KICE

시스템&솔루션개발

실기형 문제지

[2025년 파일럿 차수]

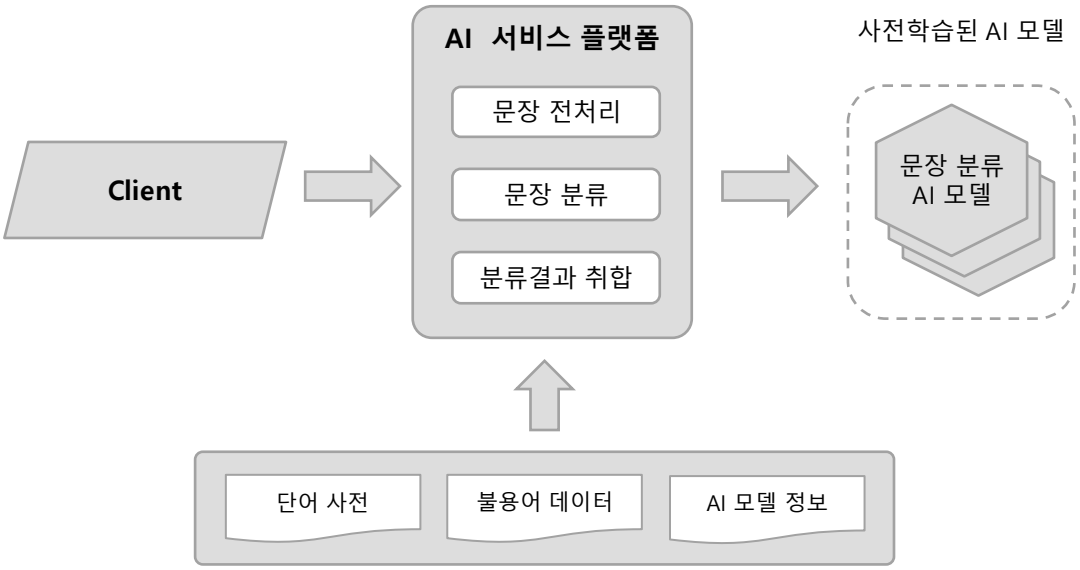
AI 서비스 플랫폼

개요

해당 시스템 구현을 통해 요구사항 분석, 데이터 구조화, HTTP Server/Client 구현 등의 기술역량 및 프로그램 구현 역량을 측정하기 위한 문제입니다.

설명

본 프로그램은 사전학습된 AI 모델을 활용하여 요청 받은 Query 문장들에 대한 분류를 수행하고 결과를 제공하는 AI 서비스 플랫폼입니다.



[기능 요약]

- 클라이언트가 AI 모델 추론 요청하면 먼저 각각의 Query 문장들에 대한 '문장 전처리'를 수행한다.
- 사전학습된 AI 모델로 '문장 분류'를 요청한다. ('문장 전처리' 결과를 송신하고, '분류코드'를 수신한다)
- AI 모델로부터 수신한 '분류코드'를 '분류결과'로 변환하여 취합한 후 클라이언트에 응답한다.

