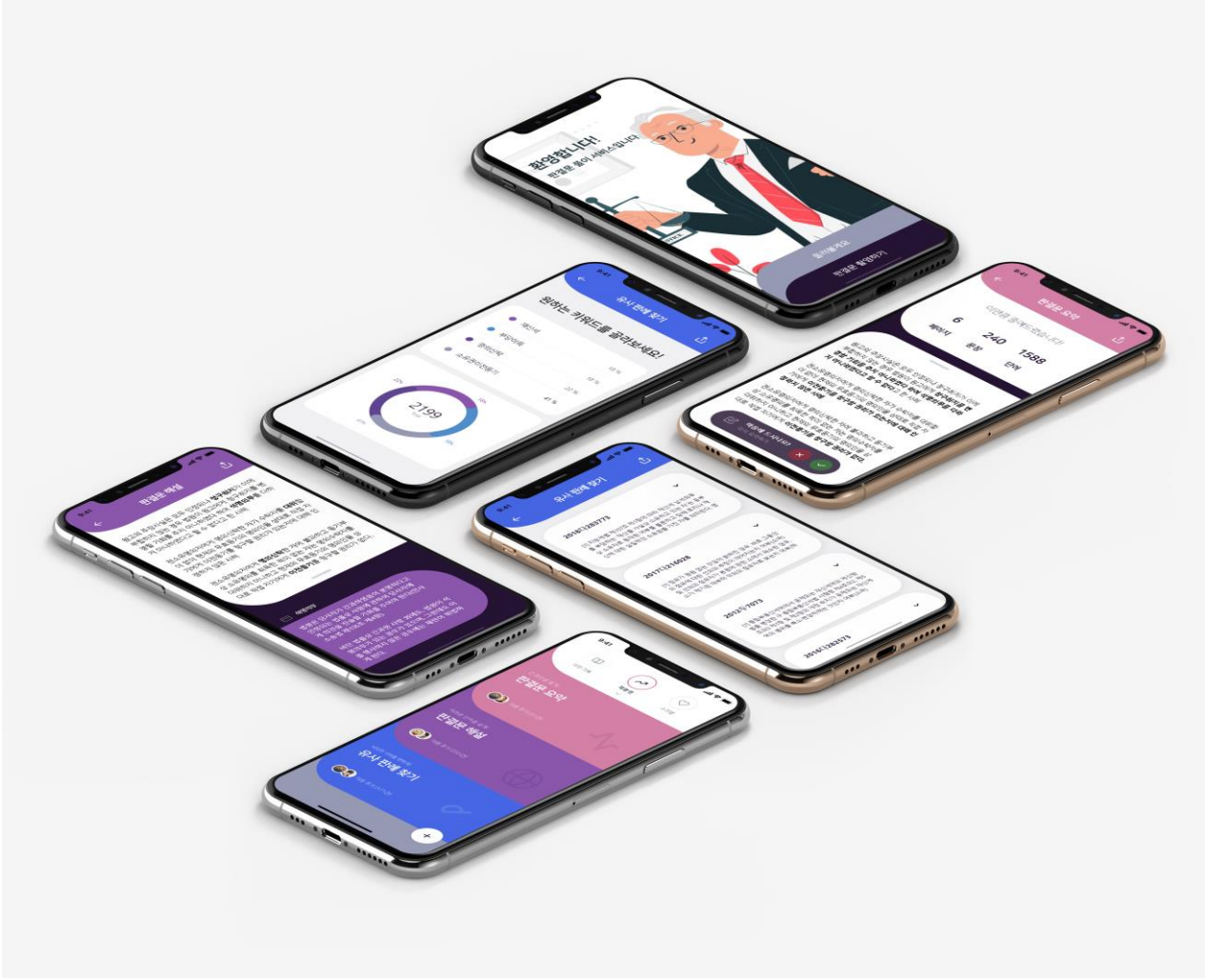
1. 인공지능 학습용 데이터 활용 아이디어 제목

인공지능 학습용 데이터 활용 아이디어 공모전 참가 신청서

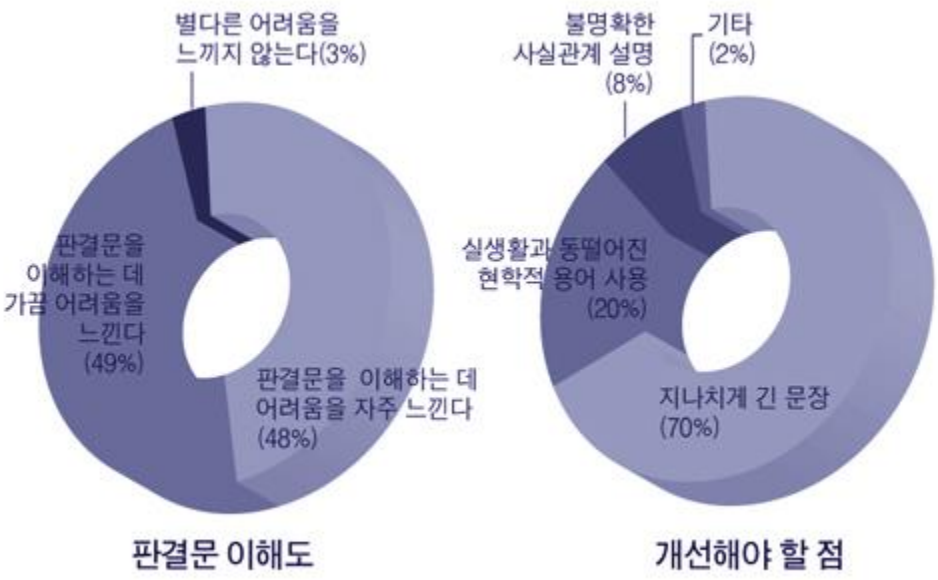
법률(law)의 장벽을 **낮게(low)**하는 서비스, **'로우로우(Law-Low)'**입니다.



2. 인공지능 학습용 데이터 활용 아이디어 내용

1. 제안 배경

재판에 대한 판결 내용 기재 문서인 판결문은 법률 지식이 없는 일반인이 보기에는 문장이 길고 어려운 용어가 많습니다. 실제로, 판결문을 이해할 수 없어 법무사나 변호사를 찾아가거나 네이버 지식인에 판결문의 뜻을 질문하는 사례를 심심치 않게 찾아볼 수 있습니다. 나아가, 2013년 로스쿨 학생들을 대상으로 한 인터뷰[1]에서도 97%의 응답자가 판결문의 이해에 어려움을 느낀다고 하였으며, 90%의 응답자가 긴 문장과 어려운 법령용어를 판결문 난해함의 원인으로 꼽았습니다. 이러한 판결문의 긴 문장과 어려운 법령용어의 문제는 2020년에도 여전히 해결되지 못하고 있습니다.[2]



[그림 1] 판결문의 이해도와 개선해야 할 점 (출처:법률신문)

[1] 최영길 기자,『로스쿨생들도 이해 못하는 '판결문' 많다』, 법률신문, 2013.06.11

[2] 류정민 기자,『판결문이 '외계어'로 불리는 이유』, 아시아경제, 2020.02.13

2. 서비스 개요

(1) 개설

