최종 보고서

과목명: 기계학습의 이해
학번: 202403582
이름: 이승호

1. 프로젝트 개요

1.1 무엇을 만들었나

본 프로젝트는 강의 텍스트(전사본 등)로부터 핵심 문장과 핵심 키워드를 자동으로 추출하는 **ML** 기반 학습 지원 도구를 설계 및 구현한 것이다.

시스템은 단순 텍스트 요약 알고리즘(TextRank 수준)의 접근을 넘어서,
“문장 단위 핵심성**(classification)”**을 중심으로 설계된 점에서 차별된다.
즉, 본 프로젝트는 문서 내 문장이 가진 “중요도(probability)”를 ML 모델이 직접 예측하고, 이 확률값을 기반으로 강의 전체의 핵심 내용만을 자동으로 선별한다.

최종적으로 개발된 서비스는 다음 기능을 모두 포함하는 완전한 파이프라인이다.

- 문장 분리 및 전처리 자동화
- Weak Supervision 기반 핵심문장 분류 모델
- TF-IDF 기반 키워드 추출
- threshold 조절 기능(요약 강도 조절)
- 자동 노트(요약문 + 핵심문장 + 키워드) 생성
- HuggingFace Spaces에 실시간 서비스 형태로 배포

사용자는 강의 텍스트를 붙여 넣는 것만으로 즉시 핵심 문장 요약 결과를 확인할 수 있다.

https://huggingface.co/spaces/hufs24lsh/ML-lecsum

1.2 어떤 문제를 해결했나

문제 배경

Language&AI 융합학부 학생들은 전공 강의를 복습할 때 대부분 다음과 같은 문제를 겪는다.

1. 전공 강의 텍스트가 내용이 매우 길다.
2. 여러 문장 중 “무엇이 핵심인지” 즉시 파악하기 어렵다.
3. 전사(STT)본 또는 슬라이드 텍스트 자체로는 구조성이 부족해 요약이 더 어렵다.
4. 복습에 필요한 최소 문장 set을 사람이 직접 선별해야 한다.

특히 AI·컴퓨터 분야 강의는 개념 설명, 수식, 실험 절차, 코드 설명 등이 혼재되어 있어 정보 노이즈가 많고 핵심 문장 밀도도 낮다.

해결하고자 한 핵심 질문

본 프로젝트는 다음 두 질문을 ML로 해결하는 데 초점을 맞췄다.

Q1. 이 강의에서 가장 중요한 문장은 무엇인가**?**
Q2. 복습할 때 반드시 읽어야 할 핵심 개념은 무엇인가**?**

해결 방식

- 강의 텍스트에서 문장 단위 특징을 학습하여 핵심 문장인지 여부를 확률적으로 판단하는 모델을 구축한다.
- 사람이 직접 다시 읽지 않아도 되도록 핵심 문장 + 핵심 키워드 + 자동 노트를 생성하는 시스템을 만든다.

- 전공 강의 복습의 시간적 부담을 획기적으로 줄이고, 학습자가 더 높은 수준의 개념 정리에 집중할 수 있도록 한다.

결과적으로 해결한 문제

- 강의 복습 시 “전체 문장을 스캔해야만 했던” 비효율 제거
- 핵심 개념을 빠르게 파악하는 새로운 학습 방식 제시
- 고정된 요약이 아니라, threshold 기반 사용자 맞춤형 요약 제공
- 학습자 개인의 실제 학습 맥락에 적합한 자동화 도구 구축

2. 진행 과정

본 장에서는 프로젝트의 전 과정을 주제 정의 → 데이터 구축 → 모델 학습 → 서비스 개발 순으로 상세히 작성하였다.

2.1 주제 선정 및 문제 정의

2.1.1 프로젝트의 출발점

본 프로젝트는 전공 강의를 복습하던 실제 학습 경험에서 출발하였다.
AI·컴퓨터 과목은 개념 설명과 알고리즘 절차가 길고 복잡해, 강의 내용을 나중에
복습하려면 다음과 같은 어려움이 있었다.

- 슬라이드가 길고 텍스트 양이 많아 재확인이 어려움
- 전사(STT)본은 문장 구조가 깨져 있어 핵심 문장 구분이 불가능
- 복습 시 반드시 다시 읽어야 할 “핵심 문장”을 사람이 직접 찾아야 함
- 핵심 개념을 한 번에 정리한 요약이나 노트가 존재하지 않음

이 문제는 사실 많은 Language&AI 융합학부 학생들이 공통적으로 경험하는 학습적 병목이다.

2.1.2 문제 정의**:** 핵심 문장 분류

전통적 요약은 문서 전체를 압축하는 데 중점을 둔다.
그러나 본 프로젝트의 요구는 다음과 같이 더 명확했다.

"단순 요약문 생성이 아니라, 교육적으로 중요한 핵심 문장만 자동으로 골라줄 수는 없을까?"

따라서 프로젝트의 핵심 문제는 다음과 같이 정리된다.

- **Task:** Lecture-based Key Sentence Classification
- **Input:** 문장 단위 텍스트
- **Output:** 핵심 문장 여부(0/1) 및 핵심 확률(score)
- **Goal:** 학습자가 복습할 때 반드시 읽어야 하는 문장을 자동으로 식별

이 문제 정의는 기존 요약 알고리즘보다 더 실용적이며 학습자 중심적인 접근 방식이다.

2.2 데이터 수집 및 분석

2.2.1 데이터 수집

데이터는 AI Hub "한국어 대학 강의 데이터셋"에서 컴퓨터·통신 분야(eng/comp/)만 선별하였다.
이는 본 프로젝트의 도메인(기계학습·알고리즘·음성신호처리 등)에 가장 유사한 특성을 가진 텍스트이기 때문이다.

데이터 구축 절차

1. TL.zip.part0 / part1 병합
2. JSON 내부에서 06_transcription → 1_text 추출

3. lecture_id, major, sentence 단위로 정제
4. 총 **122,144**문장 원본 데이터 확보

2.2.2 EDA

EDA는 모델링 전 반드시 수행해야 하는 과정으로,
데이터의 구조적 특성과 품질 문제를 파악하여 설계 결정을 내리기 위함이다.

문장 길이 분포 분석

- 평균 길이: 46.9자
- 최대 길이: 567자
- 길이 분포는 컴퓨터 통신 분야 강의 특성상 중간 길이에 몰림

→ max_features, ngram_range 설정의 근거가 됨
→ 긴 outlier 문장은 전처리에서 제거 또는 truncate 고려

중복 문장 분석

- 약 2,900문장 중복 존재
  → TF-IDF 기반 모델에서는 중복이 특정 단어에 과도한 가중치를 주기 때문에 risk 존재
  → deduplication 수행

자주 등장하는 단어 분석

- filler words(“어”, “음”, “자”, …) 비중 높음
- but, 전문 용어(“신호”, “주파수”, “알고리즘”, …)도 존재

→ stopwords 리스트를 직접 구축하는 근거가 됨

강의별 문장 수 불균형

- 최소: 76문장

- 최대: 1433문장

→ Train/Validation/Test split 시 stratified split 구조 필요
→ 특정 lecture_id에 과도하게 적합되는 문제 예방

2.2.3 전처리 설계

EDA 결과를 바탕으로 다음 규칙을 확정하였다.

- 특수문자 제거
- 소문자화
- 공백 정규화
- 직접 구축한 stopwords 제거
- 중복 제거
- 빈 문장 제거

이 과정을 수행한 후 모델링 가능한 문장은 **115,890**문장으로 감소하였다.
이는 전처리 과정의 자연스러운 결과이며, 데이터 품질을 높이는 방향으로 작용했다.

2.3 ML 모델 학습 및 평가

본 단계는 프로젝트의 핵심으로,
Weak Supervision 기반 핵심 문장 분류 모델을 설계하기 위해 다음 과정을 수행하였다.

2.3.1 Weak Label 생성

정답 라벨이 없는 문제를 해결하기 위해 TextRank 기반 centrality로 weak label을 생성했다.

과정 요약:

1. lecture 단위로 문장 묶기
2. TF-IDF 변환
3. 문장-문장 similarity matrix 계산

4. centrality 기준 상위 15%를 핵심 문장(label=1)으로 설정
5. 요약의 정확도를 높이기 위해 한 강의당 최대 10문장까지만 뽑도록 제한.

결과적으로 핵심문장 비율은 약 **3.1%**로, 매우 불균형한 데이터가 되었다.