주의사항

실행 결과로 평가하고 부분점수는 없으므로 아래사항을 필히 주의해야 함

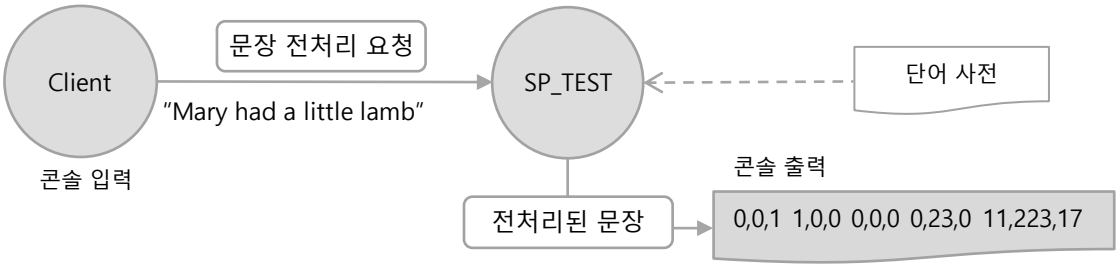
- 구현된 프로그램은 실행 완결성 필수 (명확한 실행&정확한 결과 출력, 통상의 실행 시간)
- 소 문항별 결과 검수 필수 (선행문항 오류 시, 후속문항 전체에 오류가 발생할 수 있음)
- 제시된 조건이 없는 한 선행요구사항 유지 필수
- 프로그램 실행 위치 및 실행결과출력 (위치, 파일명, 데이터포맷)은 요구사항과 정확히 일치 필수 (콘솔 출력이 평가 대상일 경우 불필요한 로그 출력 금지)
- 제시된 모든 위치는 상대경로 사용 필수 (프로그램 실행 위치 기준)
- 프로그램 종료조건에 맞는 처리 필수 (불필요한 입력대기를 하거나, 요구사항과 다르게 종료하면 안됨)
- 제공되는 샘플 파일과 다른 데이터로 채점하므로 제공되는 파일의 내용을 하드코딩하지 말 것
- 모든 문자는 요구사항에 맞는 대소문자 구분 필수

문제

아래 제시된 문항은 문항번호가 증가할수록 점진적 개선을 요구하는 방식으로 구성되어 있으며, 제시된 문항번호 별로 각각 **구현된 소스와 컴파일 된 실행파일을 제출**하시오.
cf) 1번 구현 → 1번 소스복사 → 2번 구현 → 2번 소스복사 → ...

1. 콘솔 입력을 통해서 "문장"을 입력하면 "단어"로 토큰화하고 "단어 사전"을 참조하여 워드임베딩된 결과(문장 전처리 결과)를 출력하시오 (20점)

상세설명

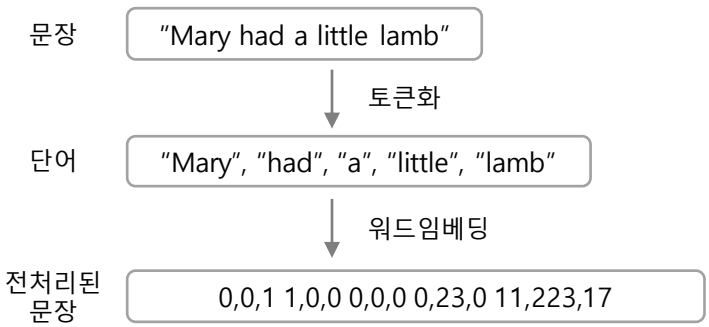


※ 토큰화
- "문장"을 공백문자로 구분된 "단어"로 쪼개는 작업
- ex) "Mary had a little lamb"은 "Mary", "had", "a", "little", "lamb"의 5 단어로 쪼개짐

※ 워드임베딩
- 각각의 <단어>를 "단어 사전"을 참조하여 <Embedding Vector>로 변환
- ex) 토큰화된 "Mary", "had", "a", "little", "lamb"의 5 단어를 각각 "단어 사전"을 참조하여 <Embedding Vector>로 변환 (단어 사전은 알파벳 소문자로만 구성되므로 소문자 변환 후 사전에서 찾을 것)

단어	소문자 변환	Embedding Vector
Mary	mary	0,0,1
had	had	1,0,0
a	a	0,0,0
little	little	0,23,0
lamb	lamb	11,223,17

※ 전처리된 문장
- "문장"을 토큰화 한 후에 워드임베딩하여 <Embedding Vector>로 변환된 결과
- 예시) "Mary had a little lamb" 전처리 결과



형식정보

※ 입력 문장 형식정보

- 형식 : <단어> + ' '(공백 문자) + ... + ' '(공백 문자) + <단어>
- 입력 문장 내 단어의 알파벳은 대소문자로 구성됨.

