

# Aivle 스쿨 지원 질문 답변 챗봇 제작 프로젝트

김영우

# 1. 주제

주제	Aivle School 지원 질문, 답변 챗봇 만들기
사전학습과목	언어 지능
데이터 출처	AIVLE School 홈페이지 Q&A 기반 자체 제작
데이터 구분	Text
문제 유형	Intent Classification 기반 챗봇
중점사항	<ul style="list-style-type: none"><li>▪ 자연어에 대한 형태소 분석하기</li><li>▪ 다양한 임베딩벡터를 기반한 모델링</li></ul>

## 2. Pain Point

### 챗봇 Chat Bot

음성이나 문자를 통한 인간과의 대화를 통해서 특정한 작업을 수행하도록 제작된 컴퓨터 프로그램



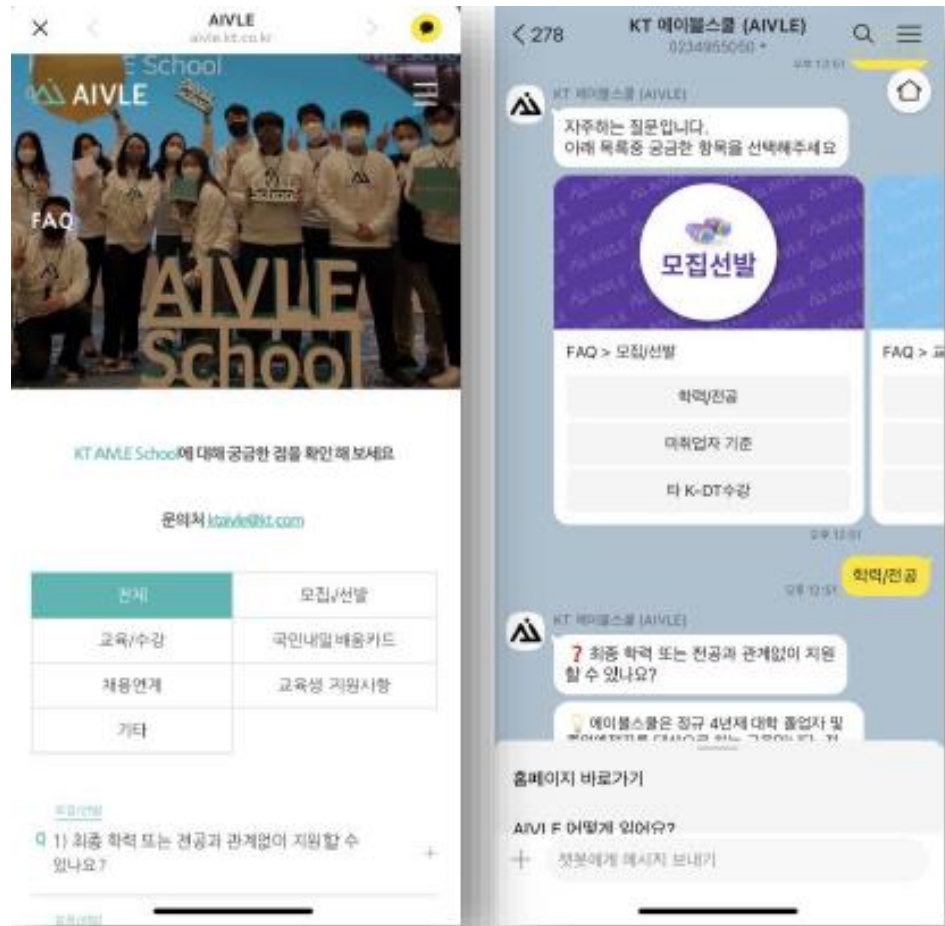
<https://ko.wikipedia.org/wiki/챗봇>

## 2. Pain Point

- Aivle School 지원자들의 단순 반복 문의에 따른 업무 부담 증가
  - 교육 운영담당자는 단순 반복 문의에 따른 업무 피로도 증대, 제한된 시간과 인력으로 인한 응대업무 한계 존재
- 교육 지원자는 교육 관련 문의를 위해 메일로 요청했는데,  
기다리는 시간이 오래 걸리는 현상 발생

## 2. Pain Point

- 따라서 이 문제를 해결하기 위해 챗봇(ChatBot)를 제작하려 한다.

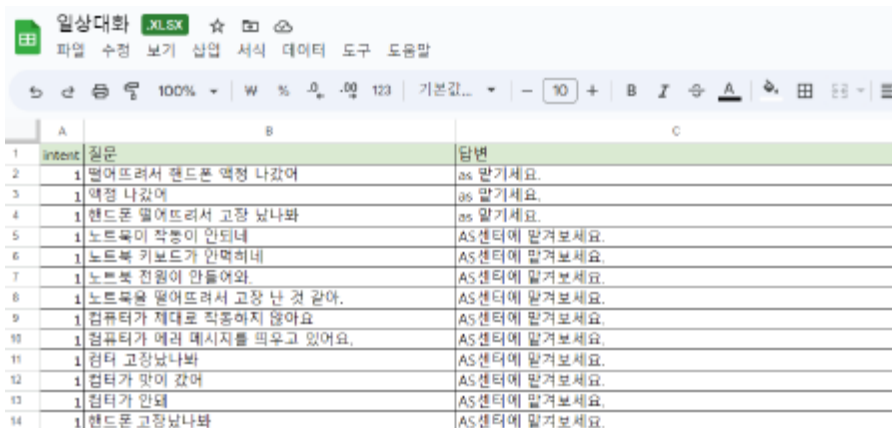


## 2. 챗봇 제작에 활용할 데이터 소개

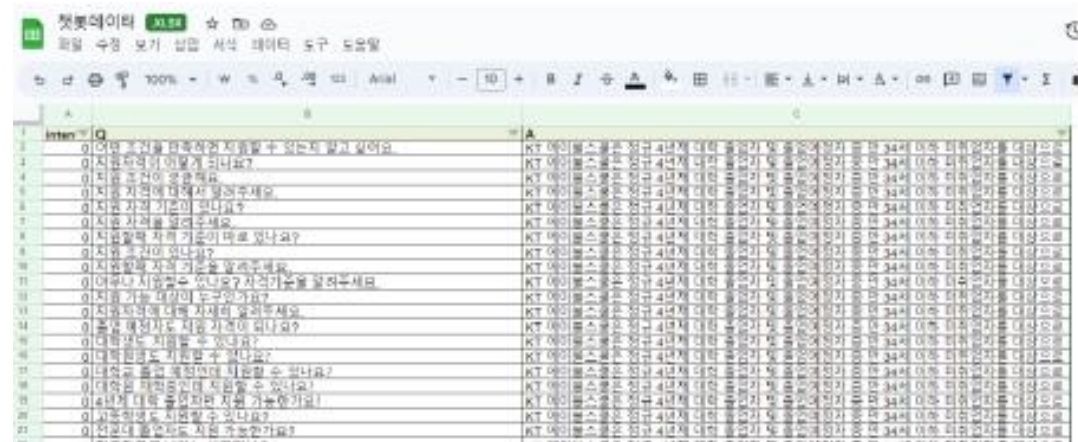
- 데이터명 : 질문 - 답변 데이터
- 원본 데이터 확보 방법 : 자체 제작
- 제작 방법 : 데이터 수집/제작

질문-답변 텍스트 인텐트 처리 가공

- 구성 내용 : 데이터유형 - 텍스트(Text)
- 데이터 제공 파일 : 일상대화.xlsx (일상적인 질문과 답변 intent)  
챗봇데이터.xlsx(에이블스쿨 지원 Q&A intent)



	A	B	C
1	intent	질문	답변
2	1	떨어뜨려서 핸드폰 액정 나갔어	as 맡기세요.
3	1	액정 나갔어	as 맡기세요.
4	1	핸드폰 떨어뜨려서 고장 났나봐	as 맡기세요.
5	1	노트북이 작동이 안되네	AS센터에 맡겨보세요.
6	1	노트북 키보드가 안먹히네	AS센터에 맡겨보세요.
7	1	노트북 전원이 안들어와	AS센터에 맡겨보세요.
8	1	노트북을 떨어뜨려서 고장 난 것 같아.	AS센터에 맡겨보세요.
9	1	컴퓨터가 제대로 작동하지 않아요	AS센터에 맡겨보세요.
10	1	컴퓨터가 여러 메시지를 띄우고 있어요.	AS센터에 맡겨보세요.
11	1	컴터 고장났나봐	AS센터에 맡겨보세요.
12	1	컴터가 맛이 갔어	AS센터에 맡겨보세요.
13	1	컴터가 안돼	AS센터에 맡겨보세요.
14	1	핸드폰 고장났나봐	AS센터에 맡겨보세요.



	A	B	C
1	intent	질문	답변
2	0	이런 조건의 제품적인지 확인할 수 있는지 알고 싶어요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
3	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
4	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
5	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
6	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
7	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
8	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
9	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
10	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
11	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
12	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
13	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
14	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
15	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
16	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
17	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
18	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
19	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
20	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로
21	0	지점별의 이모티콘이 궁금해요	KT 에이블스쿨은 전국 4년제 대학 졸업생이 입학한 지 4년 이내의 대학 재학생을 대상으로

## 2. 챗봇 제작에 활용할 데이터 소개



챗봇데이터.xlsx

( row 793, column 3 )

intent	Q	A
0	지원 자격이 궁금합니다.	KT 에이블스쿨은 정규 4년제 대학 졸업자 및 졸업예정자 중 만 34세 이하 미취업자를 대상으로 하는 교육입니다. 단, 모집시점에 만 35세여도 해당연도 1월 1일 이후 생일자는 지원이 가능합니다. 또한 전공의 종류와는 무관 합니다.
0	지원자격이 어떻게 되나요?	KT 에이블스쿨은 정규 4년제 대학 졸업자 및 졸업예정자 중 만 34세 이하 미취업자를 대상으로 하는 교육입니다. 단, 모집시점에 만 35세여도 해당연도 1월 1일 이후 생일자는 지원이 가능합니다. 또한 전공의 종류와는 무관 합니다.
...		
21	숙소나 기숙사에서 제공하는 객실의 크기와 시설은 어떻게 되나요?	숙소 및 기숙사에 대한 지원은 제공하지 않습니다.
21	대면 교육 시 별도로 교통비나 숙박비를 지원해주나요?	숙소 및 기숙사에 대한 지원은 제공하지 않습니다.
22	다른 국비 지원 교육보다 특별한 점이 있는지 궁금해요	KT 에이블스쿨에서는 이론/실습 교육에 나아가 기업 실전형 프로젝트를 중심으로 실무에서 일하는 방식을 배울 수 있습니다. 또한, KT 현직 전문가가 전담하여 산업지식, AI/DX 기술, 창의성을 갖출 수 있도록 강의와 코칭을 제공하고, 강의 중 궁금한 사항은 전담 튜터가 1:10로 직접 코칭을 제공합니다. 특히, 우수 수료생을 대상으로 KT와 KT 그룹, AI원팀, 협력사 등 채용까지 연계해 드립니다.