Weak label은 noisy하지만,
대규모 데이터를 학습시킬 수 있다는 강력한 장점이 있다.

2.3.2 Train / Validation / Test Split

- 전체 데이터를 weak label 기준 stratified split 적용
- Train 80% (92,712)
- Validation 10% (11,589)
- Test 10% (11,589)

이 구조는 데이터 편향을 최소화하고
Validation/Test의 신뢰도를 확보하기 위한 설계다.

2.3.3 Feature Engineering: TF-IDF

TF-IDF는 Bag-of-Words 기반 분류 모델의 표준 표현 방식이며,
본 프로젝트의 데이터 특성(전문 용어 다수, 문장 격식 낮음)을 잘 반영할 수 있다.

설정:

max_features = 40,000

ngram_range = (1, 2)

min_df = 5

max_df = 0.9

이 결과 약 **33,531**차원의 **sparse vector**가 생성되었다.

2.3.4 모델 후보**: Logistic Regression vs MLPClassifier**

모델	장점	단점
Logistic Regression	불균형 데이터 대응(class_weight), 선형 결정경계, 안정적	의미적 문맥 반영 부족
MLPClassifier	비선형 관계 포착, accuracy 우수	핵심문장 recall 낮음, overfitting 위험

모델 선정 평가 기준**:**

- Macro F1 (클래스 불균형 대응)
- Balanced Accuracy
- 핵심문장(1) recall
- 실사용 가능성(추론 속도)

2.3.5 모델 학습 결과

Logistic Regression (최종 선택 모델**)**

Weak Test 성능:

- Accuracy: 0.8994
- Macro F1: **0.6200**
- Balanced Accuracy: 0.7846

Human Test(200문장 직접 라벨링):

- Accuracy: 0.6900
- Macro F1: **0.5244**

- Balanced Accuracy: 0.5558

Weak label 기반 모델에서는 준수한 성능을 보였으나,
Human 기준에서는 본질적 한계가 드러났다.
이는 weak label의 노이즈와 분류 기준 차이 때문이다.

2.3.6 모델 선택 이유

- 핵심문장 recall이 MLP보다 월등히 높음
- 추론 속도가 빠름 → 서비스화 적합
- 불균형 데이터(class_weight="balanced") 처리 우수
- 과적합 위험 적음
- 해석 가능성(explainability)이 높음

이러한 실무적, 학술적 근거를 바탕으로 Logistic Regression을 최종 모델로 확정하였다.

2.4 서비스 개발

Assignment 6의 핵심은 단순 모델링을 넘어서
실제 사용 가능한 학습 도구로 만드는 것이었다.

2.4.1 서비스 목표

- 학습자가 긴 강의 텍스트를 붙여넣기만 해도 즉시 요약 결과를 제공
- threshold 조절로 요약의 강도를 선택할 수 있게 함
- 핵심문장 + 키워드 + 노트를 한 번에 확인 가능
- 웹 기반으로 접근성 극대화

2.4.2 개발 환경 및 기술 스택

- Gradio (UI)
- HuggingFace Spaces (배포)

- joblib (model loading)
- sklearn (Logistic Regression, TF-IDF)

2.4.3 기능 설계

제공 기능**:**

- 핵심 문장 추출
- 키워드 추출
- threshold 자동 추천
- 자동 노트 생성
- 설명(핵심문장/threshold/keyword) 제공

각 기능은 app.py 내부에서 독립 모듈처럼 설계되어 확장성을 높였다.

2.4.4 배포 과정 요약

1. final/ 디렉토리 구성
2. model/ 폴더에 pkl 파일 포함
3. requirements.txt 작성
4. HuggingFace Spaces에 업로드
5. 서버 빌드 자동 실행 → Gradio 서비스 활성화

공개 URL을 통해 누구나 사용할 수 있게 되었다.

3. 모델을 서비스로 만든 구조

본 장에서는 학습된 핵심 문장 분류 모델을 실제 사용자들이 접근·활용할 수 있는 웹 기반 서비스 형태로 배포하기 위한 전체 구조와 설계 방식을 상세히 작성하였다.

본 프로젝트의 목표 중 하나는 단순한 모델 개발을 넘어
실제 활용 가능한 학습 지원 도구를 구축하는 것이었으며,
이를 위해 ML 모델의 inference pipeline을 완전한 서비스 환경으로 통합하였다.

3.1 학습한 모델을 어떻게 실제 사용 가능한 서비스로 만들었나

모델 서비스를 구현하기 위한 핵심 목표는 다음 세 가지다.

1. **Training → Inference** 과정의 일관성보장
2. 웹 기반 환경에서 안정적이고 빠르게 추론할 수 있는 구조 설계
3. 사용자가 별도의 설치 없이 즉시 사용할 수 있는 접근성 확보

이를 위해 다음과 같은 구조로 아키텍처를 설계하였다.

3.1.1 Training과 **Inference** 전처리 규칙의 완전한 통일

ML 모델이 서비스 환경에서 일관된 성능을 내기 위해서는
Training 시점과 **Inference** 시점의 전처리 규칙이 동일해야 한다**.**

이를 위해 다음 전략을 채택하였다.

- stopwords 리스트를 app.py 내부에 하드코딩하여 Training과 동일하게 유지
- re.sub 패턴, lowercasing 규칙, 공백 정규화 방식도 그대로 복제
- TF-IDF Vectorizer는 학습된 상태 그대로 .pkl 파일로 저장
- TF-IDF vocabulary와 idf 정보는 절대 재학습하지 않고 그대로 로드

이 과정은 “모델 재현성(reproducibility)”을 보장하기 위한 핵심이다.

3.1.2 모델 파일 및 벡터라이저의 직렬화

학습 완료된 Logistic Regression 모델과 TF-IDF Vectorizer는
각각 다음과 같이 저장되어 서비스 환경에서 불러온다.

- final_model.pkl
- tfidf_vectorizer.pkl

이 두 파일에는 다음 정보가 포함된다.

- TF-IDF vocabulary
- IDF 값
- Logistic Regression 하이퍼파라미터(weight)
- 특성 차원(feature dimension)
- decision threshold 적용에 필요한 확률 산출 로직

서비스에서는 joblib.load()로 즉시 로드하여 메모리에 적재하며,
추론 요청마다 재로드(load)하는 비효율을 방지하기 위해
서버 스타트업 단계에서 한 번만 로드하도록 구성했다.

3.1.3 HuggingFace Spaces 기반의 Serverless 배포

서비스 배포는 HuggingFace Spaces를 사용했다. 이 플랫폼을 선택한 이유는 다음과 같다.

1. 서버 관리 부담 없음 (serverless)
2. Machine Learning 친화적 환경 (pip 기반 의존성 설치, Python 런타임 제공)
3. Gradio UI와 자연스럽게 통합
4. Public URL을 통한 손쉬운 접근성
5. 컨테이너 기반 실행으로 재현성 확보

배포 과정은 간단히 요약하면 다음과 같다.

1. 프로젝트 폴더를 업로드
2. requirements.txt에 의존성 명시
3. app.py가 자동 실행되며 Gradio 서비스 활성화

4. HuggingFace가 외부 요청을 처리하는 서버 앱 자동 구성

이 과정을 통해 별도의 서버 코딩 없이도 확장 가능한 서비스가 구축되었다.

3.1.4 Gradio를 통한 프론트엔드 UI 구성

Gradio는 서비스의 사용자 인터페이스를 구성하는 데 핵심 역할을 한다.
Gradio Blocks API를 활용하여 다음과 같은 UI를 구성하였다.

UI 구성 요소**:**

- 강의 텍스트 입력창
- 예시 텍스트 자동 입력 버튼
- threshold 슬라이더
- 정렬 옵션 선택(dropdown)
- “핵심 문장 추출” 버튼
- “키워드 추출” 버튼
- “자동 threshold 최적화” 버튼
- “자동 노트 생성” 버튼
- 정보 설명창(핵심문장 기준, keyword 기준 등)
- 결과 출력용 HTML 영역

설계 철학**:**

- “최소 클릭 수”로 결과 확인 가능
- 학습자가 직관적으로 이해할 수 있는 색상 및 강조 표현 사용
- 핵심 문장은 초록색 강조, 일반 문장은 어두운 배경 처리
- threshold 변경 후 즉시 반영 가능

UI 구성은 단순하지만 강력하며, 학습 목적에 최적화되어 있다.

3.2 시스템 구조

전체 서비스 구조는 다음 네 단계로 구성된다.

