** 문맥보다는 '구(Phrase)' 단위로 쪼개서 번역하므로 문장이 뚝뚝 끊기는 느낌이 듭니다.

3. 신경망 기반 (NMT, 2010s–Present)

- **방식:** 딥러닝(Deep Learning)을 사용해 문장 전체를 하나의 벡터(숫자 덩어리)로 이해하고 번역합니다.
- **장점:** **"End-to-End" 학습** (입력에서 출력까지 한 번에 학습). 문맥을 파악하여 훨씬 유창하고 자연스럽습니다.
- **현재:** Seq2Seq와 Transformer 모델이 이 시대를 이끌고 있습니다.

3 기계 번역의 난제 (Challenges)

번역은 왜 어려울까요? 컴퓨터 입장에서 가장 헷갈리는 문제들입니다.

1. **언어의 모호성 (Ambiguity)**
   - 단어 하나가 여러 뜻을 가집니다.
   - 예시: 영어 "Bank"는 '은행'일 수도, '강둑'일 수도 있습니다. 문맥 없이 "I went to the bank"만 보고는 알 수 없습니다.
2. **구조적 차이 (Structural Differences)**
   - 어순이 다릅니다.
   - 예시: 영어는 SVO (주어-동사-목적어)인 반면, 한국어는 SOV (주어-목적어-동사)입니다. 이를 맞추려면 문장 전체를 다 듣고 재배열해야 합니다.
3. **관용구 (Idiomatic Expressions)**
   - 직역하면 의미가 통하지 않습니다.
   - 예시: "Shoot the breeze" → (직역) 산들바람을 쏘다? → (의역) 수다를 떨다.
4. **문맥 보존 (Context Preservation)**
   - 긴 문단에서 앞 내용을 기억해야 합니다. 대명사(그것, 그 사람)가 무엇을 가리키는지 파악하는 것은 매우 어렵습니다.

4 핵심 기술 1: Seq2Seq 모델

4.1 Seq2Seq의 구조

NMT(신경망 번역)의 시초가 된 모델입니다. 크게 두 부분으로 나뉩니다.

- **인코더 (Encoder):** 원문을 읽고 그 의미를 압축하여 하나의 **'컨텍스트 벡터(Context Vector)'**로 만듭니다. (독해 담당)
- **디코더 (Decoder):** 컨텍스트 벡터를 받아서 도착 언어로 문장을 생성합니다. (작문 담당)

주의사항

Seq2Seq의 치명적 단점: 병목 현상 (Bottleneck) 초기 Seq2Seq는 긴 문장을 아주 작은 벡터 하나에 억지로 구겨 넣어야 했습니다.
비유: 책 한 권을 읽고 내용을 단 한 문장으로 요약한 뒤, 그 요약본만 보고 다시 책 전체를 다른 언어로 써내라는 것과 같습니다. 정보 손실이 발생하여 긴 문장 번역 품질이 떨어집니다.

4.2 해결책: 어텐션 메커니즘 (Attention Mechanism)

어텐션은 **"다 기억하려 하지 말고, 필요할 때 원문을 다시 컨닝하자!"**는 아이디어입니다.

- 디코더가 번역할 때, 문장의 모든 부분을 동일하게 보는 게 아니라 **관련된 단어에 가중치(집중)**를 둡니다.
- 병목 현상을 해결하고 긴 문장 번역 성능을 획기적으로 높였습니다.

5 핵심 기술 2: 트랜스포머 (Transformers)

2017년 논문 *"Attention is All You Need"*에서 발표된 모델로, 현대 AI(GPT, BERT)의 조상입니다.

5.1 기존 모델(RNN/LSTM)과의 차이

- **기존:** 단어를 순서대로 하나씩 처리(순차적). 시간이 오래 걸리고 앞부분 내용을 까먹기 쉽습니다.
- **트랜스포머:** 문장 전체를 한 번에 처리(**병렬 처리**). 속도가 매우 빠르고 문장 내 단어 간의 관계를 한눈에 파악합니다.

5.2 핵심 구성 요소

1. **Self-Attention (셀프 어텐션):** 문장 내의 단어들이 서로 어떤 관계가 있는지 계산합니다. (예: "그것"이 앞의 "사과"를 가리키는지, "바나나"를 가리키는지 파악)
2. **Positional Encoding (위치 인코딩):** 한꺼번에 입력받으므로 단어 순서 정보를 인위적으로 추가해 줍니다.
3. **Encoder-Decoder 구조:** 여전히 번역을 위해 인코더와 디코더 구조를 유지하지만, 내부는 전부 어텐션으로 채워져 있습니다.

□ 핵심 요약

트랜스포머의 충격적인 효과

- **확장성:** 병렬 처리가 가능해져서 엄청나게 큰 데이터로 학습이 가능해졌습니다.
- **범용성:** 번역뿐만 아니라 요약, 챗봇, 심지어 이미지 분석(Vision Transformer)이나 단백질 구조 예측에도 쓰입니다.

6 평가 지표: BLEU Score

기계 번역이 잘 됐는지 사람이 일일이 채점할 수 없으므로, **자동화된 채점 방식**이 필요합니다.