## 2. 챗봇 제작에 활용할 데이터 소개



일상대화.xlsx

( row 507, column 3 )

intent	질문	답변
1	떨어뜨려서 핸드폰 액정 나갔어	as 맡기세요.
1	액정 나갔어	as 맡기세요.
2	건강이 최고	가장 중요한 목표네요.
2	건강이 최고인 것 같아	가장 중요한 목표네요.
...		
29	부모님께 용돈 좀 드려야지	좋은 생각이예요.
29	이사했으니까 집들이 해야겠다	좋은 생각이예요.
30	고백 성공해서 오늘부터 1일이야~	기분 좋겠어요.
30	이번에 적금해지해서 돈 생겼다!	기분 좋겠어요.



### 3. 챗봇 제작 흐름

- 1. 사용자가 질문을 하면 자연어 처리에 특화된 숫자 형태로 변환
- 2. 머신러닝, 딥러닝 모델을 작동하여 Intent를 예측한다.
- 3. Intent에 따른 답변(A)를 랜덤하여 1개 선별한 다음 출력

## 4. 문제 정의

- 역할 : KT AI/서비스 연구개발팀 신입사원(가정)
- 업무 : 에이블스쿨 교육 지원 희망자를 대상으로  
에이블스쿨 지원 질문에 대응하는 답변 챗봇 서비스를 개발한다.

## 5. Key Point

- **1. 너무 딱딱한 질문 답변 챗봇에서 벗어나야 한다!**

- > 에이블스쿨 홈페이지에 올라와 있는 질문 답변으로 기반하여 제작한 챗봇 데이터뿐만 아니라 일상대화를 포함한 데이터셋도 추가했다.
- > Intent 수는 증가하겠지만 더 많은 데이터로 학습이 유리하고 그 결과 딱딱한 질문 답변 챗봇의 이미지를 어느정도 탈피할 수 있다.

- **2. 최대한 정확한 답변을 하는 챗봇을 제작해야 한다!**

- > 일상 대화 데이터셋 507개 + 에이블스쿨 지원 Q&A 데이터셋 793개로 비교적 적은 데이터셋으로 최대한 정확한 답변을 하는 챗봇을 제작해야 한다!
- > 1번째 챗봇을 제작하고 2번째 챗봇을 제작하려 할 때 성능 개선 버전 알고리즘(FastText)을 활용하려 한다.

## 6. 1번째 챗봇, 2번째 챗봇

- **1. 챗봇 1(Word2Vec 임베딩 벡터 기반 머신러닝 분류 모델링)**

- > Word2Vec 모델을 만들고 임베딩 벡터 생성
- > 임베딩 벡터를 이용하여 intent를 분류하는 모델링(LightGBM)
- > 예측된 intent의 답변 중 임의의 하나를 선정하여 출력

- **2. 챗봇 2(단계별 모델링)**

- > type(일상 대화 0, 에이블스쿨 Q&A 대화 1) 분류 모델 만들기  
(Embedding Layer를 활용한 딥러닝 모델)
- > 사용자의 질문과 데이터셋의 대화에 따른 FastText 모델 생성 후 임베딩 벡터 생성
- > 코사인 유사도를 활용하여 intent를 찾아, 답변 중 임의의 하나를 선정하여 출력

## 7. 데이터 탐색 & 전처리

일상 대화를 담은 데이터셋이 어떻게 이루어져 있는지 확인한다.

```
1 common_df.head()
```

	intent	질문	답변
0	1	떨어뜨려서 핸드폰 액정 나갔어	as 맡기세요.
1	1	액정 나갔어	as 맡기세요.
2	1	핸드폰 떨어뜨려서 고장 났나봐	as 맡기세요.
3	1	노트북이 작동이 안되네	AS센터에 맡겨보세요.
4	1	노트북 키보드가 안먹히네	AS센터에 맡겨보세요.

```
[10] 1 common_df.shape
```

```
(506, 3)
```

## 7. 데이터 탐색 & 전처리

일상 대화를 담은 데이터셋의 intent가 어디까지 분포되어 있는지 확인하고 Intent 1 ~ Intent 30까지 답변(Q)가 무엇인지 확인한다.

```
[11] 1 common_df_intent = common_df['intent'].unique()
```

```
1 common_df_intent
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,  
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30])
```

```
1 for cdi in common_df_intent:  
2     print(f'intent : {cdi}일 때')  
3     print(common_df.loc[common_df['intent'] == cdi, '답변'])  
4     print()
```

```
intent : 27일 때  
415 저도요!  
416 저도요!  
417 저도요!  
418 저도요!  
419 저도요!!  
420 저도요!!  
421 저도요.  
422 저도요.  
423 저도요.  
424 저도요.  
425 저도요!!  
426 저도요!!  
427 저도요!!  
428 저도요!!  
429 저도요.  
Name: 답변, dtype: object
```

## 7. 데이터 탐색 & 전처리

Aivle School 지원 Q&A 를 담은 데이터셋이 어떻게 이루어져 있는지 확인한다.

```
1 faq_df.head()
```

	intent	Q	A
0	0	어떤 조건을 만족하면 지원할 수 있는지 알고 싶어요.	KT 에이블스쿨은 정규 4년제 대학 졸업자 및 졸업예정자 중 만 34세 이하 미취업자를 대상으로 하는 교육입니다.wn단, 모집 시점에 만 35세여도 해당연도 1월 1일 이후 생일자는 지원이 가능합니다.wn또한 전공의 종류와는 무관 합니다.
1	0	지원자격이 어떻게 되나요?	KT 에이블스쿨은 정규 4년제 대학 졸업자 및 졸업예정자 중 만 34세 이하 미취업자를 대상으로 하는 교육입니다.wn단, 모집 시점에 만 35세여도 해당연도 1월 1일 이후 생일자는 지원이 가능합니다.wn또한 전공의 종류와는 무관 합니다.
2	0	지원 조건이 궁금해요.	KT 에이블스쿨은 정규 4년제 대학 졸업자 및 졸업예정자 중 만 34세 이하 미취업자를 대상으로 하는 교육입니다.wn단, 모집 시점에 만 35세여도 해당연도 1월 1일 이후 생일자는 지원이 가능합니다.wn또한 전공의 종류와는 무관 합니다.
3	0	지원 자격에 대해서 알려주세요.	KT 에이블스쿨은 정규 4년제 대학 졸업자 및 졸업예정자 중 만 34세 이하 미취업자를 대상으로 하는 교육입니다.wn단, 모집 시점에 만 35세여도 해당연도 1월 1일 이후 생일자는 지원이 가능합니다.wn전공은 상관 없습니다.
4	0	지원 자격 기준이 있나요?	KT 에이블스쿨은 정규 4년제 대학 졸업자 및 졸업예정자 중 만 34세 이하 미취업자를 대상으로 하는 교육입니다.wn단, 모집 시점에 만 35세여도 해당연도 1월 1일 이후 생일자는 지원이 가능합니다.wn전공은 상관 없습니다.

```
1 faq_df.shape
```

(792, 3)

## 7. 데이터 탐색 & 전처리

Aivle School 지원 Q&A 를 담은 데이터셋의 inten가 어디까지 분포되어 있는지 확인하고 Intent 0 ~ Intent 22까지 답변(Q)가 무엇인지 확인한다.

```
1 faq_df_intent = faq_df['intent'].unique()
```

- 에이블스쿨 FAQ 데이터의 intent가 0부터 22까지가 있다.

```
[ ] 1 faq_df_intent
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22])
```

- 저는 intent 0에서는 어떤 답변이 있고
- intent 1에서는 어떤 답변이 있고 ~ intent 22에서는 어떤 답변이 있는지 궁금해했습니다.

```
1 for fdi in faq_df_intent:
2     print(f'intent : {fdi}일 때')
3     print(faq_df.loc[faq_df['intent'] == fdi, 'A'])
4     print()
```

```
608 교육기간 동안 노트북을 무료로 지원하며, 교육에 필요한 필수 프로그램이 설치되어 있으므로, 제공한 노트북을 사용해 주시기 바랍니다.
609 교육기간 동안 노트북을 무료로 지원하며, 교육에 필요한 필수 프로그램이 설치되어 있으므로, 제공한 노트북을 사용해 주시기 바랍니다.
610 교육기간 동안 노트북을 무료로 지원하며, 교육에 필요한 필수 프로그램이 설치되어 있으므로, 제공한 노트북을 사용해 주시기 바랍니다.
611 교육기간 동안 노트북을 무료로 지원하며, 교육에 필요한 필수 프로그램이 설치되어 있으므로, 제공한 노트북을 사용해 주시기 바랍니다.
Name: A, dtype: object

intent : 16일 때
612 KT 에이블스쿨은 자기주도적인 학습을 장려하고 있으며, 교과목 종료 시 선택테스트를 통한 역량 점검이 진행 됩니다.
613 KT 에이블스쿨은 자기주도적인 학습을 장려하고 있으며, 교과목 종료 시 선택테스트를 통한 역량 점검이 진행 됩니다.
614 KT 에이블스쿨은 자기주도적인 학습을 장려하고 있으며, 교과목 종료 시 선택테스트를 통한 역량 점검이 진행 됩니다.
615 KT 에이블스쿨은 자기주도적인 학습을 장려하고 있으며, 교과목 종료 시 선택테스트를 통한 역량 점검이 진행 됩니다.
616 KT 에이블스쿨은 자기주도적인 학습을 장려하고 있으며, 교과목 종료 시 선택테스트를 통한 역량 점검이 진행 됩니다.
617 KT 에이블스쿨은 자기주도적인 학습을 장려하고 있으며, 교과목 종료 시 선택테스트를 통한 역량 점검이 진행 됩니다.
618 KT 에이블스쿨은 자기주도적인 학습을 장려하고 있으며, 교과목 종료 시 선택테스트를 통한 역량 점검이 진행 됩니다.
619 KT 에이블스쿨은 자기주도적인 학습을 장려하고 있으며, 교과목 종료 시 선택테스트를 통한 역량 점검이 진행 됩니다.
```



## 7. 데이터 탐색 & 전처리

- **Intent 번호 조정**

- > 일상 대화 데이터 셋 intent는 1 ~ 30까지 존재
- > Aivle School 데이터 셋 intent는 0 ~ 22로 존재
- > 따라서 Aivle School 데이터 셋 intent를 31 ~ 53으로 변경하는 작업을 한다.