판결문의 긴 문장과 어려운 법령용어를 해결하기 위해 본 서비스는 판결문의 길이를 짧게, 어려운 용어는 쉽게 이해할 수 있도록 하는데 중점을 두고 있습니다. 추가적으로 유사한 판례에 대해서 사용자에게 제시함으로써 판결문에 대한 이해를 보다 높이고자 하였습니다.

(2) 기능

① **<요약 서비스 - 긴 것을 짧게!>**

판결문을 요약을 합니다. 특정 주요 문장을 뽑는 추출 요약이 아닌 원문으로부터 새로운 내용이 요약된 생성 요약을 사용하여 원문 내용에 대한 전반적인 이해를 돕습니다.

② **<법령용어 서비스 - 어려운 것을 쉽게!>**

판결문에는 어려운 법률용어가 많습니다. 판결문을 읽다가 모르는 법률용어가 나오는 경우 터치 한번으로 법령용어의 뜻을 확인할 수 있습니다.

③ **<유사 판례 추천 서비스 – 비슷한 경우를!>**

원문에서 주요 키워드를 뽑아 후보군을 사용자에게 제공하고, 사용자가 선정한 키워드에 기반한 유사 판례를 유사도가 높은 순서대로 제시합니다.

(3) 서비스 구동 예시

서비스 프로토타입 구동 영상을 함께 첨부하였으니 확인 부탁드립니다. 영상 재생이 불가하신 경우에는 아래의 URL 로 접속하시면 됩니다.

링크 : https://youtu.be/KG4cad1-e8I

3. 국내외 법률 서비스와의 차별성

(1) 법률 관련 인공지능 활용 사례

- 국내

① <유렉스>는 지능형 법률추론 검색엔진으로 AI 기술을 이용해 법령/판례에 대한 검색을 수행합니다. 웹 서비스 이용 시 AI 분석을 통한 주요문장, 조항 추천 기능을 시각적으로 보여줍니다.