※ 단어 사전 형식정보

- 파일명 : DICTIONARY.TXT (각 소문항 홈 아래)
- 파일 형식 : <단어> + '#' + <Embedding Vector>

```
a#0,0,0
an#0,0,0
the#0,0,0
...
good#7,112,9816
...
```

- <단어> : 단어 사전 내 단어의 **<u>알파벳은 소문자로</u>** 구성됨
- <Embedding Vector> : 정수 3개로 구성되며 ',' (comma)로 구분됨

※ 콘솔 입/출력

- 입력 포맷 : <query 문장>
- 출력 포맷 : <Embedding Vector> + ' '(공백 문자) ... + ' '(공백 문자) + <Embedding Vector>

```
Mary had a little lamb                                    ← 콘솔 입력
0,0,1 1,0,0 0,0,0 0,23,0 11,223,17                        ← 콘솔 출력
<프로그램 종료하지 않고 계속 입력 및 결과 출력>
```

평가대상

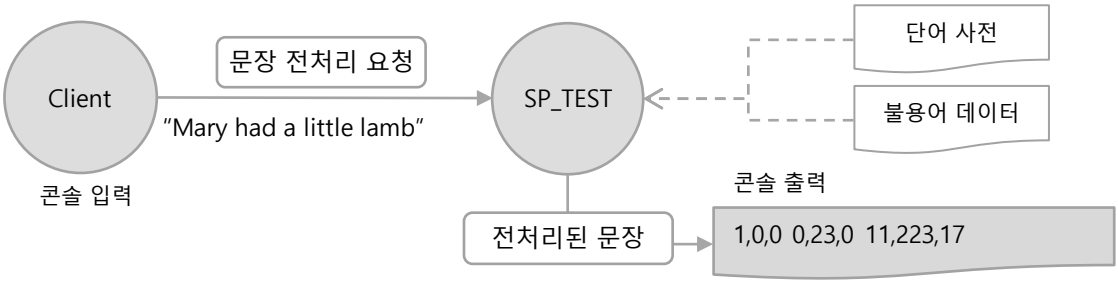
프로그램 정상 실행, 콘솔 입/출력 결과, **<u>프로그램 종료 없음</u>**
<u>자가 검수를 위해 제공되는 샘플은 채점용 데이터와 다름</u>

문제

2. 위 1번 문항까지 구현된 내용을 기준으로, 아래 사항을 추가로 반영하여 구현하시오. (30점)

- 불용어(분석에 사용하지 않는 단어) 데이터 추가
- 불용어를 제거한 후 워드임베딩된 결과(문장 전처리 결과)를 출력

상세설명

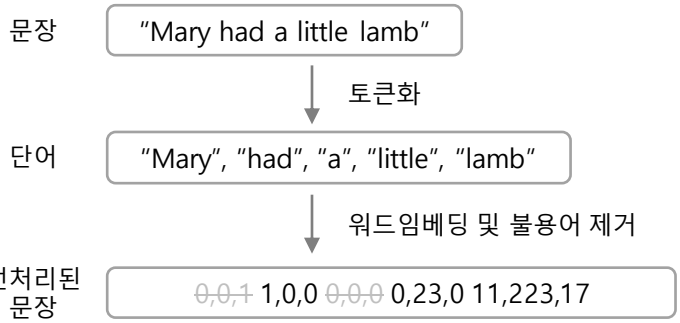


※ 불용어 제거

- 토큰화된 "단어" 가운데 분석에 사용하지 않는 단어를 제거하는 전처리 작업
- ex) 예를 들어, 불용어 데이터에 "0,0,0"이 포함되어 있다면, <Embedding Vector>가 "0,0,0"인 모든 "단어 " (예시에서는 'a', 'an', 'the'가 제외됨)

※ 전처리된 문장

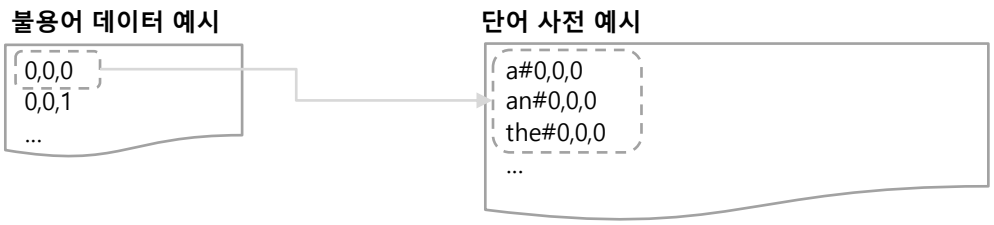
- "문장"을 토큰화 한 후에 불용어를 제외하고 워드임베딩하여 <Embedding Vector>로 변환된 결과
- 예시) "0,0,0", "0,0,1"이 불용어인 경우, "Mary had a little lamb" 전처리 결과