- intent 번호 조정

```
[ ] 1  faq_df['intent'].unique() # 기존의 faq_df 번호는 0부터 22까지 배치되었습니다.  
  
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,  
       17, 18, 19, 20, 21, 22])
```

```
[ ] 1  nums = list(range(0, 23, 1))
```

```
▶ 1  # faq_df의 intent가 0이면 31로, 1이면 32로 ~~~ 22면 53으로 변경해줘요!  
2  for num in nums:  
3  |    faq_df.loc[faq_df['intent'] == num, 'intent'] = num + 31
```

```
▶ 1  faq_df['intent'].unique() # faq_df 번호가 변경됐는지 프린트합니다.
```

```
📄 array([31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,  
       48, 49, 50, 51, 52, 53])
```

## 7. 데이터 탐색 & 전처리

- **pd.concat()를 통해 두 데이터셋 통합**
  - > 일상 대화 데이터셋과 Aivle School 지원 Q&A 데이터셋 통합
  - > df.shape가 1298건으로 확장

### • 데이터셋 통합(pd.concat)

```
[24] 1 df = pd.concat([common_df, faq_df], axis=0) # 아래 방향으로 두 데이터프레임을 통합합니다.
```

```
[25] 1 df.shape # 아래 방향으로 잘 합쳐졌는지 한번 프린트합니다.
```

```
(1298, 3)
```

```
1 df['intent'].unique() # intent가 1부터 53까지 잘 계시는지 한번 봅시다.
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53])
```

## 7. 데이터 탐색 & 전처리

- 일상 대화와 Aivle School 지원 Q&A 대화를 구분하는 칼럼 'type' 생성

- > Intent가 1 ~ 30이면 일상 대화로 type을 0으로 생성

- > Intent가 31 ~ 53이면 Aivle School 지원 Q&A 대화로 type을 1로 생성

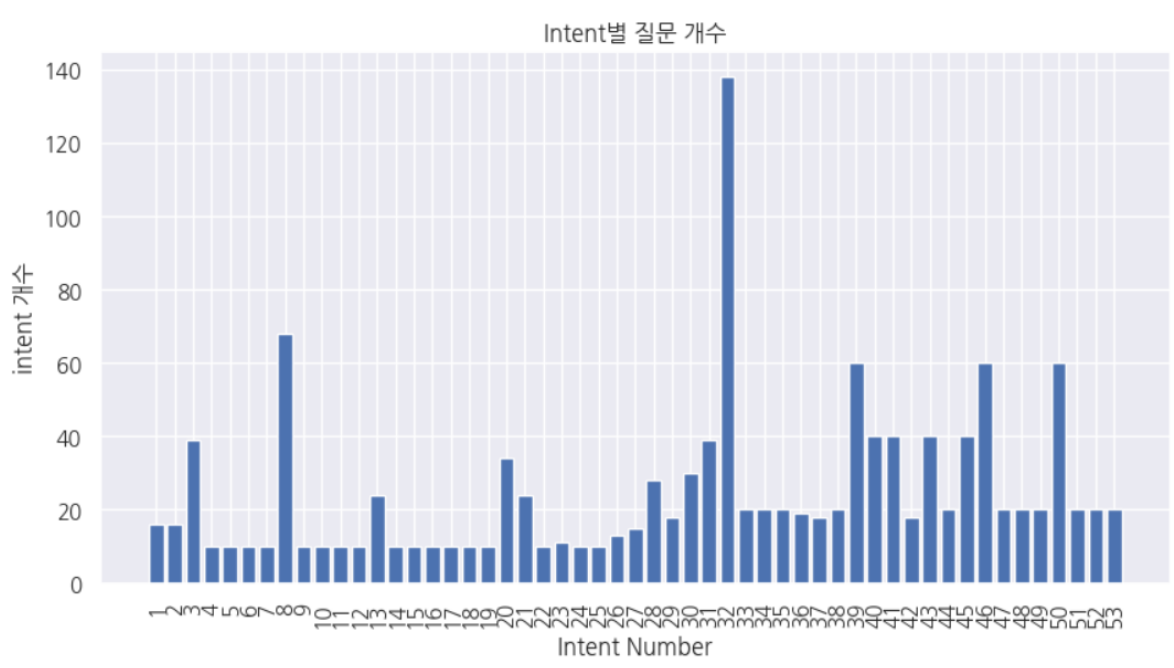
```
1 # 'type' 컬럼 설정
2 df['type'] = df['intent'].apply(lambda x: 0 if 1 <= x <= 30 else 1)
```

+ 코드   + 텍스트

## 7. 데이터 탐색 & 전처리

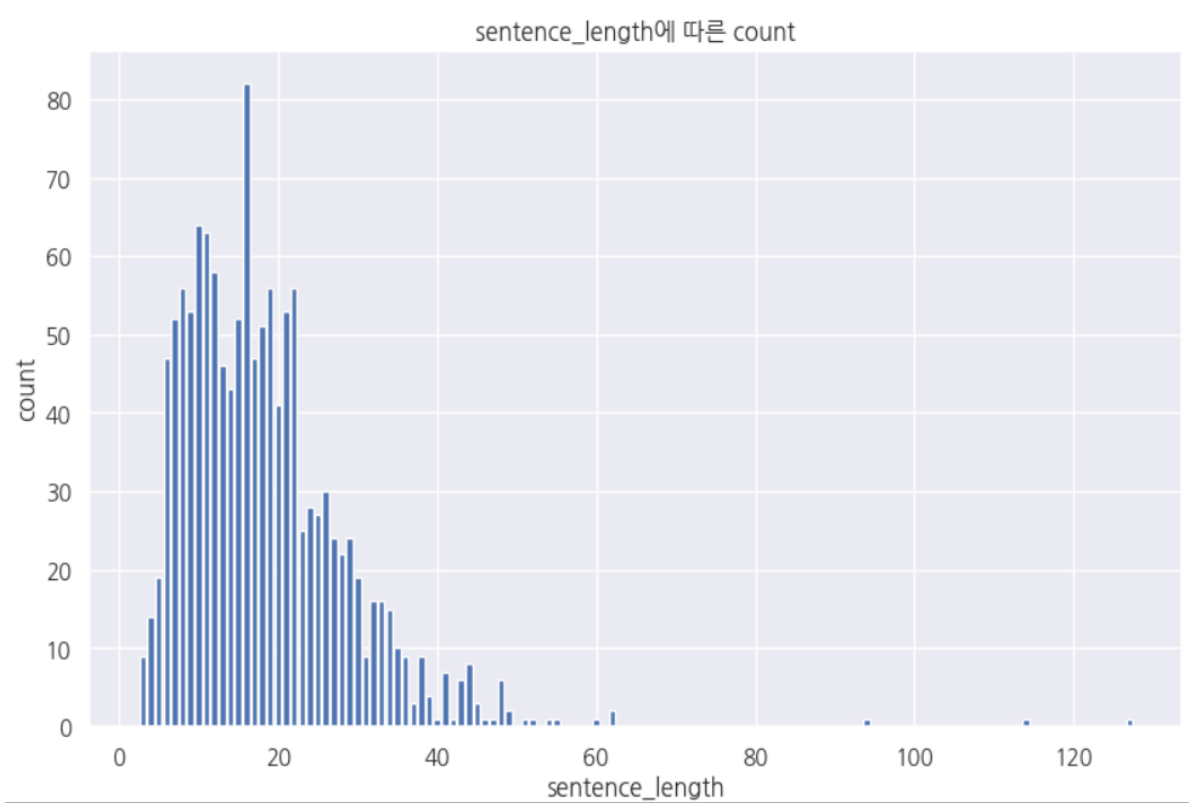
- **Intent별 질문 개수 분포 확인하기**

- > Intent 개수가 불균형을 이루고 있음을 알 수 있다. -> **클래스 불균형**
  - > 따라서 머신러닝, 딥러닝 모델을 학습할 때 클래스 불균형이 정확도를 높이는 모델로 만드는데 방해가 된다.
  - > 클래스 불균형을 대처하기 위한 방법
  - > 1. 리샘플링(Resampling)을 통해 오버샘플링(Oversampling) 하거나 언더샘플링(Undersampling)을 한다.
  - > **2. 모델을 학습할 때 소수 클래스에 대해서 가중치를 많이 부여한다.**
- > 개인적으로 2번째 방법을 선호하기 때문에 모델을 학습할 때 **2번째 방법**을 활용할 예정이다.



## 7. 데이터 탐색 & 전처리

- 질문(Q)별 문장 길이 분포 확인하기

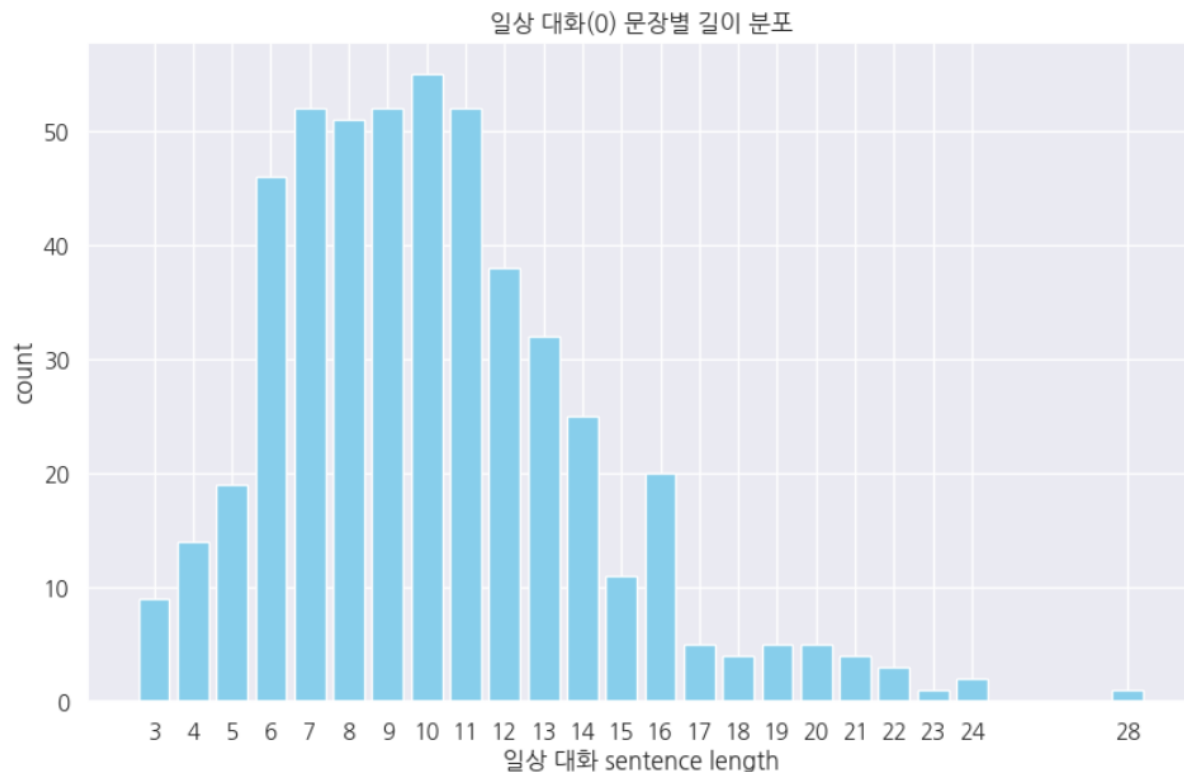


## 7. 데이터 탐색 & 전처리

- **일상 대화(O) 질문(Q)별 문장 길이 분포 확인**

- > 굳이 이상치라고 판단한다면 길이가 28이라 할 수 있다.