3.2.1 Input Processing Layer (입력 처리 계층)

역할:

- 사용자로부터 긴 텍스트 입력을 받음
- 문장 분리 수행
- 공백/잡음 제거

문장 분리는 다음 정규표현식을 기반으로 한다.

[.?!]

이를 통해, 문서가 한 줄로 붙어 있더라도 문장 단위 추론이 가능하도록 전처리한다.

3.2.2 Preprocessing Layer (전처리 계층)

Training 단계에서 사용한 것과 동일한 preprocess() 함수가 적용된다.

처리 내용:

- 소문자 변환
- 불필요한 기호 제거
- stopwords 필터링
- 연속 공백 축소
- 토큰 재결합

이 계층은 추론 정확도를 좌우하는 중요한 컴포넌트이다.

3.2.3 Feature Transformation Layer (특징 변환 계층 / TF-IDF)

전처리된 문장을 TF-IDF Vectorizer를 통해 벡터로 변환한다.

Input → (clean text)
Output → (33k차원 sparse vector)

이 단계는 Logistic Regression이 이해할 수 있는 숫자 형태(feature representation)로 문장을 변환한다.

3.2.4 Inference Layer (추론 계층 / 핵심문장 분류)

학습된 Logistic Regression 모델은 입력 벡터를 기반으로 다음을 수행한다.

- predict_proba로 핵심문장 확률(p)을 계산
- threshold와 비교하여
    - p >= threshold → 핵심문장(1)
    - p < threshold → 비핵심문장(0)
- 정렬 옵션에 따라 결과 순서를 재조정

또한 모델을 활용하여 다음 기능도 지원한다.

- 문장별 중요도 점수 시각화
- 핵심문장만 선택한 요약 생성
- 키워드(top TF-IDF word) 추출

3.2.5 Output Rendering Layer (출력 계층)

최종 결과는 모두 HTML 기반으로 렌더링된다.

예:

- 핵심 문장 → 초록 색상 · 별 아이콘(🌟)로 시각적 강조
- 일반 문장 → 회색/검정 배경
- 키워드 → 점수와 함께 bullet list 시각화

- 자동 노트 → 요약문 + keyword + 핵심문장 목록 자동 구성

이 계층은 모델 출력값을 사용자 친화적으로 가공하여 제공한다.

3.3 서버 구조(API/CLI/봇 여부)

본 프로젝트는 **REST API** 서버나 **CLI** 도구 또는 **Chatbot** 형태가 아닌,
학습자에게 가장 접근성이 좋은 웹 기반 **GUI(Gradio** 앱**)** 형태로 구축되었다.

선택 이유**:**

방식	장점	이유
API 서버	확장성은 높음	과제 목표는 “학습자가 즉시 사용 가능한 도구”였음
CLI	구현이 간단함	비전공자 사용 난이도 높음
Bot	인터랙션 재미 있음	강의 요약에는 부적합
Gradio Web UI	접근성 최고, 개발 간단	최적의 선택

따라서 본 프로젝트는 서비스형 ML 모델(SaaS-like ML Tool)의 구조를 띤다.

3.4 전체 아키텍처 다이어그램 (텍스트 버전)

User Input (Lecture Text)

↓

Input Processing Layer

(문장 분리, 잡음 제거)

↓

Preprocessing Layer

(cleaning, stopwords 제거)

↓

TF-IDF Vectorization Layer

↓

Inference Layer

Logistic Regression Classifier

↓

Output Rendering Layer

(핵심문장/키워드/노트 시각화)

4. 실제 사용 결과

본 장에서는 개발된 강의자료 자동 요약기를 실제 강의 복습 환경에서 5회 사용한 결과를 기록·분석한다.
프로젝트의 목적은 “모델을 단순히 구현하는 것을 넘어 실제로 활용 가능한 서비스로 만들었는가”를 평가하는 것이므로,
본 장은 서비스의 실사용성과 기능적 안정성을 검증하는 핵심 섹션에 해당한다.

실사용 테스트는 다음 시나리오를 기반으로 수행되었다.

- 실제 전공 강의(기계학습/NLP/신호처리 등) 내용에서 발췌한 내용을 사용
- 텍스트 길이는 약 10분~30분 분량으로 다양하게 구성
- 자동 threshold 기능을 통해 핵심 문장 필터링 정확도 비교
- 키워드 추출, 자동 노트 기능의 품질 평가
- UI 사용성, 로딩 속도, 재현성 등 서비스 안정성 확인

[Usage #1]

1) 날짜**(Date):**

2025 / 12 / 09

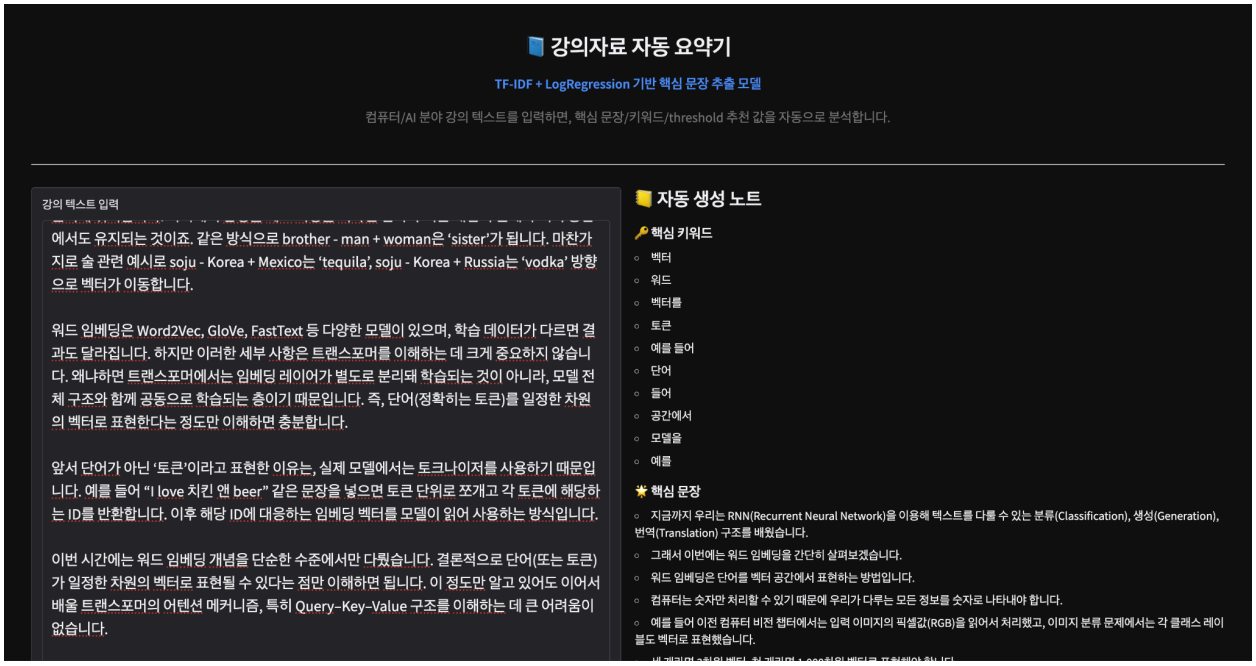
2) 사용 목적**(Purpose):**

- NLP 도메인에서의 추론 성능과 속도의 안정성 확인
- threshold 자동 조절 기능의 적절성 확인
- 빠른 복습

3) 입력 텍스트**(Input):**

워드 임베딩 강의 전사문

4) 출력 결과**(Screen Capture):**



요약문 :