형식정보

※ '불용어 데이터' 파일 형식정보

- 파일명 : STOPWORD.TXT (각 소문항 홈 아래)
- 파일 형식 : <Embedding Vector>



평가대상

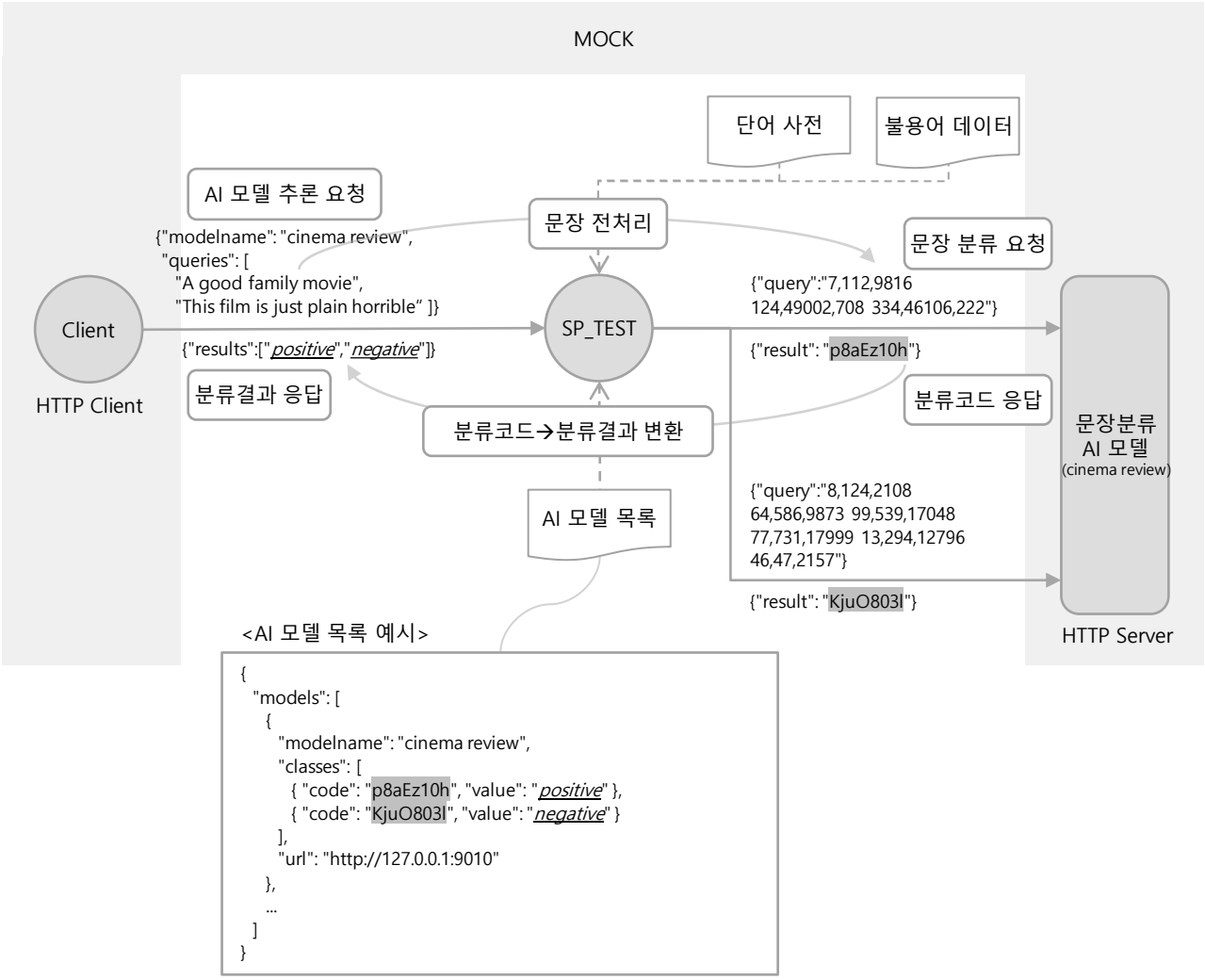
프로그램 정상 실행, 콘솔 입/출력 결과, **프로그램 종료 없음**
자가 검수를 위해 제공되는 샘플은 채점용 데이터와 다름