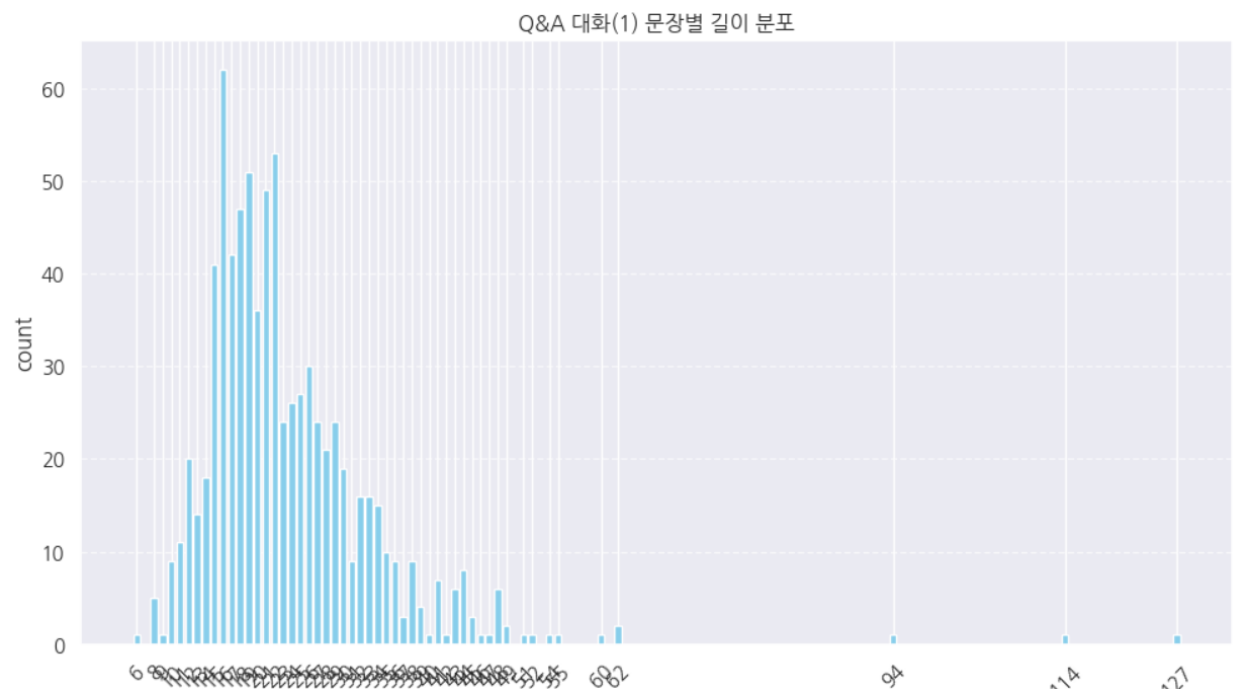
- > 하지만 17, 18, 19 ~ 24나 비교적 문장 길이나 28이나 비교적 같은 길이의 문장 길이 이므로 개인적으로 이상치라고 판단하지는 않았다.



## 7. 데이터 탐색 & 전처리

- **Aivle School 지원 Q&A 대화 질문(Q)별 문장 길이 분포 확인**

- > 이상치라고 판단한다면 길이가 94, 114, 127이라 할 수 있다.
- > 개인적으로 들었던 생각은 길이가 94, 114, 127인 행은 삭제를 하거나, 길이를 60 정도로 줄여보는 것도 괜찮다는 생각이 들었다.
- > 하지만 저는 생각만 해봤고, 실제 대치는 하지 않았다.



## 7. 데이터 탐색 & 전처리

- 학습/평가 데이터 분리

- > test 데이터 : intent마다 무작위로 질문 2개를 뽑아 test 데이터로 분리
- > train 데이터 : 나머지 데이터

```
1 # 테스트 데이터를 저장할 빈 데이터프레임 생성
2 test_df = pd.DataFrame(columns=df.columns)
3
4 # 각 intent 그룹별로 무작위로 2개의 행을 선택하여 테스트 데이터로 추가
5 for intent in df['intent'].unique():
6     intent_group = df.loc[df['intent'] == intent, :]
7     test_samples = intent_group.sample(2, random_state=1) # 랜덤 스테이트 설정으로 일관된 결과 얻기
8     test_df = pd.concat([test_df, test_samples])
9
10 # 테스트 데이터에 속하는 행을 원본 데이터프레임에서 제거하여 트레이닝 데이터 생성
11 train_df = df.drop(test_df.index)
```

```
[47] 1 train_df.shape
```

```
(1192, 4)
```

```
[48] 1 test_df.shape
```

```
(106, 4)
```



## 7. 데이터 탐색 & 전처리

- 1. 질문(Q)에 대해서 맞춤법을 적용한다.
  - > 질문(Q)을 사람이 작성한 것이므로, 맞춤법을 100% 생각하여 작성한 것으로 판단하기 이른다.
  - > 따라서 질문(Q)에 대해서 맞춤법을 적용하겠다는 생각이 들었다.
  - > 맞춤법 라이브러리로 Hanspell이 많이 사용되지만,  
서버 상태에 따라서는 사용이 제한될 수 있다 라는 의견이 있었다.  
KeyError가 나서 사용하지 못하여 다른 방법을 이용하기로 했지만 결국 실패했다.  
그래서 맞춤법은 적용하지 못했다.

## 7. 데이터 탐색 & 전처리

- 2. 질문(Q)에 대한 특수문자를 제거한다.

```
1 import re
2 import string
3 #removal_list = ".,;:'\"`~!@#$%^&*()_+-=|{}[]\0-\n\r\t' ", ">>, <, /, ~, -, ., *, <, >, ,, ?, !, [, ], ... , ◆, %"
4 removal_list = ".,:;'\"`~!@#%&*()-+=|{ } [ ] \0-\n\r\t' " + "%◆%"
5 removal_list += string.punctuation      (class) str
6 def cleansing_special(sentence: str = None) -> str:
7     """
8     특수문자를 전처리를 하는 함수
9     :param sentence: 전처리 대상 문장
10    :return: 전처리 완료된 문장
11    """
12    sentence = re.sub("[.\,#\'\"'" "" "[?]", "", sentence)
13    sentence = re.sub("[^가-힣0-9a-zA-Z\\w\s]", " ", sentence)
14    sentence = re.sub("#s+", " ", sentence)
15
16    sentence = sentence.translate(str.maketrans(removal_list, ' '*len(removal_list)))
17    sentence = sentence.strip()
18    return sentence
```

```
1 train_df['Q'] = train_df['Q'].apply(cleansing_special)
2 test_df['Q'] = test_df['Q'].apply(cleansing_special)
```

## 7. 데이터 탐색 & 전처리

- 3. 직접 구한 한글 불용어 사전을 활용하여  
질문(Q)에 대해서 의미가 없는 단어라고 판단되는 단어는 제거한다.

```
[ ] 1 # 한글 불용어 사전을 만들어요...
2 korean_stopwords = []
3 for fileName in ['koreanStopwords.txt', 'stopwords.txt', 'stopword2.txt', 'stopword3.txt', 'stopword4.txt']:
4     f = open(f'/content/drive/MyDrive/미니프로젝트6차part1/데이터/한국어불용어사전/{fileName}', 'r')
5     lines = f.readlines()
6     for line in lines:
7         line = line.replace('\n', '')
8         line = line.replace('\t', '')
9         korean_stopwords.append(line)
10    f.close()
11 korean_stopwords = list(set(korean_stopwords))
```

```
1 from konlpy.tag import Okt
2
3 okt = Okt()
4
5 def remove_korean_stopwords(text): # 불용어 제거 함수
6     morphs = okt.morphs(text) # 텍스트를 형태소 단위로 분리
7     filtered_words = [morph for morph in morphs if morph not in korean_stopwords] # 한글 불용어 제거
8     return ' '.join(filtered_words)
```

```
1 # 불용어 제거 적용
2 train_df['Q'] = train_df['Q'].apply(remove_korean_stopwords)
3 test_df['Q'] = test_df['Q'].apply(remove_korean_stopwords)
```

## 7. 데이터 탐색 & 전처리

- 4. 형태소 분석을 하는 함수를 만든다.

```
1 from konlpy.tag import Okt, Komoran, Mecab, Hannanum, Kkma
2
3 # 다양한 토크나이저를 사용할 수 있는 함수
4 def get_tokenizer(tokenizer_name):
5     if tokenizer_name == "komoran":
6         tokenizer = Komoran()
7     elif tokenizer_name == "okt":
8         tokenizer = Okt()
9     elif tokenizer_name == "mecab":
10        tokenizer = Mecab()
11    elif tokenizer_name == "hannanum":
12        tokenizer = Hannanum()
13    else:
14        # "kkma":
15        tokenizer = Kkma()
16
17    return tokenizer
```

```
1 # 형태소 분석을 수행하는 함수
2 def tokenize(tokenizer_name, original_sent, nouns=False):
3     # 미리 정의된 몇 가지 tokenizer 중 하나를 선택
4     tokenizer = get_tokenizer(tokenizer_name)
5
6     # tokenizer를 이용하여 original_sent를 토큰화하여 tokenized_sent에 저장하고, 이를 반환합니다.
7     sentence = original_sent.replace('\n', '').strip()
8     if nouns:
9         # tokenizer.nouns(sentence) -> 명사만 추출
10        tokens = tokenizer.nouns(sentence)
11    else:
12        tokens = tokenizer.morphs(sentence)
13    tokenized_sent = ' '.join(tokens)
14
15    return tokenized_sent
```

## 7. 데이터 탐색 & 전처리

- 5. 질문(Q)에 대해서 형태소 분석을 하여 clean\_train\_questions, clean\_test\_questions 이름으로 저장한다.  
-> 후에 이들은 자연어 처리에 특화된 숫자 형태로 변환할 예정이다.

```
1 clean_train_questions = train_df['Q'].apply(lambda q : tokenize(tokenizer_name='okt', original_sent=q, nouns=False)) # Series로 저장
2 clean_test_questions = test_df['Q'].apply(lambda q : tokenize(tokenizer_name='okt', original_sent=q, nouns=False)) # Series로 저장
```

```
1 clean_test_questions
9      아빠 생각난다
0      번호 달라 고 해볼까
1      번호 따 볼까
2      어딘가 떠나고 싶다
3      돈 벌고 싶어
4      놀 없네
5      위로 해줄 필요해
6      야구장 이나 갈까
7      가족 이랑 여행 가려고
8      친구 랑 수다 떨고 왔어
9      선물 받았어
0      지원 조건 궁금해요
1      지원 자격 나이 상관 있나요
2      재직 에는 참여 불가능한가요
3      재직 에도 수강 가능한가요
4      이전 불합격 했는데 지원 할 있나요
5      서류 탈락자 도 재지 원할 있나요
6      합격 후 교육 등록 신청 하지 않으면 재 지원이 가능한가요
7      중도 포기 재 지원 할 있나요
8      합격 과정 포기 하는 추가 합격자 생기나요
9      추가 합격 은 없나요
0      면접 도 보나 요
1      면접 은 대면 인가요 비 대면 인가요
2      코딩 테스트 효과 인 해결 책 찾기 위 팀 은 인가요
3      코딩 테스트 보기 준비 해야 할 사항 있나요
4      인 적성검사 KT 인 적성검사 동일한지 궁금합니다
5      KT 인 적성검사 과목 같은가요
6      대학원생 인 서류 되나요
```

## 8. 챗봇 1 모델링

- 상세 요구사항

- > Word2Vec를 활용한 LightGBM 모델링(Intent 분류)

- > Word2Vec를 이용하여 임베딩 벡터 생성하기

- > Word Embedding으로 문장벡터 구하기

- > 임베딩 벡터를 이용하여 ML 기반 모델링 수행하기

- > LightGBM ML 알고리즘을 이용한다.