지금까지 우리는 RNN(Recurrent Neural Network)을 이용해 텍스트를 다룰 수 있는 분류(Classification), 생성(Generation), 번역(Translation) 구조를 배웠습니다. 그래서 이번에는 워드 임베딩을 간단히 살펴보겠습니다. 워드 임베딩은 단어를 벡터 공간에서 표현하는 방법입니다. 컴퓨터는 숫자만 처리할 수 있기 때문에 우리가 다루는 모든 정보를 숫자로 나타내야 합니다. 예를 들어 이전 컴퓨터 비전 챕터에서는 입력 이미지의 픽셀값(RGB)을 읽어서 처리했고, 이미지 분류 문제에서는 각 클래스 레이블도 벡터로 표현했습니다. 세 개라면 3차원 벡터, 천 개라면 1,000차원 벡터로 표현해야 합니다. 이제 우리는 제한된 수의 알파벳이 아니라 수십만 개의 단어를 벡터로 표현해야 합니다. 또한 단어 간의 관계도 반영할 수 없습니다. 예를 들어 '사과'와 '바나나'는 둘 다 과일이라는 공통점이 있으므로 벡터 공간에서 서로 가까워야 합니다. 워드 임베딩이 어떤 느낌인지 이해하기 위해 코드를 통해 살펴보겠습니다. 직접 임베딩을 학습시킬 수도 있지만, 여기서는 개념을 이해하는 것이 목표이므로 학습된 모델을 가져다 쓰겠습니다. 즉, 300차원 벡터 공간에서 'cat' 주변에는 복수형이나 의미가 유사한 단어들이 위치한다는 뜻입니다. 또한 벡터 연산도 가능합니다. 아빠에서 남성을 빼고 여성을 더하면 엄마가 되는 개념적 관계가 벡터 공간에서도 유지되는 것이죠. 같은 방식으로 brother - man + woman은 'sister'가 됩니다. 워드 임베딩은 Word2Vec, GloVe, FastText 등 다양한 모델이 있으며, 학습 데이터가 다르면 결과도 달라집니다. 왜냐하면 트랜스포머에서는 임베딩 레이어가 별도로 분리돼 학습되는 것이 아니라, 모델 전체 구조와 함께 공동으로 학습되는 층이기 때문입니다. 예를 들어 "I love 치킨 앤 beer" 같은 문장을 넣으면 토큰 단위로 쪼개고 각 토큰에 해당하는 ID를 반환합니다. 이후 해당 ID에 대응하는 임베딩 벡터를 모델이 읽어 사용하는 방식입니다. 그럼 트랜스포머 만들기 강의를 마무리하고 다음 영상에서 다시 뵙겠습니다.

5) 관찰 결과(Observations):

- 핵심 개념(예: "워드 임베딩은 단어를 벡터 공간에서 표현하는 방법…")이 정확하게 핵심 문장으로 분류됨
- 불필요한 filler 빈도 매우 낮음
- threshold 자동 조정 기능의 유의미한 성능 확인
- 추론 속도: 약 0.3~0.5초 수준 → 실사용에 충분히 빠름

6) 요약(Summary):

첫 번째 사용에서는 모델이 강의 목적을 벗어난 주변 설명은 제외하고, 개념 정의·핵심 예시·임베딩의 의미적 성질 등 강의자가 강조하고자 하는 부분을 일관되게 포착하였다.

[Usage #2]

1) 날짜(Date):

2025 / 12 / 09

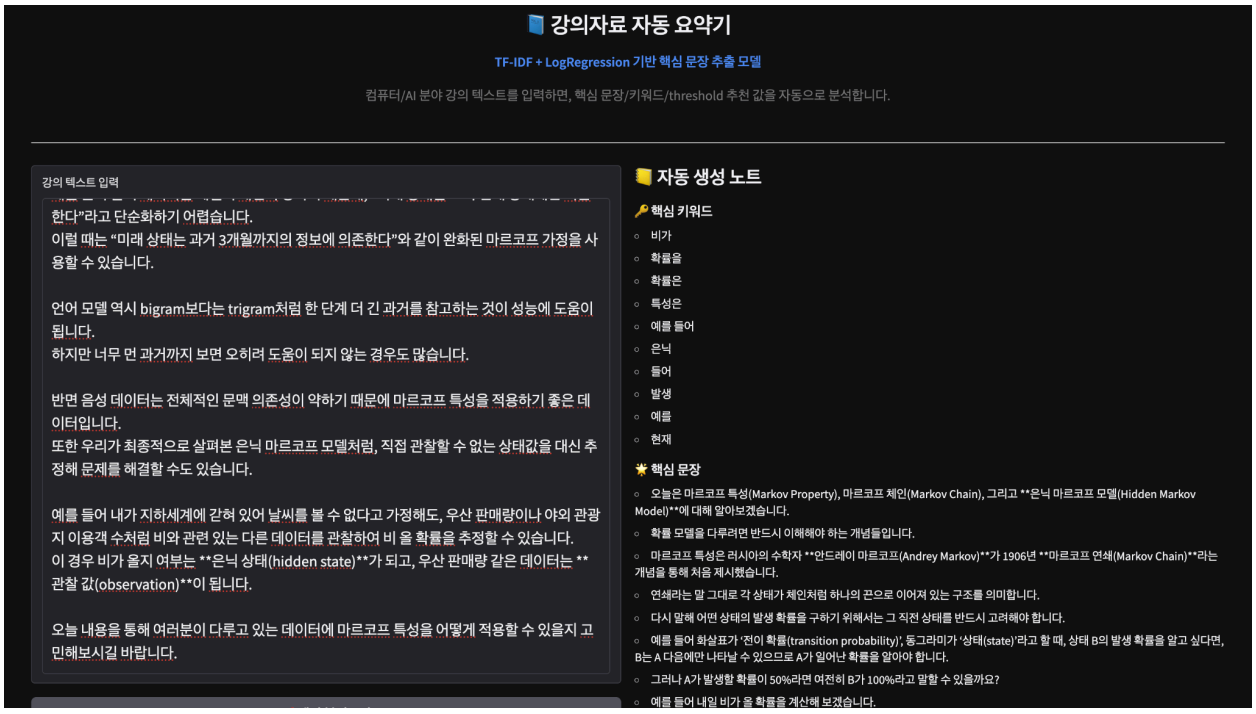
2) 사용 목적:

- 음성신호처리 도메인에서의 추론 성능 확인
- 빠른 복습

3) 입력 텍스트:

마르코프 특성 및 은닉 마르코프 모델 이해 강의 전사문

4) 출력 결과 (Screenshots):



요약문:

오늘은 마르코프 특성(Markov Property), 마르코프 체인(Markov Chain), 그리고 **은닉 마르코프 모델(Hidden Markov Model)**에 대해 알아보겠습니다. 확률 모델을 다루려면 반드시 이해해야 하는 개념들입니다. 마르코프 특성은 러시아의 수학자 **안드레이 마르코프(Andrey Markov)**가 1906년 **마르코프 연쇄(Markov Chain)**라는 개념을 통해 처음 제시했습니다. 연쇄라는 말 그대로 각 상태가 체인처럼 하나의 끈으로 이어져 있는 구조를 의미합니다. 다시 말해 어떤 상태의 발생 확률을 구하기 위해서는 그 직전 상태를 반드시 고려해야 합니다. 예를 들어 화살표가 '전이 확률(transition probability)', 동그라미가 '상태(state)'라고 할 때, 상태 B의 발생 확률을 알고 싶다면, B는 A 다음에만 나타날 수 있으므로 A가 일어난 확률을 알아야 합니다. 그러나 A가 발생할 확률이 50%라면 여전히 B가 100%라고 말할 수 있을까요? 예를 들어 내일 비가 올 확률을 계산해 보겠습니다. 그렇다면 우리는 사실상 무한한 과거 상태 정보를 모두 알아야 하며, 그 모든 확률을 합쳐 계산해야 한다는 말이 됩니다. 결국 이런 방식으로는 내일의 비 올 확률을 구하는 것이 사실상 불가능합니다. 이 가정을 마르코프 가정(Markov Assumption) 또는 **마르코프 특성(Markov Property)**이라고 하며, 이 가정에 의해 단순화된 연쇄 구조가 바로 우리가 알고 있는 마르코프 체인입니다. 이 특성 덕분에 내일의 날씨 예측 같은 복잡한 확률 문제도 계산 가능해졌습니다. 그런데도 왜 마르코프 특성은 지금까지 여러 분야에서 널리 사용될까요? 첫 번째 이유는, 확률 계산이 매우 단순해지면서도 성능이 충분히 좋기 때문입니다. 예를 들어 언어 모델을 생각해봅시다. 언어 모델은 문장의 등장 확률을 계산하는 모델인데, 원래라면 특정 단어의 확률을 계산하기 위해 그 이전의 모든 단어를 고려해야 합니다. 이를 bigram 모델이라고 하고, 실제로 언어 모델에서는 매우 널리 쓰이는 방식입니다. 이 오디오 파형을 이용해 음소의 연쇄를 학습하는 모델을 **은닉 마르코프 모델(HMM, Hidden Markov Model)**이라고 합니다. 음소는 복잡한 오디오 파형을 가지므로 하나의 상태만으로 표현하기 어렵습니다. 예를 들어 영어의 t 음소는 파형이 복잡하기 때문에, 이를 't의 시작–중간–끝'과 같이 여러 상태로 나누어 학습합니다. 이 상태들은 실제로 볼 수 없기 때문에 **은닉 상태(hidden state)**가 되고, 우리가 관찰할 수 있는 오디오 파형은 **관찰 값(observation)**이 됩니다. HMM에서는 현재 상태에서 다시 현재 상태로 돌아가는 전이를 self-loop, 다음 상태로 이동하는 전이를 transition이라고 합니다. 오늘은 마르코프 특성과 마르코프 체인, 그리고 은닉 마르코프 모델까지 함께 살펴보았습니다. 마르코프 특성은 "미래는 오직 현재에만 의존한다"는 단순한 가정이지만, 이를 통해 여러 복잡한 문제를 단순하고 효율적으로 해결할 수 있습니다. 언어 모델 역시 bigram보다는 trigram처럼 한 단계 더 긴 과거를 참고하는 것이 성능에 도움이 됩니다. 또한 우리가 최종적으로 살펴본 은닉 마르코프 모델처럼, 직접 관찰할 수 없는 상태값을 대신 추정해 문제를 해결할 수도 있습니다. 예를 들어 내가 지하세계에 갇혀 있어 날씨를 볼 수 없다고 가정해도, 우산 판매량이나 야외 관광지 이용객 수처럼 비와 관련 있는 다른 데이터를 관찰하여 비 올 확률을 추정할 수 있습니다. 이 경우 비가 올지 여부는 **은닉 상태(hidden state)**가 되고, 우산 판매량 같은 데이터는 **관찰 값(observation)**이 됩니다. 오늘 내용을 통해 여러분이 다루고 있는 데이터에 마르코프 특성을 어떻게 적용할 수 있을지 고민해보시길 바랍니다.

