October 26, 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: m번째 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: m번째 문서의 토픽 분포 (K차원 벡터, $\theta_m \sim Dirichlet(\alpha)$)
- ϕ_k: k번째 토픽의 단어 분포 (V차원 벡터, V=어휘 수, $\phi_k \sim Dirichlet(\beta)$)
- $z_{m,n}$: m번째 문서의 n번째 단어에 할당된 토픽 ($z_{m,n} \sim Multinomial(\theta_m)$)
- $w_{m,n}$: m번째 문서의 n번째 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.

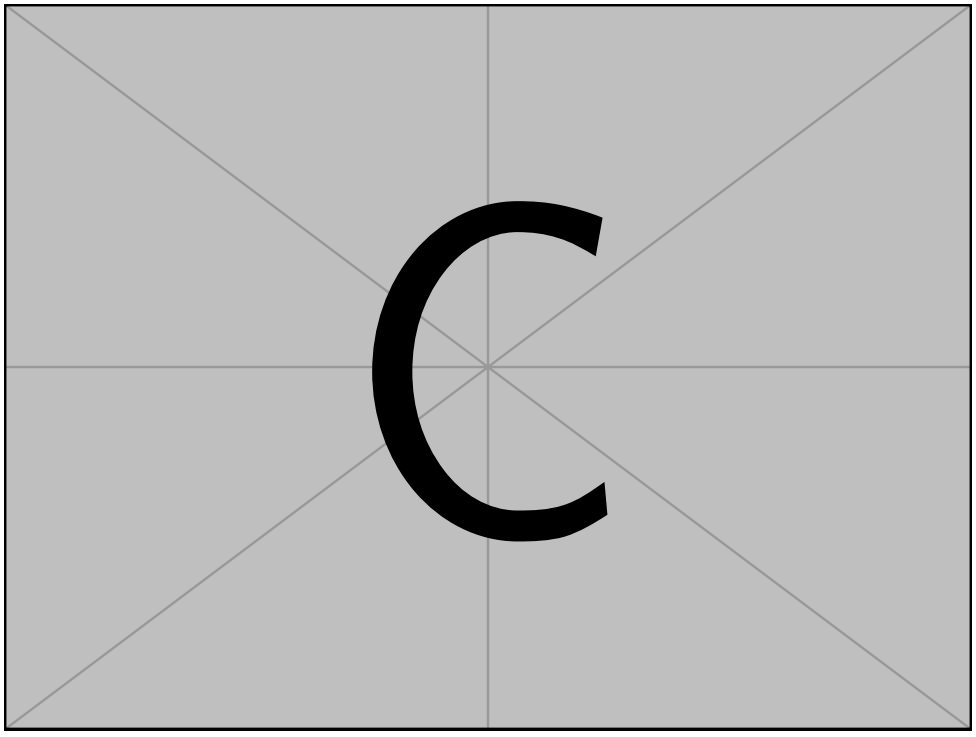

Figure 1: 관찰된 데이터(히스토그램)와 다른 파라미터(*A, B, C*)를 가진 모델

- **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** $\times$ **단어]** 크기의 행렬. (예: m번째 문서의 n번째 단어 빈도수)
- W **(문서-토픽 행렬): [문서** $\times$ **토픽 개수** K**]** 크기의 행렬. (각 문서가 K개의 토픽을 얼마나 포함하는지 나타내는 가중치)
- H **(토픽-단어 행렬): [토픽 개수** K $\times$ **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 $\times$ 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.

```
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

```
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.",
  "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).