- > 챗봇 : 모델의 예측결과(intent)에 따라 답변하는 챗봇 만들기

- > 질문을 입력받아, 답변하는 함수 생성

## 8. 챗봇 1 모델링

- Word2Vec를 쓰기 위해서 질문 데이터를 리스트 형태로 변환한다.

```
1 # 시리즈를 리스트로 변환하고 토큰화 -> 2차원 리스트임을 알 수 있다.  
2 clean_train_questions = [okt.morphs(sentence) for sentence in clean_train_questions]
```

```
[['떨어뜨려서', '핸드폰', '액정', '나갔어'],  
 ['액정', '나갔어'],  
 ['핸드폰', '떨어뜨려서', '고장', '났나'],  
 ['노트북', '키', '보드', '막히네'],  
 ['노트북', '전원', '들어와'],  
 ['노트북', '떨어뜨려서', '고장', '난', '같아'],  
 ['컴퓨터', '제대로', '작동', '하지', '않아요'],  
 ['컴퓨터', '에러', '메시지', '띄우고', '있어요'],  
 ['컴터', '고장', '났나'],  
 ['컴터', '맛', '갔어'],  
 ['컴터', '돼'],  
 ['핸드폰', '고장', '났나'],  
 ['핸드폰', '돼'],  
 ['노트북', '돼'],  
 ['의지', '는', '상관없나'],  
 ['의지', '안되는', '일인', '가봐'],  
 ['건강', '최고'],  
 ['아프면', '되는데'],  
 ['올해', '도', '건강하길'],  
 ['올해', '도', '행복하길'],  
 ['올해', '취업', '하길'],  
 ['올해', '한겨', '하길'],  
 ['건강', '최고'],
```

## 8. 챗봇 1 모델링

- Word2Vec를 쓰기 위해서 질문 데이터를 리스트 형태로 변환한다.

```
[ ] 1 # 시리즈를 리스트로 변환하고 토큰화 -> 2차원 리스트임을 알 수 있다.  
    2 clean_test_questions = [okt.morphs(sentence) for sentence in clean_test_questions]
```

```
▶ 1 clean_train_questions[0:5:1] # Word2Vec를 만들기 위해 2차원 리스트 형태로 되어있는 것을 확인할 수 있다.
```

```
↳ [['떨어뜨려서', '핸드폰', '액정', '나갔어'],  
    ['액정', '나갔어'],  
    ['핸드폰', '떨어뜨려서', '고장', '났나'],  
    ['노트북', '키', '보드', '먹히네'],  
    ['노트북', '전원', '들어와']]
```



## 8. 챗봇 1 모델링

- 질문 데이터를 자연어 처리에 최적화된 숫자 형태로 변경하기 위해 Word2Vec 모델을 정의한다.

```
[ ] 1 SIZE = 100
    2 WINDOW = 5
    3 MIN_COUNT = 1
    4 WORKERS = 4
```

```
▶ 1 from gensim.models import Word2Vec
   2
   3 # Word2Vec 모델 생성
   4 wv_model = Word2Vec(sentences=clean_train_questions, # 1번째 매개변수로 2차원 리스트를 기대한다.
   5                     vector_size=SIZE,
   6                     window=WINDOW,
   7                     min_count=MIN_COUNT,
   8                     workers=WORKERS,
   9                     sg=0, )
```

## 8. 챗봇 1 모델링

- 질문 데이터를 자연어 처리에 최적화된 숫자 형태로 변경하는 실질적 부분이다.
- `get_dataset()`, `get_sent_embedding()`

```
1 # Word2Vec 모델로부터 하나의 문장을 벡터화 시키는 함수 생성
2 def get_sent_embedding(model, embedding_size, tokenized_words):
3     # 임베딩 벡터를 0으로 초기화
4     feature_vec = np.zeros((embedding_size,), dtype='float32')
5     # 단어 개수 초기화
6     n_words = 0
7     # 모델 단어 집합 생성
8     index2word_set = set(model.wv.key_to_index.keys())
9     # 문장의 단어들을 하나씩 반복
10    for word in tokenized_words:
11        # 모델 단어 집합에 해당하는 단어일 경우에만
12        if word in index2word_set:
13            # 단어 개수 1 증가
14            n_words += 1
15            # 임베딩 벡터에 해당 단어의 벡터를 더함
16            feature_vec = np.add(feature_vec, model.wv.get_vector(word))
17    # 단어 개수가 0보다 큰 경우 벡터를 단어 개수로 나눴음 (평균 임베딩 벡터 계산)
18    if (n_words > 0):
19        feature_vec = np.divide(feature_vec, n_words)
20    return feature_vec
```

```
[ ] 1 # 문장벡터 데이터 셋 만들기
2 def get_dataset(sentences, model, num_features):
3     dataset = list()
4
5     # 각 문장을 벡터화해서 리스트에 저장
6     for sent in sentences:
7         dataset.append(get_sent_embedding(model, num_features, sent))
8
9     # 리스트를 numpy 배열로 변환하여 반환
10    sent_embedding_vectors = np.stack(dataset)
11
12    return sent_embedding_vectors
```

## 8. 챗봇 1 모델링

- train에 있었던 질문 데이터를 Word2Vec 모델을 활용하여 자연어 처리에 특화된 숫자 형태로 변환한다.  
-> num\_features를 100으로 주면 100차원으로 설정한다는 뜻이다.

- 이제 학습데이터의 Q를 Word2Vec 모델을 사용하여 벡터화 합니다.

```
1 # 학습 데이터의 문장들을 Word2Vec 모델을 사용하여 벡터화
2 train_data_vecs = get_dataset(sentences=clean_train_questions, model=word_embeddings, num_features=100) # 1번째 매개변수는 2차원 리스트를 기대한다.
```

```
1 train_data_vecs # Word2Vec 모델로부터 학습 데이터를 벡터화 한 결과입니다.
```

```
array([[ 0.00188884,  0.00409556,  0.00337401, ..., -0.0039295 ,
         0.00452117,  0.0008851 ],
       [ 0.00155076,  0.00153275,  0.00029263, ...,  0.00080626,
         0.0028481 ,  0.00539704],
       [-0.0017958 ,  0.00391856, -0.001328 , ..., -0.00570017,
        -0.00020655,  0.0002442 ],
       ...,
       [-0.00752427,  0.01470243,  0.00470035, ..., -0.01843934,
         0.00071943, -0.00453997],
       [-0.00404422,  0.0055398 ,  0.00066577, ..., -0.00553379,
         0.00169501, -0.00131452],
       [-0.0063858 ,  0.00674818,  0.00355475, ..., -0.01261365,
        -0.00032547,  0.00294473]], dtype=float32)
```

```
1 train_data_vecs.shape # 우리 shape도 찍어봐요
```

```
(1192, 100)
```

## 8. 챗봇 1 모델링

- 챗봇 1이라는 모델을 만들기 전, train\_test\_split를 이용하여, 학습 데이터와 테스트 데이터로 구분한다.
  - > X : 이전 단계에서 저장된 임베딩 벡터
  - > Y : intent 값들
- > 학습 데이터가 953건, 테스트 데이터가 239건으로 비교적 데이터 건수가 적음

```
[ ] 1  # X와 y 데이터 분리
     2  X = train_data_vecs
     3  y = train_df['intent']
     4
     5  # Train-Test split
     6  train_X, val_X, train_y, val_y = train_test_split(X, y, test_size=0.2, stratify=y, random_state=76)
```

```
[ ] 1  print(train_X.shape, val_X.shape, train_y.shape, val_y.shape)
```

```
(953, 100) (239, 100) (953,) (239,)
```

## 8. 챗봇 1 모델링

- LightGBM, RandomForest 모델을 학습하기에 앞서, `train_y(intent)`에 대한 데이터 불균형이 존재하므로 소수 클래스에 대해서 가중치를 더 부여하는 방법을 활용한다.

```
1 train_y.value_counts()
```

32	109
8	53
50	46
46	46
39	46
41	30
43	30
45	30
3	30
31	30
40	30
20	26
30	22
28	21
21	18
13	18
47	14
52	14
44	14
48	14
36	14
34	14
51	14

## 8. 챗봇 1 모델링

- LightGBM, RandomForest 모델을 학습하기에 앞서, **train\_y(intent)**에 대한 데이터 불균형이 존재하므로 소수 클래스에 대해서 가중치를 더 부여하는 방법을 활용한다.  
-> 소수 클래스에 대해서 더 가중치를 부여하는 것을 알 수 있다.

```
[35] 1 # 클래스 레이블과 해당 레이블의 빈도수
      2 labels = np.unique(train_y)
      3 class_weight = compute_class_weight(class_weight='balanced', classes=labels, y=train_y)
      4
      5 # 클래스 가중치 딕셔너리 생성
      6 class_weight_dict = {label: weight for label, weight in zip(labels, class_weight)}
```

```
1 class_weight_dict # 비교적 소수 클래스를 가중치 더 부여하는 것으로 알 수 있다.
```

```
{1: 1.634648370497427,
2: 1.634648370497427,
3: 0.5993710691823899,
4: 2.568733153638814,
5: 2.99685534591195,
6: 2.99685534591195,
7: 2.568733153638814,
8: 0.3392666429334283,
9: 2.568733153638814,
10: 2.568733153638814,
11: 2.568733153638814,
12: 2.99685534591195,
13: 0.9989517819706499,
14: 2.99685534591195,
15: 2.568733153638814,
16: 2.99685534591195,
17: 2.568733153638814,
18: 2.568733153638814,
19: 2.99685534591195,
20: 0.6915820029027576,
21: 0.9989517819706499,
22: 2.568733153638814,
23: 2.568733153638814,
24: 2.568733153638814,
25: 2.568733153638814,
26: 1.9979035639412999,
27: 1.7981132075471697,
28: 0.8562443845462714,
29: 1.3831640058055152,
```

```
30: 0.8173241852487135,
31: 0.5993710691823899,
32: 0.16496451445386878,
33: 1.284366576819407,
34: 1.284366576819407,
35: 1.284366576819407,
36: 1.284366576819407,
37: 1.3831640058055152,
38: 1.284366576819407,
39: 0.39089417555373257,
40: 0.5993710691823899,
41: 0.5993710691823899,
42: 1.3831640058055152,
43: 0.5993710691823899,
44: 1.284366576819407,
45: 0.5993710691823899,
46: 0.39089417555373257,
47: 1.284366576819407,
48: 1.284366576819407,
49: 1.284366576819407,
50: 0.39089417555373257,
51: 1.284366576819407,
52: 1.284366576819407,
53: 1.284366576819407}
```