문제

3. 위 2번 문항까지 구현된 내용을 기준으로, 아래 사항을 추가로 반영하여 AI 서비스 플랫폼을 구현하시오. (50점)

- 사전학습된 모델의 상세정보가 '문장 분류 AI 모델 목록'으로 제공됨
- HTTP 통신으로 AI 모델 추론 요청을 수신하면 사전학습된 문장 분류 AI 모델을 활용하여 분류 결과를 응답

상세설명



※ 문장 분류 AI 모델

- 사전학습되어 전처리(토큰화, 워드임베딩, 불용어제거)된 문장을 입력하면 **분류코드**를 응답

※ AI 모델 추론 요청

- 사전학습된 문장 분류 AI 모델 중 하나를 선택하여, **여러 개의 Query 문장**에 대하여 추론 요청
- 여러 개의 Query 문장에 대한 분류 결과를 배열로 **순서대로** 응답
- AI 모델은 <분류코드>를 응답하므로 <분류결과>로 변환 필요

형식정보

※ "AI 모델 목록" 형식정보

- 파일 경로 : MODELS.JSON
- 아래의 Json 형식

```
{
  "models": [ {
    "modelname": <AI 모델명>,
    "classes": [
        { "code":<분류코드>, "value":<분류결과> },
        ...
    ],
    "url":<AI 모델 Url>,
  }, ...
  ]
}
```

- <AI 모델명> : 문장 분류에 사용할 사전학습된 AI 모델명
- <분류코드> : 문장 분류 AI 모델이 응답하는 분류 코드
- <분류결과> : 분류 코드에 해당하는 실제 분류 결과
- <AI 모델 Url> : 문장 분류를 수행하기 위해 호출할 Url

※ "AI 모델 추론 요청" 형식정보

- 아래의 Json 형식

```
{
  "modelname": <AI 모델명>
  "queries": [<Query 문장>,...,<Query 문장>],
}
```

- <AI 모델명> : 문장 분류에 사용할 사전학습된 AI 모델명
- <Query 문장> : 분류할 문장

AI 모델 추론 요청 예시

```
{
  "modelname": "cinema review",
  "queries": ["A good family movie", "This film is just plain horrible"]
}
```

※ "문장 분류 요청" 형식정보

- 아래의 Json 형식

```
{
  "query": <전처리된 문장>
}
```

- <전처리된 문장> : 문장 전처리 과정을 통해 '토큰화' 및 '워드임베딩'된 문장

문장 분류 요청 예시

```
{
  "query": "7,112,9816 124,49002,708 334,46106,222"
}
```

형식정보 (계속)

※ AI 모델 추론 요청

- URI : POST http://127.0.0.1:8080/
- 요청 Body : Json 문자열 형식 (※ "AI 모델 추론 요청 " 형식정보 참조)
- 동작 : 'AI 모델 목록' 중에 지정된 문장 분류 AI 모델을 활용하여 요청된 문장들을 **하나씩 순서대로** 분류한 후 결과 목록을 응답
- 응답 Status Code : 200 OK
- 응답 Body : Json 문자열 형식
  { "results": [<분류결과>,... ,<분류결과>] }
  ex) { "results": ["positive", "negative"] }

※ 문장 분류 요청

- URI : POST <AI 모델 Url>
- 요청 Body : Json 문자열 형식 (※ "문장 분류 요청 " 형식정보 참조)
- 동작 : 문장 분류를 수행하여 분류코드를 응답
- 응답 Status Code : 200 OK
- 응답 Body : Json 문자열 형식
  { "result": <분류코드> }
  ex) { "result": "p8aEz10h" }

※ 제공프로그램(MOCK.EXE)