6.1 BLEU (Bilingual Evaluation Understudy)의 개념

기계가 번역한 문장이 **사람(참조, Reference)이 번역한 문장과 얼마나 단어가 많이 겹치는가**를 측정합니다.

6.2 계산 방법 (직관적 이해)

1. **N-gram 정밀도 (Precision):** 단어(1-gram), 두 단어 짝(2-gram), 세 단어 짝(3-gram)... 이 얼마나 일치하는지 봅니다.

$$\text{Precision}_n = \frac{\sum \text{Matches (일치하는 n-gram 수)}}{\sum \text{Total (기계 번역의 총 n-gram 수)}} \tag{1}$$

2. **간결성 페널티 (Brevity Penalty, BP):** 기계가 틀릴까 봐 단어 하나만 내뱉는 꼼수를 부리지 못하게, **길이가 참조 문장보다 짧으면 점수를 깎습니다.**
3. **최종 점수:** N-gram 정밀도의 평균에 BP를 곱해서 계산합니다. (1에 가까울수록 완벽, 0에 가까울수록 엉망)

주의사항

BLEU 점수의 한계

- **의미 파악 불가:** "아름답다"를 "예쁘다"로 번역하면, 뜻은 통하지만 단어가 달라서 점수가 깎일 수 있습니다. (동의어 처리 미흡)
- **어순 유연성 부족:** 한국어처럼 어순이 자유로운 언어에서는 점수가 부정확할 수 있습니다.

7 하이브리드 모델 (Hybrid Models)

최근에는 하나의 방식만 고집하지 않고 섞어서 씁니다.

7.1 왜 섞어 쓰는가?

- **NMT(신경망)의 약점:** 가끔 엉뚱한 번역을 하거나 전문 용어를 틀립니다.
- **RBMT/SMT(규칙/통계)의 강점:** 문법이 정확하고 특정 분야 용어 사전을 강제로 적용하기 좋습니다.

7.2 구성 요소 및 장점

- **신경망 + 규칙:** 전체적인 번역은 신경망이 유창하게 하고, 법률 용어나 문법 교정은 규칙 기반 시스템이 마무리합니다.
- **장점:** 유창함(Fluency)과 정확성(Accuracy)을 동시에 잡을 수 있습니다.
- **사례:** SYSTRAN, Microsoft Translator, IBM Watson (의료, 법률 등 특수 분야용).

8 학습 체크리스트 및 FAQ

8.1 학습 완료 체크리스트

RBMT, SMT, NMT의 차이점과 발전 순서를 설명할 수 있는가?

기계 번역의 4가지 난제(모호성, 구조, 관용구, 문맥)를 예시와 함께 말할 수 있는가?

Seq2Seq의 '병목 현상'이 무엇이며, '어텐션'이 이를 어떻게 해결했는지 이해했는가?

트랜스포머가 RNN보다 빠른 이유(병렬 처리)를 아는가?

BLEU 점수의 계산 원리(N-gram 일치)와 한계점(의미 파악 불가)을 아는가?

8.2 FAQ: 자주 묻는 질문

Q. 트랜스포머는 번역에만 쓰이나요? A. 아니요. 트랜스포머는 현재 AI의 표준입니다. 챗GPT 같은 대화형 AI, 요약, 문장 생성, 심지어 코딩 작성까지 모든 자연어 처리에 쓰입니다.

Q. BLEU 점수가 높으면 무조건 좋은 번역인가요? A. 꼭 그렇지는 않습니다. 점수는 높아도 사람이 읽기에 어색할 수 있고, 점수는 낮아도 의미가 완벽하게 통할 수 있습니다. 그래서 사람에 의한 평가도 여전히 중요합니다.

Q. 요즘도 규칙 기반(RBMT)을 쓰나요? A. 순수 RBMT는 거의 안 쓰지만, 하이브리드 모델의 일부로써 문법 교정이나 특정 용어 강제 적용을 위해 여전히 중요한 역할을 합니다.

9 부록: 1페이지 요약 (Cheat Sheet)

Machine Translation 한 장 정리

1. 역사적 흐름 (Evolution)

- **Rule-Based (50s-80s):** 규칙+사전. 정확하지만 확장 불가. (비유: 문법책 보고 작문)
- **Statistical (90s-10s):** 데이터 확률 기반. 유연하지만 문맥 부족. (비유: 단어 맞추기 퍼즐)
- **Neural (10s-Present):** 딥러닝(Seq2Seq, Transformer). 유창하고 문맥 파악. (비유: 전문 통역사)

2. 핵심 모델 (Models)

- **Seq2Seq:** 인코더 → 컨텍스트 벡터 → 디코더. (단점: 정보 병목)
- **Attention:** "필요한 부분만 집중해서 보자". 병목 해결, 긴 문장 OK.
- **Transformer:** "Attention is All You Need". 병렬 처리로 초고속 학습, 현대 AI의 기반.

3. 평가 (Evaluation - BLEU)

- **원리:** 기계 번역과 사람 번역의 N-gram(단어 뭉치) 일치도 측정.
- **특징:** 1.0 만점. 짧으면 페널티(BP). 동의어/의미 파악은 못함.

4. 하이브리드 (Hybrid)

- 신경망의 유창함 + 통계/규칙의 정확성 결합.
- 법률, 의료 등 전문 분야에서 선호됨.