② <법률메카>는 지능형 법률 QA 시스템으로 법률에 관한 질문을 입력하면 자연어 처리를 통해 적절한 정답이나 유사사례를 제공하며, QA 를 바탕으로 변호사 추천까지 도와줍니다.

③ <로톡>은 변호사를 연결해주는 플랫폼으로, 최근에는 형량 예측 서비스를 제공하고 있습니다.

- 해외

① <DoNotPay>는 금전 거래에 대응하는 인공지능 법률 봇입니다. 불법 주정차 과태료 및 유료앱 결제 환불에 관한 사항들을 주로 다루고 있습니다.
② <Robot lawyer LISA>는 계약서 작성의 중재자 역할을 하는 인공지능 법률 봇입니다. 최근에는 부동산 계약서로 기능을 확대하고 있습니다.
③ <ROSS>는 변호사들이 법정에 나서기 전에 패소할 수 있는 요인에 대해서 인공지능이 사전적으로 검토하고 수정을 조언하는 봇입니다.
④ <Billy Bot>는 별도의 전문적인 법률 사항을 다루기보다는 법무법인의 업무 일정을 관리해주는 봇입니다. <Automio>는 변호사들의 의미 없는 반복적인 업무를 줄여주는 봇입니다.
⑤ <소결> 위의 서비스는 대부분 변호사의 업무를 대체 및 자동화하는 것에만 초점을 맞추고 있습니다. DoNotPay 의 경우는 일상 생활의 법 문제를 해결해주나, 일반인의 법률 문서 검토에 대해서는 보조하고 있지 못합니다.

(2) 본 서비스의 차별성

① OCR 알고리즘 도입

3 가지의 서비스와 비교를 해보았을 때, 본 서비스는 OCR 을 사용하여 판결문을 직접 해석해준다는 점에서 차별성을 가집니다. 기존 서비스들의 경우, 직접 판결문 속 법령 용어나 문장, 사건번호를 검색함으로써 그 뜻풀이를 해주게 되지만 본 서비스에서 도입하는 기능의 경우, 그런 번거로움 없이 판결문 전체를 사진을 찍는 것만으로 모든 법령 용어, 문장 등을 해석해주기 때문에 차별성을 가지며 사용자의 편의성 측면에서도 높은 강점을 지니고 있습니다.