- MOCK.EXE를 콘솔에서 실행하면 테스트 시나리오에 따라 요구사항을 순차적으로 테스트함
- MOCK.EXE는 자가 검수의 기능도 수행하며, 모든 테스트 시나리오 성공 시 다음의 문구가 콘솔에 출력

```
C:₩>MOCK.EXE<엔터키>        ← 제공프로그램 실행
...                         ← 테스트 시나리오에 따른 테스트 실행
테스트에 성공했습니다!      ← 소문항의 모든 테스트시나리오 성공
```

평가대상

프로그램 정상 실행, HTTP 요청, HTTP 응답 결과, **프로그램 종료 없음**
자가 검수를 위해 제공되는 샘플과 MOCK은 채점용 데이터와 다름

폴더 정보

※ 프로그램 및 파일 위치 정보 (실행위치 기반 상대경로 사용 필수)
- 구현할 프로그램 위치 및 실행 위치 : 각 소문항 홈 (SUB1/SUB2/SUB3)
- 자가 검수용 참고 파일명 : COMPARE 폴더 내 CMP_CONSOLE.TXT (SUB1/SUB2)
- 제공되는 MOCK 프로그램 파일명 : 각 소문항 홈 아래 MOCK.EXE (SUB3)

*** 제공되는 파일들은 문항에 따라 다를 수 있음**

*** 자가 검수를 위해 제공되는 샘플은 채점용 데이터와 다름**

실행 방식

※ 구현할 프로그램 형식
- 프로그램 형태 : 콘솔(Console) 프로그램
- 프로그램 파일명 : SP_TEST
- 실행 방식(문항1~2) : 콘솔 실행 → 콘솔 입/출력처리 (종료 없음)

```
C:₩>SP_TEST<엔터키>          ← 구현한 프로그램 실행 (Argument 없음)
...
```

- 실행방식(문항3) : 콘솔 실행 → 실시간 HTTP 요청 수신 (종료 없음)

```
C:₩>SP_TEST<엔터키>          ← 구현한 프로그램 실행 (Argument 없음)
...
```

제공 및 제출

✓ 각 언어별 제공파일 압축 해제 후 자동 생성된 폴더 사용 필수

✓ 제공되는 주요 내용
- 샘플 파일
- 제공 프로그램 실행파일 (MOCK.EXE)
- 제출시 사용할 문항별 폴더 구조

✓ 제출 파일 및 폴더 상세 내용 (각 언어별 실기 가이드 참고)

테스트 방법

※ 자가 검수를 위해 제공되는 샘플은 채점용 데이터와 다름

검수를 위한 샘플 결과 파일은 각 문항 출력 폴더(COMPARE)에 사전 제공됨

[문항1]

- SP_TEST를 실행한 후 콘솔 입/출력 결과를 샘플 결과 파일(CMP_CONSOLE.TXT)와 동일한지 비교

```
C:₩>SP_TEST<엔터키>              ← 구현한 프로그램 실행 (Argument 없음)
A good family movie                                  ← 콘솔 입력
0,0,0  7,112,9816  124,49002,708  334,46106,222      ← 콘솔 출력
It was great to see some of my favorite ...          ← 콘솔 입력
3,750,1951  81,220,6524  91,720,6265  4,708,4711  ...  ← 콘솔 출력
```

[문항2]

- SP_TEST를 실행한 후 콘솔 입/출력 결과를 샘플 결과 파일(CMP_CONSOLE.TXT)와 동일한지 비교

```
C:₩>SP_TEST<엔터키>              ← 구현한 프로그램 실행 (Argument 없음)
A good family movie                                  ← 콘솔 입력
7,112,9816  124,49002,708  334,46106,222             ← 콘솔 출력
It was great to see some of my favorite ...          ← 콘솔 입력
3,750,1951  81,220,6524  91,720,6265  92,164,7646  ...  ← 콘솔 출력
```

[문항3]

- SP_TEST를 실행한 후, MOCK.EXE를 실행
- MOCK.EXE의 콘솔에 출력되는 테스트 결과 확인

(SP_TEST가 문항의 요건을 만족하지 못하는 경우, MOCK.EXE의 콘솔에 테스트 Fail의 사유 또는 에러 메시지가 출력되므로 자가 검수에 참고 요망)

```
C:₩>MOCK.EXE<엔터키>
테스트를 시작합니다.
...
테스트에 성공했습니다!
```

수고하셨습니다.