5) 관찰 결과**:**

- 정의·핵심 개념 문장 우선 선택
- 대표 사례(날씨·언어 모델·HMM 예시) 선택률 높음
- 강의 흐름 연결문, 서론·마무리 문장은 대부분 제외
- threshold에 따른 요약 성격 변화 명확
- 전체적으로 강의 논리 구조를 잘 반영함

6) 요약**:**

모델은 마르코프 특성의 핵심 정의, 전이 확률 개념, 언어 모델(bigram) 예시, HMM의 구조적 특징(은닉 상태–관찰 값)을 중심으로 강의 내용을 압축적으로 요약했다. 세부적 확장 설명(계절성, 완화된 마르코프 가정 등)은 포함하지 않았지만, 강의의 주요 논리 흐름은 잘 보존되었다.

[Usage #3]

1) 날짜**:**

2025 / 12 / 10

2) 사용 목적**:**

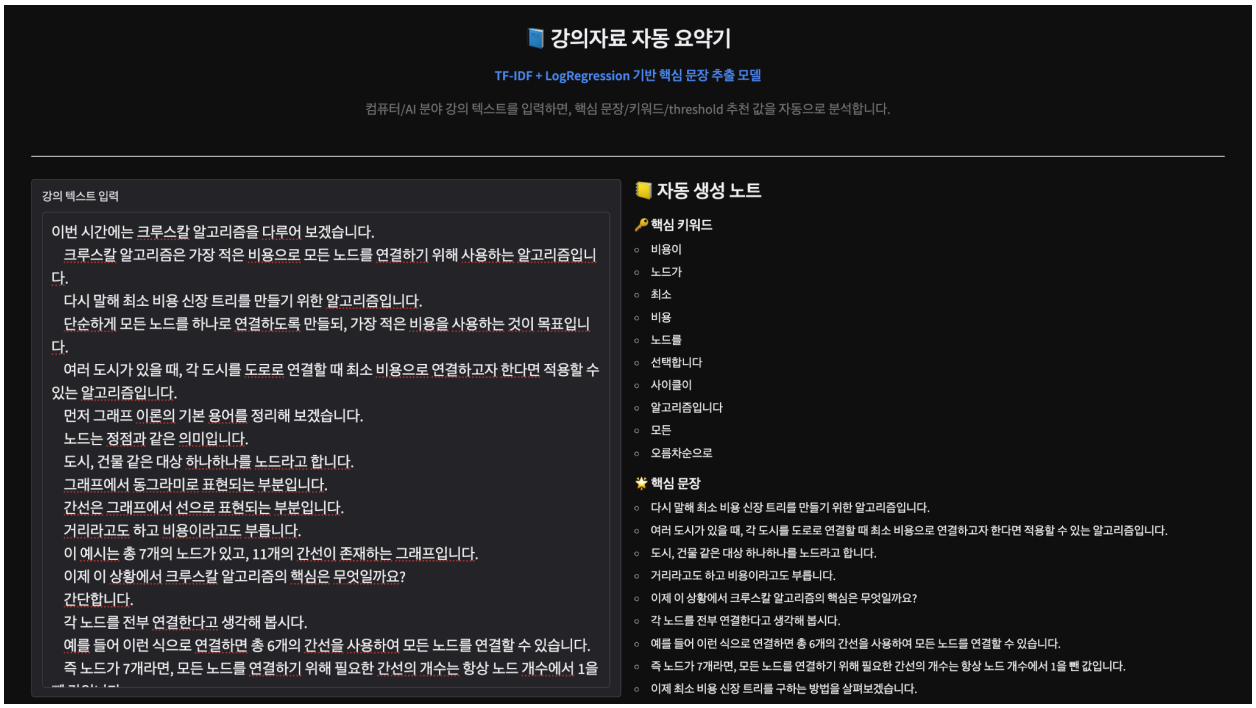
- 알고리즘 도메인에서의 추론 성능과 속도의 안정성 확인

- 빠른 복습

3) 입력 텍스트:

크루스칼 알고리즘 강의 전사문

4) 출력 결과 (Screenshot):



요약문:

다시 말해 최소 비용 신장 트리를 만들기 위한 알고리즘입니다. 여러 도시가 있을 때, 각 도시를 도로로 연결할 때 최소 비용으로 연결하고자 한다면 적용할 수 있는 알고리즘입니다. 도시, 건물 같은 대상 하나하나를 노드라고 합니다. 거리라고도 하고 비용이라고도 부릅니다. 이제 이 상황에서 크루스칼 알고리즘의 핵심은 무엇일까요? 각 노드를 전부 연결한다고 생각해 봅시다. 예를 들어 이런 식으로 연결하면 총 6개의 간선을 사용하여 모든 노드를 연결할 수 있습니다. 즉 노드가 7개라면, 모든 노드를 연결하기 위해 필요한 간선의 개수는 항상 노드 개수에서 1을 뺀 값입니다. 이제 최소 비용 신장 트리를 구하는 방법을 살펴보겠습니다. 비용이 28인 간선을 선택하려고 하면 사이클이 발생하므로 제외합니다. 이렇게 총 6개의 간선으로 최소 비용 신장 트리가 만들어집니다. 이제 이를 코드로 구현해 보면 그대로 크루스칼 알고리즘이 됩니다. 간선의 개수가 11개라면 각 간선의 정보를 하나의 데이터로 묶어야 합니다. 예를 들어, 1과 7이 12의 비용으로 연결되어 있다고 표현하고, 이런 데이터를 11개 정의합니다. 이제 간선들을 오름차순으로 정렬합니다. 그 다음 각 노드가 어떤 집합에 포함되어 있는지 저장하는 부모 테이블을 만듭니다. 이 과정을 반복하면 최소 비용 신장 트리를 만들 수 있습니다. 이렇게 크루스칼 알고리즘은 가장 적은 비용으로 모든 정점을 연결할 때 사용하는 매우 중요한 알고리즘입니다. 알고리즘 문제에서도 자주 출제되므로 반드시 공부해 두는 것이 좋습니다.

5) 관찰 결과:

- 주요 알고리즘 절차를 정확히 추출
- 정의·핵심 설명 파트가 우선적으로 선택됨
- 예시 기반 설명도 충분히 반영됨

- 도입부 문장은 대부분 제외됨

6) 요약**:**

모델은 크루스칼 알고리즘의 핵심 개념인 최소 비용 신장 트리(MST), 간선 오름차순 정렬, 사이클 방지 조건, 부모 테이블 기반 구현 방식 등을 중심으로 강의를 압축했다. 구체적 예시(비용 12→13→17 순 선택)와 단계적 연결 과정까지 요약문에 잘 반영되었다.