## 8. 챗봇 1 모델링

- LightGBM 모델을 학습한다. (소수 클래스에 더 가중치를 부여하는 방향 옵션 적용)  
-> 정확도 약 58%로, 2개 질문을 던지면, 1개 대답은 잘한다는 뜻으로 받아들일 수 있다.

```
1 # LightGBM 분류기 생성
2 lgbmC = lgb.LGBMClassifier(
3     objective='multiclass',
4     num_class=53,
5     learning_rate=0.05,
6     n_estimators=100,
7     num_leaves=31,
8     class_weight= class_weight_dict,
9 )
10
11 # 학습
12 lgbmC.fit(train_X, train_y)
```

```
1 # 예측 및 검증
2 val_y_pred = lgbmC.predict(val_X)
3 accuracy = accuracy_score(val_y, val_y_pred)
4 print(f'Accuracy: {accuracy}')
```

Accuracy: 0.5815899581589958

## 8. 챗봇 1 모델링

- Random Forest 모델을 학습한다. (소수 클래스에 더 가중치를 부여하는 방향 옵션 적용)  
-> 정확도 약 56%로, 2개 질문을 던지면, 1개 대답은 잘한다는 뜻으로 받아들일 수 있다.

```
[45] 1 # RandomForest 학습
      2 clf = RandomForestClassifier(n_estimators=100,
      3                               class_weight=class_weight_dict,
      4                               random_state=75)
      5 clf.fit(train_X, train_y)
```

```
1 # 예측 및 검증
2 val_y_pred = clf.predict(val_X)
3 accuracy = accuracy_score(val_y, val_y_pred)
4 print(f'Accuracy: {accuracy}')
```

```
➤ Accuracy: 0.5648535564853556
```



## 8. 챗봇 1 모델링

- 원래 테스트 데이터로 챗봇 1 모델의 성능을 확인한다.  
-> 테스트 데이터를 Word2Vec 모델을 활용하여 자연어 처리에 특화된 숫자 형태로 변환한다.

```
[47] 1 # 입력 문장 벡터화하여 결국 토큰화된 텍스트를 자연어 처리에 특화된 숫자 형태로 변환하는 get_dataset() 함수  
2 # 혹시나 말합니다. test_vector는 2차원 넘파이 배열 형태입니다. 왜냐하면 X를 모델에 넣을 때 2차원 넘파이 배열이나 데이터프레임을 기대하기 때문이죠.  
3 test_vector = get_dataset(sentences=clean_test_questions, model=mv_model, num_features=100)
```

## 8. 챗봇 1 모델링

- 테스트 데이터를 이용하여 챗봇 1 모델의 성능을 확인한다.  
-> 정확도 약 48프로, 사용자가 2개 질문을 던지면 1개라도 캐치할지 말지 정도의 성능 같다.

```
1 predict_intents = lgbmC.predict(test_vector) # 예측 intent들
2 test_y = test_df['intent'].astype(int) # 왜 object인지 모르겠지만 일단 int로 변경해야 함
3
4 # 실제값과 예측값을 비교하여 정확도를 확인한다.
5 print('lgbm 모델 정확도 : ', accuracy_score(test_y, predict_intents)) # 입력 데이터 형태는 Series나 1차원 넘파이 배열을 기대한다.
```

lgbm 모델 정확도 : 0.4811320754716981

## 8. 챗봇 1 모델링

- 사용자가 임의로 질문을 던졌을 때 답변을 하는 것을 함수로 정리한다.

```
1 def get_answer1(sentence):
2     # 0. 맞춤법 정리
3     # 원래는 맞춤법 정리도 하려고 했지만, Hanspell이 말을 안들어서(서버 오류인지... 잘 모르겠음)
4     # 저도 아직 못하고 있었답니다. 관심이 있으신분들은 한번 해보실쇼. 하지만 안해도 크게 지장은 없습니다.
5
6     # 1. 특수 문자 제거
7     removal_list = " ' < > \" ' ' . # \" ' . △ ● ■ () # >> ` / ~ = . < > , ? ! [ ] ... ◆ %"
8     removal_list += string.punctuation
9
10    sentence = re.sub("[.,#\"'\"' ' \" '[?]", "", sentence)
11    sentence = re.sub("[^가-힣0-9a-zA-Z\\s]", " ", sentence)
12    sentence = re.sub("\\s+", " ", sentence)
13
14    sentence = sentence.translate(str.maketrans(removal_list, ' '*len(removal_list)))
15    sentence = sentence.strip()
16
17    # 1.5. 텍스트를 형태소 단위로 분리
18    sentence = okt.morphs(sentence)
19
```

## 8. 챗봇 1 모델링

- 사용자가 임의로 질문을 던졌을 때 답변을 하는 것을 함수로 정리한다.

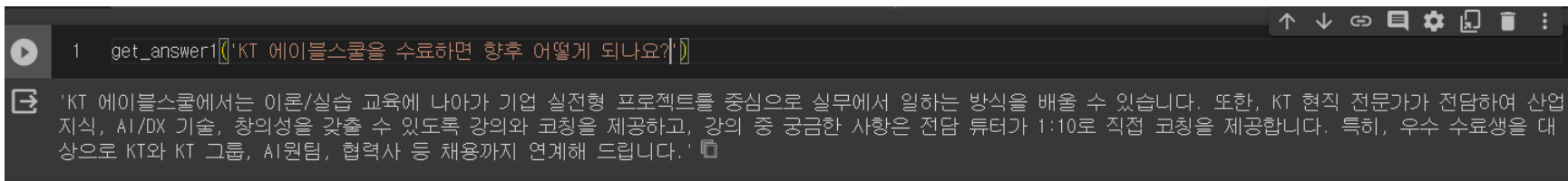
```
# 2. 한글 불용어 처리(자신만의 한글 불용어 txt 파일을 만들어서 경로 바꾸고 적용해보십시오!, 만약 하고싶지 않으시면, 2번은 주석 처리 하십시오!)
korean_stopwords = []
for fileName in ['koreanStopwords.txt', 'stopwords.txt', 'stopword2.txt', 'stopword3.txt', 'stopword4.txt']:
    f = open(f'/content/drive/MyDrive/미니프로젝트6차part1/데이터/한국어불용어사전/{fileName}', 'r')
    lines = f.readlines()
    for line in lines:
        line = line.replace('\n', '')
        line = line.replace('\t', '')
        korean_stopwords.append(line)
    f.close()
korean_stopwords = list(set(korean_stopwords))
sentence = [morph for morph in sentence if morph not in korean_stopwords] # 한글 불용어 제거

# 3. sentence를 2차원 리스트로 만들 -> 왜냐하면 get_dataset() 함수를 사용할 때 1번째 매개변수가 2차원 리스트 형태를 기대하기 때문
sentence = [sentence]

# 4. get_dataset() 함수를 사용한다. -> 입력문장 벡터화를 하여 결국 토큰화된 텍스트를 자연어 처리에 특화된 숫자 형태로 변환하는 것이죠.
# 혹시나 말합니다. sentence는 2차원 넘파이 배열 형태입니다. 왜냐하면 X를 모델에 넣을 때 2차원 넘파이 배열이나 데이터프레임을 기대하기 때문이죠.
sentence = get_dataset(sentences=sentence, model=mv_model, num_features=100)

# 5. 우리가 만들었던 머신러닝 모델(LightGBM, RandomForest)에 predict 한다.
predict_intent = (lgbmC.predict(sentence))[0]

# 6. 예측한 intent에 따른 Answer를 리스트로 변환하고 랜덤하여 하나만 출력한다.
return random.choice(df.loc[df['intent'] == predict_intent, 'A'].tolist())
```



## 9. 챗봇 2 모델링

- 상세 요구사항

- > Type(일상대화, 0, 에이블스쿨 지원 Q&A, 1)을 분류하는 Embedding + LSTM 모델을 만든다.

- > FastText 모델을 생성하여 사용자가 던진 input 문장과 train의 임베딩 벡터 저장

- > input 문장과 train 임베딩 벡터와 코사인 유사도 계산

- > 가장 유사도가 높은 질문의 intent 찾아

- > 해당 intent의 답변 중 무작위로 하나를 선정하여 답변하기

## 9. 챗봇 2 모델링

- `clean_train_questions`, `clean_test_questions`를 자연어 처리에 특화된 숫자 형태로 변환하기 위한 작업을 수행한다.  
-> `get_dataset()`, `get_sent_embedding()`을 활용하던 기존의 방식과 달리 새로운 방식으로 적용

```
[95] 1  # 각각의 토큰에 인덱스 부여하는 토크나이저 선언
      2  tokenizer = Tokenizer()
      3
      4  # .fit_on_texts 이용하여 토크나이저 만들기
      5  tokenizer.fit_on_texts(clean_train_questions) # 입력 변수는 2차원 리스트를 기대합니다.
```


```
▶ 1  # 전체 토큰의 수 확인
   2  total_tokens = len(tokenizer.word_index)
   3  print('전체 토큰의 수:', total_tokens)
```

```
📄 전체 토큰의 수: 1283
```

## 9. 챗봇 2 모델링

- `clean_train_questions`, `clean_test_questions`를 자연어 처리에 특화된 숫자 형태로 변환하기 위한 작업을 수행한다.  
-> `get_dataset()`, `get_sent_embedding()`을 활용하던 기존의 방식과 달리 새로운 방식으로 적용

```
1 # 전체 토큰의 수가 vocab 사이즈가 됨
2 vocab_size = total_tokens
3 print("Vocab size:", vocab_size)
4
5 # fit_on_texts을 위에서 한번만 해도 되지만, vocab 사이즈를 확인하고 줄이거나 하는 시도를 할 수도 있기에 다시 수행
6 # fit_on_texts을 위에서 한번만 해도 되지만, vocab 사이즈를 확인한 후에 필요에 따라 특정 단어를 제외하거나, 빈도수
7 tokenizer = Tokenizer(num_words=vocab_size, oov_token='<OOV>') # vocab 사이즈를 vocab_size(1283)로 제한
8 tokenizer.fit_on_texts(clean_train_questions) # 입력 변수는 2차원 리스트를 기대합니다.
9
10 # .texts_to_sequences : 토큰나이징 된 데이터를 가지고 모두 시퀀스로 변환
11 # 이렇게 변환된 시퀀스 데이터는 각 문장을 구성하는 단어들이 해당 단어의 정수 인덱스로 변환된 리스트로 표현됩니다.
12 # 이 시퀀스 데이터는 이후에 패딩을 적용하여 모든 문장이 동일한 길이를 갖게 하고, 신경망 모델에 입력으로 사용될 수
13 train_sequences = tokenizer.texts_to_sequences(clean_train_questions) # 입력 변수는 2차원 리스트를 기대한다.
14 test_sequences = tokenizer.texts_to_sequences(clean_test_questions) # 입력 변수는 2차원 리스트를 기대한다.
```