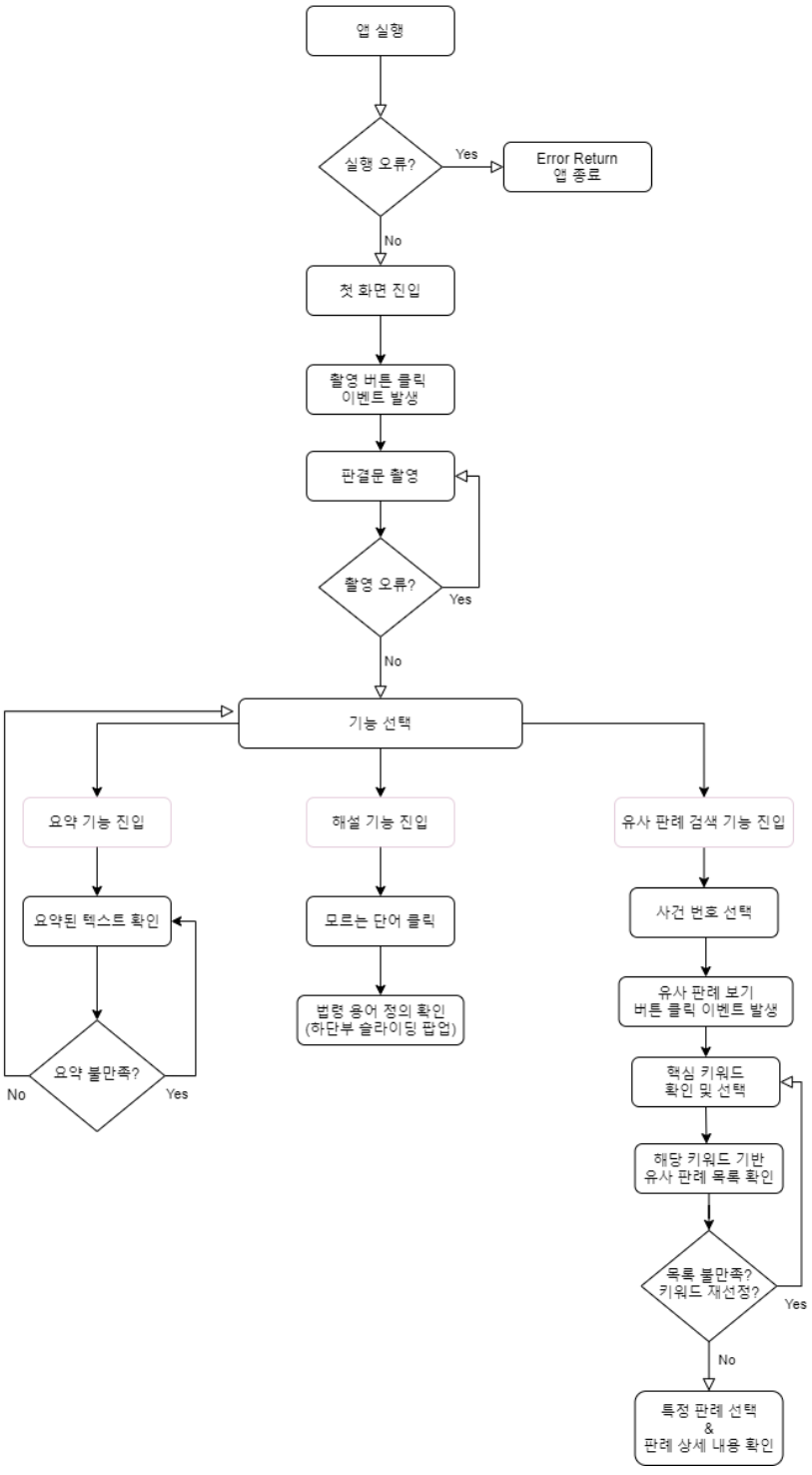
② 판결문 요약 보기 기능 제공

판결문의 길이는 대개 300 자 이상입니다. 글의 요지와 흐름을 파악하고 긴 글을 읽는 것이 이해도가 높습니다. 따라서 긴 판결문의 요지와 흐름을 파악할 수 있도록 요약 기능을 제공하고 있습니다. 하지만 어떠한 국내 서비스에서도 판결문을 '요약'해주는 기능은 없습니다. 유렉스의 주요문장, 조항 추천과 같은 AI 분석은 Background Color 를 이용해 시각화해주는 작업일 뿐입니다. 요약과는 관계가 없는 기능이라고 할 수 있습니다.

③ 사용자 맞춤형 서비스

본 서비스는 모든 기능들을 사용자에게 편의성을 제공하는 맞춤형 서비스를 제공합니다. 우선, OCR 알고리즘 도입에서 언급했듯이 단순히 사진 촬영만으로 판결문을 해석해주기 때문에 전국민이 스마트폰을 사용하는 오늘날에 적합합니다. 또한, 해외 리걸 테크 사례와 비교하면, 해외 리걸 테크의 경우, 주로 법조계에 종사하는 법조인들을 위한 서비스들이 대부분입니다. 이와 달리 본 서비스는 법조인이 아닌, 법에 대해 잘 모르는 일반인들을 위한 서비스이기에 법조인과 일반인 간의 법률 관련 정보 비대칭을 해결하고자 합니다. 더불어, 유사 판례 서비스의 경우 키워드 목록을 보여줌으로써, 사용자가 직접 원하는 키워드 기반으로 유사 판례를 추천받을 수 있도록 하여 사용자 맞춤형 서비스를 제공합니다. 이와 같은 사용자 맞춤형 서비스는 기존에 존재하는 리걸 테크 서비스와는 다른 차별성을 보여줍니다.

4. 유저 플로우(User flow)



3. 아이디어를 실현하기 위해 필요한 인공지능 학습용 데이터

[판결문 한글 글자 인식 OCR]

AI Hub '한국어 글자체 이미지' 중 인쇄체 데이터 사용

[생성 요약 서비스]

AI Hub '문서 요약 텍스트' 중 법률 데이터 사용

4. 인공지능 학습용 데이터 학습방법

[판결문 한글 글자 인식 OCR]

1. 학습 목적

판결문은 모두 인쇄체로 되어있습니다. 인쇄체 인식에 적합한 알고리즘을 설계하여 '판결문 인식'에 최적화된 모델을 구축하는 것이 목표입니다. 시중에 OCR 알고리즘은 많지만, 판결문 인식에 특화된 한글 OCR 모델은 공개되어 있지 않습니다. 또한 한글 OCR 모델 개발의 고질적인 한계점이었던 Dataset 부족 문제를 AI Hub 글자체 이미지 Dataset 을 활용함으로써 과대적합과 같은 문제를 최소화한 모델을 만들 수 있을 것이라고 예상합니다.