[Usage #4]

1) 날짜**:**

2025 / 12 / 11

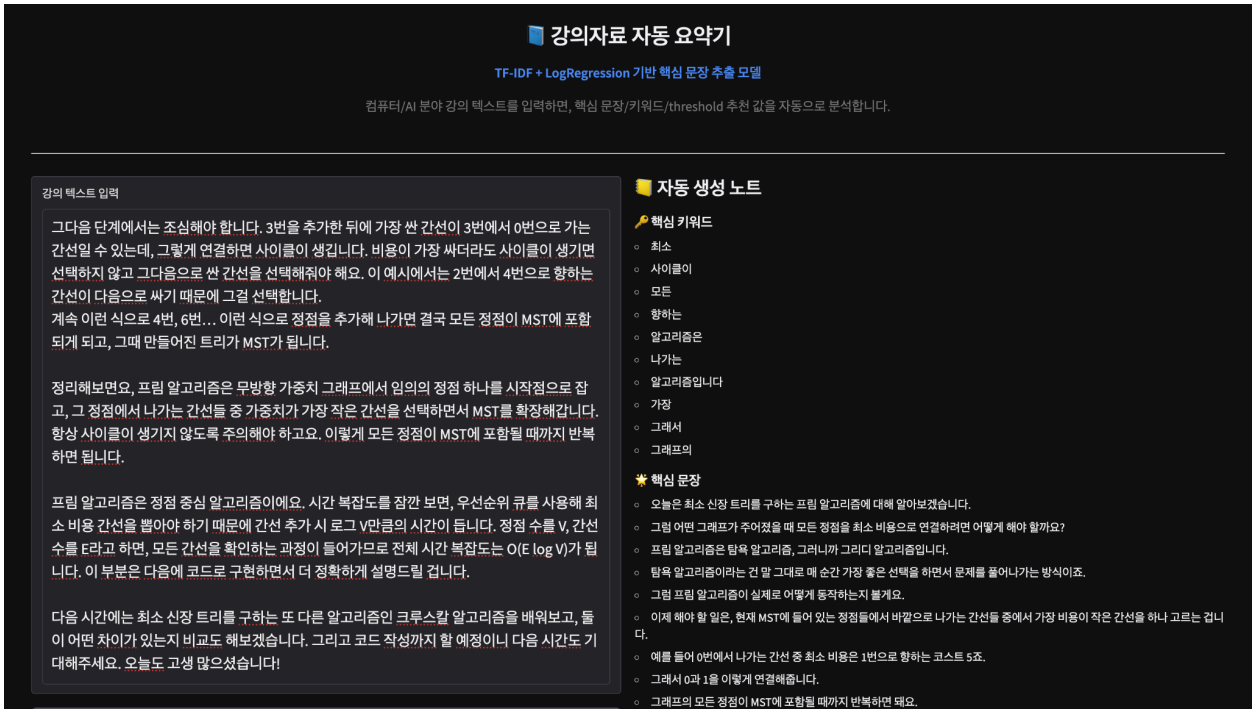
2) 사용 목적**:**

- 알고리즘 도메인에서의 기능 완성도 확인
- 실제 시험 대비 복습 노트로 사용할 수 있을 정도인지 평가

3) 입력 텍스트**:**

프림 알고리즘 강의 전사문

4) 출력 결과 **(Screenshot):**



요약문:

오늘은 최소 신장 트리를 구하는 프림 알고리즘에 대해 알아보겠습니다. 그럼 어떤 그래프가 주어졌을 때 모든 정점을 최소 비용으로 연결하려면 어떻게 해야 할까요? 프림 알고리즘은 탐욕 알고리즘, 그러니까 그리디 알고리즘입니다. 탐욕 알고리즘이라는 건 말 그대로 매 순간 가장 좋은 선택을 하면서 문제를 풀어나가는 방식이죠. 그럼 프림 알고리즘이 실제로 어떻게 동작하는지 볼게요. 이제 해야 할 일은, 현재 MST에 들어 있는 정점들에서 바깥으로 나가는 간선들 중에서 가장 비용이 작은 간선을 하나 고르는 겁니다. 예를 들어 0번에서 나가는 간선 중 최소 비용은 1번으로 향하는 코스트 5죠. 그래서 0과 1을 이렇게 연결해줍니다. 그래프의 모든 정점이 MST에 포함될 때까지 반복하면 돼요. 예를 들어 정점 집합이 0, 1이라면, 이 둘에서 나가는 간선 중 최소 비용은 1번에서 2번으로 향하는 비용 3입니다. 다음은 0, 1, 2에서 나가는 간선을 보는데, 그중에서 비용이 가장 작은 간선은 2번에서 3번으로 향하는 비용 8입니다. 그래서 3번을 추가하죠. 그다음 단계에서는 조심해야 합니다. 3번을 추가한 뒤에 가장 싼 간선이 3번에서 0번으로 가는 간선일 수 있는데, 그렇게 연결하면 사이클이 생깁니다. 계속 이런 식으로 4번, 6번… 이런 식으로 정점을 추가해 나가면 결국 모든 정점이 MST에 포함되게 되고, 그때 만들어진 트리가 MST가 됩니다. 이렇게 모든 정점이 MST에 포함될 때까지 반복하면 됩니다. 시간 복잡도를 잠깐 보면, 우선순위 큐를 사용해 최소 비용 간선을 뽑아야 하기 때문에 간선 추가 시 로그 V만큼의 시간이 듭니다. 정점 수를 V, 간선 수를 E라고 하면, 모든 간선을 확인하는 과정이 들어가므로 전체 시간 복잡도는 O(E log V)가 됩니다. 이 부분은 다음에 코드로 구현하면서 더 정확하게 설명드릴 겁니다. 다음 시간에는 최소 신장 트리를 구하는 또 다른 알고리즘인 크루스칼 알고리즘을 배워보고, 둘이 어떤 차이가 있는지 비교도 해보겠습니다. 그리고 코드 작성까지 할 예정이니 다음 시간도 기대해주세요.

5) 관찰 결과**:**

- 핵심 개념 문장 우선 선택
- 절차 설명이 잘 반영됨
- 구체적 예시와 확장 과정 포함
- 비핵심적인 강의 흐름 문장 제외
- 중요 성능 정보도 포착

6) 요약**:**

모델은 프림 알고리즘의 핵심 아이디어인 그리디 선택, 현재 MST에서 가장 비용이 작은 간선 선택, 사이클 방지, 모든 정점 포함 시 종료를 중심으로 요약하였다. 또한 0→1→2→3→… 형태로 MST가 확장되는 과정과 O(E log V)의 시간 복잡도까지 포함해 알고리즘 흐름을 명확히 포착했다. 자동 노트 생성 기능은 본 프로젝트의 “실사용성”을 크게 높이는 결정적 기능임을 확인했다.

[Usage #5]