 Vocab size: 1283

## 9. 챗봇 2 모델링

- clean\_train\_questions, clean\_test\_questions를  
자연어 처리에 특화된 숫자 형태로 변환하기 위한 작업을 수행한다.  
-> get\_dataset(), get\_sent\_embedding()을 활용하던 기존의 방식과 달리 새로운 방식으로 적용

```
[105] 1 # 문장 길이 분포를 바탕으로 MAX_SEQUENCE_LENGTH 결정
      2 MAX_SEQUENCE_LENGTH = 20 # 문장별 토큰 수를 최대 1000으로 설정하면 될 듯 싶다...
```

```
1 # 시퀀스 데이터의 길이를 MAX_SEQUENCE_LENGTH에 맞추어 조정
2 X_train = pad_sequences(train_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
```

```
[109] 1 X_train
      array([[336, 84, 440, ..., 0, 0, 0],
             [440, 441, 0, ..., 0, 0, 0],
             [ 84, 336, 260, ..., 0, 0, 0],
             ...,
             [ 3, 439, 2, ..., 0, 0, 0],
             [117, 4, 3, ..., 0, 0, 0],
             [ 3, 136, 335, ..., 0, 0, 0]], dtype=int32)
```



## 9. 챗봇 2 모델링

- Y(Target)을 정한다.
  - > 일상 대화(0), Aivle School 지원 Q&A 대화(1) 데이터 불균형이 발생하므로 이에 대한 대책을 마련한다.

```
[107] 1  y_train = train_df['type']
```

```
[108] 1  y_train.value_counts()
```

```
1    746  
0    446  
Name: type, dtype: int64
```

## 9. 챗봇 2 모델링

- Embedding + LSTM 모델을 학습하기에 앞서 소수의 클래스에 더 가중치를 부여하도록 미리 준비한다.  
-> 0이 더 높은 가중치를 부여됐음을 알 수 있다.

```
[111] 1 # 클래스 레이블과 해당 레이블의 빈도수
      2 labels = np.unique(y_train)
      3 class_weight = compute_class_weight(class_weight='balanced', classes=labels, y=y_train)
      4
      5 # 클래스 가중치 딕셔너리 생성
      6 class_weight_dict = {label: weight for label, weight in zip(labels, class_weight)}
```

```
▶ 1 class_weight_dict # 비교적 수가 적은 0에 더 가중치를 부여했다.
```

```
📄 {0: 1.336322869955157, 1: 0.7989276139410187}
```

## 9. 챗봇 2 모델링

- clean\_train\_questions, claen\_test\_questions를  
자연어 처리에 특화된 숫자 형태로 변환하기 위한 작업을 수행한다.  
-> get\_dataset(), get\_sent\_embedding()을 활용하던 기존의 방식과 달리 새로운 방식으로 적용

```
[114] 1  # 시퀀스 데이터의 길이를 MAX_SEQUENCE_LENGTH에 맞추어 조정
      2  X_test = pad_sequences(test_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
```

```
[115] 1  X_test
      array([[ 16,  680,  749, ...,  0,  0,  0],
             [ 84,  686,  687, ...,  0,  0,  0],
             [444,  445,   25, ...,  0,  0,  0],
             ...,
             [105,  106,   36, ...,  0,  0,  0],
             [117,    4,    3, ...,  0,  0,  0],
             [  3, 1266,  439, ...,  0,  0,  0]], dtype=int32)
```

## 9. 챗봇 2 모델링

- Y(Target)을 정한다.

```
[ ] 1 y_test = test_df['type'].astype(int) # 이게 왜 object인지는 모르겠지만 int로 변환해줍니다
```

```
▶ 1 X_test.shape, y_test.shape
```

```
↵ ((106, 20), (106,))
```

2) 모델링

## 9. 챗봇 2 모델링

- Embedding + LSTM를 이용하여 모델을 학습한다.  
-> 이진 분류 모델(0, 1 분류)이고, 학습할 때 소수 클래스 0에 가중치를 더 부여하도록 설정한다.

```
1 # 모델 구축
2 model1 = Sequential()
3 model1.add(Embedding(input_dim=vocab_size + 1, output_dim=64, input_length=20))
4 model1.add(LSTM(64, return_sequences=True))
5 model1.add(Flatten()) # Embedding Layer 사용후 Flatten()을 해야 한다.
6 model1.add(Dense(64, activation='swish'))
7 model1.add(Dense(16, activation='swish'))
8 model1.add(Dense(1, activation='sigmoid'))
9
10 # 모델 컴파일
11 model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
12
13 # 모델 학습
14 model1.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2, class_weight=class_weight_dict)
```

## 9. 챗봇 2 모델링

- 테스트 데이터로 모델의 정확도가 약 99% 정도 나왔다.  
-> 즉 단계적 모델링에서 일상 대화(0), Aivle School 지원 Q&A 대화(1)를 판별한다고 볼 수 있다.

```
1 predictions = model1.predict(X_test) # sigmoid() 함수를 거치기 때문에 0 ~ 1 범위의 값들이 존재할 것이다.
2 predicted_classes = np.where(predictions > 0.5, 1, 0).ravel() # ravel()를 사용해서 1차원 넘파이 배열로 변환
3
4 # 모델 평가
5 print("정확도: ", accuracy_score(y_test, predicted_classes)) # Series나 1차원 넘파이 배열 형태면 됩니다.
```

4/4 [=====] - 1s 3ms/step  
정확도: 0.9905660377358491

## 9. 챗봇 2 모델링

- 0과 1를 분류한 후, 사용자가 던진 input 질문(Test Data도 될 수 있음)과 train의 질문들을 FastText 모델을 이용하여 임베딩 하기 위한 작업을 한다.
  - > 나중에 코사인 유사도로 비교할 예정
  - > FastText 모델을 정의하는 부분이다.

```
1 from gensim.models.fasttext import FastText
2 import gensim.models.word2vec
3
4 ft_model = FastText(clean_train_questions, # 입력 형태로 2차원 리스트를 기대한다.
5                     vector_size=200,
6                     window=3,
7                     min_count=2,
8                     workers=3,
9                     sg=0)
```

Loading...

## 9. 챗봇 2 모델링

- 0과 1를 분류한 후, 사용자가 던진 input 질문(Test Data도 될 수 있음)과 train의 질문들을 FastText 모델을 이용하여 임베딩 하기 위한 작업을 한다.
  - > 나중에 코사인 유사도로 비교할 예정
  - > train에 있는 질문들을 FastText 모델을 이용하여 임베딩 벡터화 한후 'ft\_vec' 변수에 추가한다.

```
[ ] 1 rr = get_dataset(sentences=clean_train_questions, model=ft_model, num_features=ft_model.vector_size)

[ ] 1 train_df['ft_vec'] = list(rr) # FastText로 임베딩한 결과물을 train_df의 새로운 변수 'ft_vec'에 추가
```



## 9. 챗봇 2 모델링

- 테스트 데이터를 이용하여 단계적 모델링을 거쳐 최종 intent를 예측하고 그에 따른 답변을 도출한다.
  - > 1. 0과 1를 분류하는 **Embedding + LSTM** 모델을 사용한다.
  - > 2. 사용자가 입력한 input 문장을 FastText로 이용하여 임베딩한후, train의 질문 데이터를 FastText를 이용하여 임베딩한 결과와 코사인 유사도를 체크한다.
  - > 3. 코사인 유사도가 가장 높은 문장의 intent를 확인한다.
  - > 4. intent에 따른 A(답변)을 랜덤하게 1개 골라 출력한다.

```
[ ] 1  # 실제값과 예측값
    2  y_true = test_df['intent'].astype(int) # 실제값, Series 형태
    3  y_pred = [] # 예측값
```

## 9. 챗봇 2 모델링

- 테스트 데이터를 이용하여 단계적 모델링을 거쳐 최종 intent를 예측하고 그에 따른 답변을 도출한다.
  - > 1. 0과 1를 분류하는 **Embedding + LSTM** 모델을 사용한다.
  - > 2. 사용자가 입력한 input 문장을 FastText로 이용하여 임베딩한후, train의 질문 데이터를 FastText를 이용하여 임베딩한 결과와 코사인 유사도를 체크한다.
  - > 3. 코사인 유사도가 가장 높은 문장의 intent를 확인한다.
  - > 4. intent에 따른 A(답변)을 랜덤하게 1개 골라 출력한다.

```
1 predictions = model1.predict(X_test) # sigmoid() 함수를 거치기 때문에 0 ~ 1 범위의 값들이 존재할 것이다.
2 predicted_classes = np.where(predictions > 0.5, 1, 0).ravel() # flatten()를 사용해서 1차원 넘파이 배열로 변환
3
4 # 모델 평가
5 print("정확도: ", accuracy_score(y_test, predicted_classes)) # Series나 1차원 넘파이 배열 형태면 됩니다.
```

```
4/4 [=====] - 0s 3ms/step
정확도: 0.9905660377358491
```

## 9. 챗봇 2 모델링

- 테스트 데이터를 이용하여 단계적 모델링을 거쳐 최종 intent를 예측하고 그에 따른 답변을 도출한다.
  - > 1. 0과 1를 분류하는 Embedding + LSTM 모델을 사용한다.
  - > 2. 사용자가 입력한 input 문장을 FastText로 이용하여 임베딩한후, train의 질문 데이터를 FastText를 이용하여 임베딩한 결과와 코사인 유사도를 체크한다.
  - > 3. 코사인 유사도가 가장 높은 문장의 intent를 확인한다.
  - > 4. intent에 따른 A(답변)을 랜덤하게 1개 골라 출력한다.

```
1 # 예측한 Type 클래스 106개를 for문으로 돈다.
2 for idx, pc in enumerate(predicted_classes):
3     txt = test_df.iloc[idx]['Q'] # test_df에 있는 'Q'를 순차적으로 가져온다.
4     txt = [okt.morphs(txt)] # 형태소 분석 후 2차원 리스트로 만들
5     txt = get_dataset(sentences=txt, model=ft_model, num_features=ft_model.vector_size) # FastText를 이용하여 임
6
7     if pc == 0: # 예측한 Type 클래스가 0이었을 때
8         temp_df = train_df.loc[train_df['type'] == 0]
9     else: # 예측한 Type 클래스가 1이었을 때
10        temp_df = train_df.loc[train_df['type'] == 1]
11
12 # 사용자가 입력한 질문 txt와 train에 있는 질문을 FastText로 임베딩, 벡터화 한 'ft_vec'과 코사인 유사도를 체크
13 temp_df['cosine_ft'] = temp_df['ft_vec'].apply(lambda x : cosine_similarity(txt, [x])[0][0])
14
15 # 코사인 유사도 결과, 가장 높은 값으로 내림차순 정렬 후 가장 맨 1번째의 Intent를 구한다.
16 temp_df = temp_df.sort_values(by='cosine_ft', ascending=False, )
17 head_intent = temp_df.head(1)['intent'].iloc[0] # Intent를 찾기 위함
18
19 # y_pred에 예측값을 append한다.
20 y_pred.append(head_intent)
```