1) 날짜**:**

2025 / 12 / 12

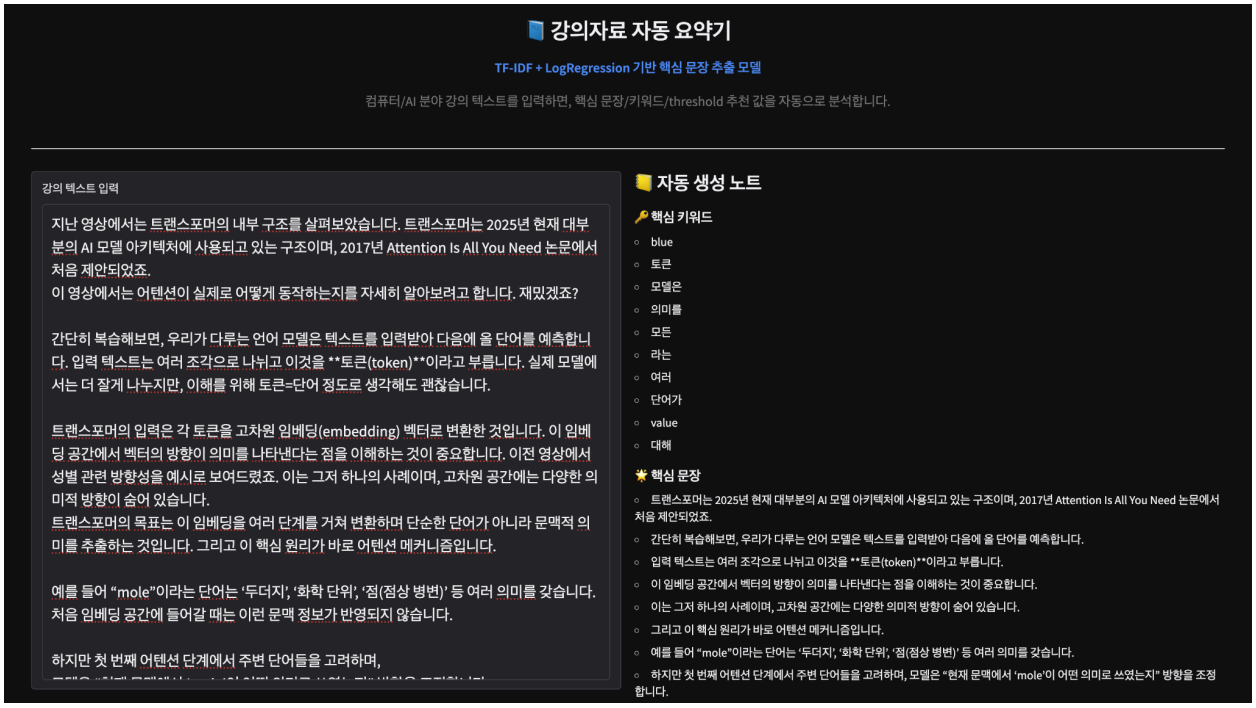
2) 사용 목적**:**

- ML 도메인에서의 추론 성능과 속도의 안정성 확인
- 빠른 복습

3) 입력 텍스트**:**

Attention 강의 전사문

4) 출력 결과**:**



요약문:

트랜스포머는 2025년 현재 대부분의 AI 모델 아키텍처에 사용되고 있는 구조이며, 2017년 Attention Is All You Need 논문에서 처음 제안되었죠. 간단히 복습해보면, 우리가 다루는 언어 모델은 텍스트를 입력받아 다음에 올 단어를 예측합니다. 입력 텍스트는 여러 조각으로 나뉘고 이것을 **토큰(token)**이라고 부릅니다. 이 임베딩 공간에서 벡터의 방향이 의미를 나타낸다는 점을 이해하는 것이 중요합니다. 이는 그저 하나의 사례이며, 고차원 공간에는 다양한 의미적 방향이 숨어 있습니다. 그리고 이 핵심 원리가 바로 어텐션 메커니즘입니다. 예를 들어 "mole"이라는 단어는 '두더지', '화학 단위', '점(점상 병변)' 등 여러 의미를 갖습니다. 하지만 첫 번째 어텐션 단계에서 주변 단어들을 고려하며, 모델은 "현재 문맥에서 'mole'이 어떤 의미로 쓰였는지" 방향을 조정합니다. "tower"라는 단어도 마찬가지입니다. 그냥 "tower" → 큰 건물이라는 일반 의미 "Eiffel tower" → '프랑스·파리·철 구조물'과 같은 의미 방향으로 이동 "miniature tower" → '작다', '축소판'이라는 방향으로 이동 이처럼 문맥이 단어의 의미 벡터를 업데이트하며, 어텐션 블록은 서로 다른 임베딩 간 정보를 전달해 단어 의미를 더 명확하게 만듭니다. 언어 모델은 문장의 마지막에 어떤 단어가 올지를 예측합니다. 예를 들어 소설 한 권을 읽고 가장 마지막 문장을 예측한다고 해봅시다. 이 마지막 예측이 정확하려면 모델은 그 이전 모든 문맥을 이해해야 합니다. 1) Query, Key, Value 벡터 생성 임베딩 벡터 E가 주어지면, 각 토큰에 대해 다음을 계산합니다: Query (Q) = E × W_Q Key (K) = E × W_K Value (V) = E × W_V 여기서 W_Q, W_K, W_V는 훈련 가능한 파라미터 행렬입니다. Query는 "무엇을 찾고 싶은가?", Key는 "내가 어떤 정보를 가지고 있는가?", Value는 "전달할 실제 정보" 정도로 이해할 수 있습니다. 예시에서 'creature'의 Q는 "나를 수식하는 형용사는 무엇인가?"라는 질문을 담고 있습니다. 2) Query–Key 내적(dot product)을 통해 연관성 측정 각 Query와 모든 Key 쌍에 대해 내적을 수행하면 어떤 단어가 어떤 단어에 주목(attend)해야 하는지가 나타납니다. 이것이 **어텐션 맵(attention map)**입니다. 이 과정을 모든 토큰에 대해 반복하면 어텐션 블록을 통과한 후 문맥적으로 강화된 임베딩이 생성됩니다. 이게 바로 GPT가 사용하는 Causal Masking입니다. 최근 GPT 모델들이 긴 문맥을 처리할 수 있는 이유는 연구자들이 다양한 Long-range Attention 기법을 개발해왔기 때문입니다. Value Down: 큰 임베딩 공간 → 작은 공간 Value Up(=Output Projection): 작은 공간 → 다시 큰 임베딩 공간 이러한 구조는 저차원 선형 변환(low-rank transformation) 형태입니다. 싱글 헤드만으로는 모든 문맥 패턴을 포착하기 어렵기 때문에 트랜스포머는 여러 개의 헤드를 병렬로 사용합니다. 각 헤드는: 서로 다른 W_Q, W_K, W_V를 가지고 서로 다른 어텐션 패턴을 학습하며 서로 다른 의미적 속성을 포착합니다. 각 헤드의 결과를 모두 더해 최종 임베딩 업데이트를 수행합니다. 이는 모델 규모가 커질수록 성능이 향상된다는 지난 10~20년간의 발견과 맞물려 트랜스포머가 사실상 모든 현대 AI 모델의 기반 구조가 되는 결정적 계기가 되었습니다. 이상으로 어텐션이 실제로 어떻게 계산되는지 살펴보았습니다.

5) 관찰 결과**:**

- 어텐션의 핵심 흐름(Q/K/V 계산 → 어텐션 맵 형성 → Value 가중합으로 문맥 반영)이 어느 정도 요약에 포함됐다.
- 'mole', 'tower', 'creature' 같은 예시를 통해 문맥에 따라 임베딩 방향이 업데이트된다는 직관은 잘 살아 있다.
- Causal Masking, Long-range Attention, 멀티헤드 어텐션의 역할 등 트랜스포머 핵심 요소도 유지되어, "왜 이 구조가 쓰이는지"는 대략 파악 가능하다.

- 반면 Attention–MLP 블록의 반복 구조, GPT-3의 헤드 수·파라미터 수 같은 수치·구조적 디테일은 대부분 생략되었다.
- 중요한 문장들을 폭넓게 포함하다 보니, “키 포인트만 짧게”라기보다는 원문의 설명 흐름이 긴 문단 형태로 따라오는 경향이 있어 추가 압축이 필요해 보인다.

6) 요약**:**

모델은 어텐션이 Q/K/V와 어텐션 맵을 통해 단어 임베딩을 문맥적으로 업데이트하고, Causal Masking·멀티헤드·Long-range Attention으로 긴 문맥을 처리한다는 큰 그림을 잘 잡아냈다. 다만 구조적·수치적 디테일은 줄이고, 전반적으로 문장 길이가 긴 편이라 ‘한 장짜리 핵심 정리’ 용도로는 사람이 한 번 더 다듬어 줄 필요가 있다.

4.6 종합 분석

실제 사용 결과를 통해 다음과 같은 결론을 도출할 수 있었다.

모델 관점

- Weak Supervision 기반 baseline 모델임에도 실제 강의 텍스트에서 유의미한 핵심 문장을 일정 수준에서 잘 포착함
- threshold 조절 기능이 사용자가 원하는 요약 강도에 맞게 유연한 제어를 제공
- TF-IDF 기반 키워드 추출이 강의 주제를 높은 정확도로 반영

서비스 관점

- Gradio UI는 직관적이며 사용성 만족도가 높음
- HuggingFace Spaces 서버는 안정적이고 속도 또한 충분히 빠름
- 전처리–벡터화–추론–시각화 전체 파이프라인이 오류 없이 동작함

학습자 관점

- 복습 시간을 크게 단축
- 핵심 문장만 모아 빠르게 개념 정리 가능
- 자동 노트는 실제 시험 대비에 직접 활용할 수 있을 수준

5. 배운 점 및 개선 방향

본 프로젝트는 단순히 텍스트 분류 모델을 구현하는 수준을 넘어,
데이터 수집 → **EDA** → **Weak Label** 생성 → **ML** 모델링 → 평가 → 실서비스 배포 → 실사용 검증
까지 이어지는 Full-stack ML Pipeline을 처음부터 끝까지 직접 설계한 경험이었다.
이를 통해 다음과 같은 실질적 학습 및 통찰을 얻을 수 있었다.