## 9. 챗봇 2 모델링

- 테스트 데이터를 이용하여 단계적 모델링을 거쳐 최종 intent를 예측하고 그에 따른 답변을 도출한다.
  - > 1. 0과 1를 분류하는 Embedding + LSTM 모델을 사용한다.
  - > 2. 사용자가 입력한 input 문장을 FastText로 이용하여 임베딩한후, train의 질문 데이터를 FastText를 이용하여 임베딩한 결과와 코사인 유사도를 체크한다.
  - > 3. 코사인 유사도가 가장 높은 문장의 intent를 확인한다.
  - > 4. intent에 따른 A(답변)을 랜덤하게 1개 골라 출력한다.
    - > 실제로 여기서는 실제 intent와 최종 예측 intent를 비교하여 정확도를 확인했다.
    - > 챗봇 1에 약 49% 정확도보다 약 70% 정확도를 보이고 있어 성능 개선했다고 볼 수 있다.
    - 하지만 범용적으로 사용하기에는 아직 많이 부족한 수준이라 판단할 수 있다.

```
1 # 실제값과 예측값을 비교해서 정확도 계산
2 print('정확도 : ', accuracy_score(y_true, np.array(y_pred))) # 1차원 넘파이 배열 혹은 Series 여야 한다.
```

정확도 : 0.6981132075471698

## 9. 챗봇 2 모델링

- 사용자가 임의로 입력한 input 문장을 가지고 단계별 모델링하여 최종 intent를 예측하고, 어떻게 답변이 나오는지 흐름을 함수로 정리했다.  
-> 특수 문자 제거

```

1 def get_answer2(question):
2     # 0. 맞춤법 정리
3     # 원래는 맞춤법 정리도 하려고 했지만, Hanspell이 말을 안들어서(서버 오류인지... 잘 모르겠음)
4     # 저도 아직 못하고 있었답니다. 관심이 있으신분들은 한번 해보십쇼. 하지만 안해도 크게 지장은 없습니다.
5
6     # 1. 특수 문자 제거
7     removal_list = " \" ' < > ` ~ = . < > . ? ! [ ] ... ◆ %"
8     removal_list += string.punctuation
9
10    question_1 = re.sub("[.,#\"'\" \" \" \" !?]", "", question)
11    question_2 = re.sub("[^가-힣0-9a-zA-Z\\ws]", " ", question_1)
12    question_3 = re.sub("\\ws+", " ", question_2)
13
14    question_4 = question_3.translate(str.maketrans(removal_list, ' ' * len(removal_list)))
15    question_5 = question_4.strip()
16
17    # 1.5. 텍스트를 형태소 단위로 분리
18    question_6 = okt.morphs(question_5)
19

```

## 9. 챗봇 2 모델링

- 사용자가 임의로 입력한 input 문장을 가지고 단계별 모델링하여 최종 intent를 예측하고, 어떻게 답변이 나오는지 흐름을 함수로 정리했다.  
-> 한글 불용어 처리

```
# 2. 한글 불용어 처리(자신만의 한글 불용어 txt 파일을 만들어서 경로 바꾸고 적용해보십시오!, 만약 하고싶지 않으면  
korean_stopwords = []  
for fileName in ['koreanStopwords.txt', 'stopwords.txt', 'stopword2.txt', 'stopword3.txt', 'stopword4.txt']:  
    f = open(f'/content/drive/MyDrive/미니프로젝트6차part1/데이터/한국어불용어사전/{fileName}', 'r')  
    lines = f.readlines()  
    for line in lines:  
        line = line.replace('\n', '')  
        line = line.replace('\t', '')  
        korean_stopwords.append(line)  
    f.close()  
korean_stopwords = list(set(korean_stopwords))  
question_7 = [morph for morph in question_6 if morph not in korean_stopwords] # 한글 불용어 제거
```

## 9. 챗봇 2 모델링

- 사용자가 임의로 입력한 input 문장을 가지고 단계별 모델링하여 최종 intent를 예측하고, 어떻게 답변이 나오는지 흐름을 함수로 정리했다.

-> Embedding + LSTM이 적용된 Model1에 입력 형태로 맞추고, predict 하여 일상 대화(0)이냐, Aivle School 지원 Q&A 대화(1) 인지를 판별한다.

```
# 3. sentence를 2차원 리스트로 만들 -> Embdding + LSTM를 이용해서 Type(0), Type(1)를 예측하는 모델에 적합한 9
question_8 = [question_7]
question_9 = tokenizer.texts_to_sequences(question_8)
question_10 = pad_sequences(question_9, maxlen=MAX_SEQUENCE_LENGTH, padding='post')

# 4. 모델 1에 predict하여 예측 클래스가 Type(0)인지 Type(1)인지 확인한다.
predictions = model1.predict(question_10) # sigmoid() 함수를 거치기 때문에 0 ~ 1 범위의 값으로 존재할 것이다
pred_class = 1 if predictions > 0.5 else 0 # 일반적으로 0.5 초과이면 1로 판단, 미만이면 0으로 판단한다. 우리는
```

## 9. 챗봇 2 모델링

- 사용자가 임의로 입력한 input 문장을 가지고 단계별 모델링하여 최종 intent를 예측하고, 어떻게 답변이 나오는지 흐름을 함수로 정리했다.

-> FastText 모델을 사용하여, 사용자가 입력한 input 문장과 train에 있었던 질문 데이터를 임베딩 벡터화 한다.

```
# 5. fastText 모델을 도입한다.
ft_model = FastText(clean_train_questions, # 입력 형태로 2차원 리스트를 기대한다.
                    vector_size=200,
                    window=3,
                    min_count=2,
                    workers=3,
                    sg=0)

# 6. train에 있었던 질문에 대해서 FastText를 이용해서 임베딩, 벡터화 하는 작업을 한다.
rr = get_dataset(sentences=clean_train_questions,
                model=ft_model,
                num_features=ft_model.vector_size)
train_df['ft_vec'] = list(rr) # FastText로 임베딩한 결과물을 train_df의 새로운 변수 'ft_vec'에 추가

# 7. 사용자가 입력한 질문에 대해서 FastText를 이용해서 임베딩, 벡터화 작업 한다.
question_11 = get_dataset(sentences=question_8, model=ft_model, num_features=ft_model.vector_size)
```



## 9. 챗봇 2 모델링

- 사용자가 임의로 입력한 input 문장을 가지고 단계별 모델링하여 최종 intent를 예측하고, 어떻게 답변이 나오는지 흐름을 함수로 정리했다.
- > 임베딩 된 사용자 input 문장과 train에 있었던 질문 문장들과 코사인 유사도 계산
- > 가장 코사인 유사도가 높은 행의 intent를 파악한다.
- > intent에 따른 대답(A)을 랜덤하게 1개만 출력한다.

```
# 8. 예측 Type 클래스를 0으로 했으면 Type 클래스가 0인 데이터에서만 코사인 유사도 계산, 반대로 Type 클래스를
temp_df = train_df.loc[train_df['type'] == pred_class]
temp_df['cosine_ft'] = temp_df['ft_vec'].apply(lambda x : cosine_similarity(question_11, [x])[0][0])

# 9. 코사인 유사도 결과, 가장 높은 값으로 내림차순 정렬 후 가장 맨 1번째의 intent를 구하고 그에 대한 Answer를
temp_df = temp_df.sort_values(by='cosine_ft', ascending=False, )
head_intent = temp_df.head(1)['intent'].iloc[0] # intent를 찾기 위함
return random.choice(temp_df.loc[temp_df['intent'] == head_intent, 'A'].tolist())
```

## 9. 챗봇 2 모델링

- 사용자가 입력한 input 문장을 챗봇 2를 통해 대답을 살펴본다.

```
1 test_sample = test_df.sample(1) # 테스트 데이터 중에 임의로 1개 선택
2 print('*' * 100)
3 print('질문 : ', test_sample['Q'].values[0])
4 print('*' * 100)
5 print('대답 : ', get_answer2(test_sample['Q'].values[0])) # '차 너무 막혀' 이런 식으로 데이터가 들어간다.
6 print('*' * 100)
```

```
*****
질문 : 소개팅 해볼까
*****
1/1 [=====] - 0s 19ms/step
대답 : 로맨틱하네요.
```

## 10. 챗봇 1, 챗봇 2 사용

- 사용자가 입력한 input 문장을 챗봇 1, 챗봇 2를 통해 대답을 살펴본다.
  - > 챗봇 1은 약 49퍼의 정확도를, 챗봇 2는 약 70퍼의 정확도를 보이고 있다.
  - > 대체로 챗봇2를 사용하는 것이 권장된다.
  - > 하지만 실무적으로 투입될 정도의 모델은 아닌 것으로 판단한다.
  - > FastText 모델 말고도, Bert와 같은 모델로 챗봇을 구현해볼 필요성이 생기게 되었다.

```
*****
질문 : 비 전공자 도 수업 따라갈 있을까요
*****
챗봇 1 대답 : 비전공자도 SW개발에 관심과 열정이 있으신 분들은 충분히 따라 갈 수 있습니다. 이 경우 데이터 사이언스 기반:
*****
1/1 [=====] - 0s 20ms/step
챗봇 2 대답 : 비전공자도 SW개발에 관심과 열정이 있으신 분들은 충분히 따라 갈 수 있습니다.
이 경우 데이터 사이언스 기반으로 산업 현장에 AI를 접목하고 DX를 이끄는 DX컨설턴트 Track을 추천 드립니다.
SW기초 지식 및 코딩역량을 보유하고 있고 개발자로 성장하고 싶은 분께는 AI개발자 Track을 추천 드립니다.

파이썬 프로그래밍 사용법과 AI 개념에 대한 기초 이해를 공부하고 오시면 도움이 되실 것 같습니다.
*****
<ipython-input-136-8a37418a5170>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-temp-df
temp_df['cosine_ft'] = temp_df['ft_vec'].apply(lambda x : cosine_similarity(question_t1, [x])[0][0])
```