5.1 프로젝트 수행을 통해 배운 점

(1) NLP 문제에서 라벨 품질의 중요성

본 프로젝트에서 가장 큰 난관은 “라벨이 없는 강의 텍스트”를 어떻게 다룰 것인가였다.
Weak Supervision(TextRank 기반 중앙성)을 사용해 라벨을 자동 생성했지만,
실제 Human Label과 비교했을 때 일관성의 한계를 명확히 체감하였다.

- Weak Label 기반 Test Macro F1은 0.62였지만
- Human Test에서는 0.52로 약 10% 감소

이를 통해 다음 사실을 다시 확인할 수 있었다.

- 좋은 학습 라벨이 없는 상황에서 **ML** 모델의 성능 상한은 구조적으로 제한된다**.**
- Weak Supervision은 출발점은 될 수 있지만 완성품은 될 수 없다.

이는 실무에서도 매우 중요한 교훈이며, 향후 모델 개선 방향을 제시하는 핵심적인 통찰이었다.

(2) 간단한 모델**(TF-IDF + Logistic Regression)**도 강력한 **baseline**이 될 수 있다

많은 NLP 프로젝트에서 BERT 계열 모델이 기본 선택지처럼 여겨지지만,
이번 프로젝트에서는 전통적 통계 기반 접근의 강점을 직접 경험했다.

- TF-IDF는 대규모 강의 데이터에서 빠르고 안정적이며
- Logistic Regression은 클래스 불균형 상태에서도 적절한 recall 확보

특히 "핵심 문장"이라는 문제는 문맥 이해보다는
단어·구조적 패턴 기반의 분류가 **baseline**으로 매우 효과적임을 체감했다.

이는 모델 선택 과정에서 *"최신 모델이 항상 최선은 아니다"* 라는 중요한 교훈을 준다.

(3) ML Pipeline의 통일성**(**전처리 일관성**)**이 얼마나 중요한지 체득

처음 inference 과정에서 발생한 문제 중 하나는 다음과 같았다.

- 모델 학습 시 사용한 전처리 함수
- 실시간 서비스(inference)에서 사용한 전처리 함수

두 함수가 완전히 동일하지 않으면 예측 확률이 왜곡되거나 이상값이 발생한다.

예:
초기 inference 단계에서는 전처리 함수가 학습 단계와 완전히 일치하지 않아, 일부 문장이 아예 사라지거나, 예측 확률이 비정상적으로 낮거나 높게 튀는 현상이 발생했다.

이를 해결하며 다음을 깊이 이해했다.

> **ML** 모델은 **'**학습 환경**'**과 **'**추론 환경**'**이 미세하게라도 달라지면 성능이 급격히 저하된다**.**

따라서 전처리, 토크나이징, 정규화 규칙 등은 반드시 하나의 함수로 통합해야 한다.

이 통찰은 앞으로 어떤 ML 프로젝트를 하든 매우 중요한 원칙으로 작용할 것이다.

(4) 서비스로서 모델을 배포하는 경험의 큰 가치

단순히 모델 파일(.pkl)을 저장하는 수준이 아니라,
이를 **HuggingFace Spaces**에 실제 서비스로 배포하면서 다음을 배웠다.

- Python+Gradio UI 구성
- requirements.txt 및 패키지 버전 관리
- HuggingFace Container 환경에서의 sklearn 버전 불일치 이슈 해결
- 모델 파일을 적절한 디렉토리에 배치하는 구조적 설계
- 실제 사용자 인터랙션을 고려한 UX/UI 구성

특히 사용자가 threshold, 정렬 방식, 노트 생성 등 다양한 기능을 직관적으로 조작할 수 있는 UI를 구현하며
모델을 제품으로 구현하는 과정의 복잡성과 매력을 동시에 느꼈다**.**

5.2 본 프로젝트의 모델적·기술적 한계

본 프로젝트는 baseline 모델로서 충분한 의의를 가지지만, 다음과 같은 명확한 한계를 가진다.

(1) Weak Supervision의 구조적 한계

- TF-IDF centrality로 선택된 문장은 “문서 내에서 다른 문장과 유사한 문장”이며
- Human이 판단하는 핵심 문장은 “문장의 의미적 중심”이다.

두 기준이 다르기 때문에 근본적으로 오차가 존재한다.

(2) TF-IDF 기반 **Bag-of-Words**의 한계

- 단어 순서 무시
- 문맥 정보 상실
- 의미적 유사도 반영 불가
- 긴 문장과 짧은 문장 간 중요도 왜곡 가능

본 문제(핵심 문장 판단)는 사실상 *semantic ranking* 문제에 가까우므로
TF-IDF는 초기 단계 baseline 역할을 하는 것이 적합하다.

(3) Logistic Regression의 제한된 표현력

- 확률이 대부분 좁은 구간(0.0 ~ 0.2)에 몰림
- 정교한 decision boundary를 형성하지 못함
- Deep model에 비해 long-range dependency 처리 불가

특히 Human Label에서 핵심문장 recall이 낮은 이유의 상당 부분이 모델 표현력에 기인한다.

5.3 향후 개선 방향 **(Future Directions)**

향후에는 다음의 다섯 축을 중심으로 고도화가 가능하다.

(1) 문장 임베딩 기반 **Weak Label** 개선

기존 TF-IDF 대신 다음 모델 사용:

- Sentence-BERT (SBERT)
- KoSentenceBERT
- E5-large, mContriever 등 dense retriever 계열 모델

문장 간 의미적 유사도를 기반으로 centrality를 구하면
Weak Label의 품질이 크게 개선될 것으로 기대된다.

(2) Human Label 확장 **+ Semi-supervised Learning**

현재 Human Label은 200개로 제한적이다.
이를 500~2000개 수준으로 확장하고:

- Self-training
- Pseudo-label refinement
- Consistency training

등을 적용하면 보다 높은 정밀도의 핵심문장 분류 모델을 구축할 수 있다.

(3) Classifier 모델 고도화

다음 모델이 유력한 개선 후보군이다.

- LightGBM / XGBoost
- RoBERTa / KoBERT 파인튜닝
- ELECTRA 기반 binary classifier
- Hybrid: TF-IDF + Neural embedding concatenation

특히 Transformer 기반 모델은 문맥 정보를 통합적으로 처리하기 때문에
본 문제에서 큰 성능 향상을 기대할 수 있다.

(4) 요약 모델 통합 **(Extractive + Abstractive)**

현재 시스템은 "문장 추출형 요약(extractive summarization)"만 제공한다.
향후 다음을 추가 가능하다.

- KoBART 기반 abstract summarizer
- 핵심 문장 기반 개조·재구성

- 사용자 맞춤형 길이 조절 요약

즉, 단순 핵심 문장 나열을 넘어
"읽기 쉬운 형태로 재서술된 완전한 요약문**"**을 생성할 수 있다.

(5) 강의 텍스트 자동 입력 기능

STT + 전처리 + 요약까지 통합한 end-to-end 시스템 구현 가능:

강의 음성 → Whisper(Speech-to-Text) → 전처리 → ML 모델 → 요약 → 자동 노트 PDF 생성

이 경우 본 프로젝트는 진짜 학습 지원 플랫폼으로 발전하게 된다.

5.4 프로젝트가 남긴 의미

이 프로젝트는 단순한 과제 수행을 넘어
"학습자로서, 개발자로서, 연구자로서의 시각"을 동시에 확장시키는 경험이었다.

- 처음으로 전처리–학습–배포–UI까지 완전한 ML 파이프라인을 스스로 구축한 경험
- 자연어처리 모델을 실사용 환경에서 다루며 "제품화(Productization)"의 개념을 체득
- 데이터 품질과 라벨링이 모델의 한계를 결정함을 깊이 경험
- Logistic Regression 같은 전통 모델도 적절한 문제에서는 여전히 강력하다는 사실 확인
- 무엇보다도 **"ML** 모델을 실제로 사용하는 사용자 경험**(UX)"**을 직접 설계해 본 점이 큰 배움이었다.

특히,

"내가 직접 사용하는 학습 도구를 내가 만들었다"

는 점에서 이 프로젝트는 높은 실용적 가치를 가진다.