2. 단계별 과정

AI 허브에서 제공하는 글자체 이미지는 판결문 인식에 적합한 Dataset 이 아니기 때문에 재구축하는 과정이 필요합니다. 재구축된 최종 Dataset 을 가지고, 카메라 촬영 시 왜곡으로 인한 인식률 감소를 최소화하기 위하여 데이터 증축과정을 거쳐 전처리 과정을 진행합니다. 그리고 OCR 에 적합한 알고리즘을 선별하여 최종적인 모델을 만들어냅니다.

[그림 2] 학습 방법 구조

▶ 1 단계. 데이터 수집

AI 허브에서 제공하는 글자체 이미지 Dataset 의 구성과 정보를 알아야 재구축을 원활히 할 수 있습니다. 글자체 이미지는 인쇄체, 손글씨, 실사 이미지 3 가지의 섹션으로 되어있습니다. 우리가 사용하는 것은 인쇄체입니다. 음절 55 만자, 단어 75 만자와 문장 150 만자로 구성되어 있습니다. 아래는 50 가지의 폰트 종류입니다.

번호	폰트명	번호	폰트명	번호	폰트명
1	바탕	18	대한민국정부 상징	35	함초롬돋움
2	돋움	19	호국	36	함초롬바탕
3	굴림	20	서울남산	37	농협희망
4	맑은고딕	21	서울한강	38	하나
5	궁서	22	제주 한라산	39	조선명조
6	휴먼명조	23	제주 명조	40	한겨레
7	휴먼고딕	24	제주 고딕	41	대한
8	중고딕	25	부산	42	가는안상수체
9	나눔고딕	26	고양	43	수화명조
10	나눔바른고딕	27	전북	44	태룡고딕
11	나눔고딕 코딩	28	푸른전남	45	EBS 훈민정음새론
12	나눔명조	29	아리따부리	46	EBS 훈민정음
13	나눔손글씨펜	30	SKT 뫼비우스	47	KBIZ 한마음고딕
14	나눔손글씨붓	31	빙그레따옴	48	KBIZ 한마음명조
15	노토산스	32	티몬소리	49	만화진흥원
16	본고딕	33	미생	50	이롭게바탕
17	노토세리프	34	스포카한산스		

우리는 판결문 인식에 최적화 된 모델을 생성하기 위하여 인쇄체 중 판결문에 쓰이는 폰트만을 사용하려 합니다. 법원 판결문에서 사용되는 폰트는 '판결 서체'입니다. 하지만 법원에서 독자적으로 개발하고 법관이 사용하는 프로그램에만 제공되는 특징을 가지고 있습니다. 따라서 판결 서체를 학습 시키는 것은 불가능합니다. 하지만 아래의 그림을 보면 굉장히 유사한 폰트가 존재합니다.

[그림 3] 판결서체와 일반 폰트의 비교 (출처:제타위키)

따라서 판결서체와 유사한 '바탕체'와 '휴먼명조체'만을 Dataset 으로 지정합니다. 추가적으로 판결문의 폰트 구성은 다음과 같습니다. 본문은 판결서체 12 포인트, 제목은 휴먼고딕 15 포인트, 목차는 휴먼고딕 13 포인트입니다. 하지만 실제 판결문의 구성과 함께 보게 된다면, 우리는 본문 내용만을 필요로 합니다. 그렇기 때문에, 제목과 목차의 폰트는 배제하고 본문의 폰트인 '판결서체'를 위한 모델을 생성하기로 합니다.

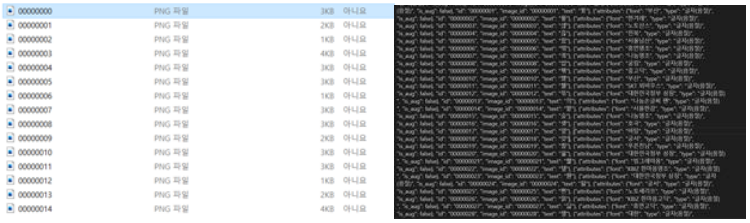
제목, 목차

본문

[그림 4] 판결문 구성 예시 (판결문 출처:서울중앙지방법원)

▶ 2 단계. Dataset 재구축

기존의 글자체 이미지 Dataset 중 인쇄체 이미지 파일은 폰트 순서 상관없이 저장되어있습니다. 따라서 다음과 같은 과정이 필요합니다.



데이터 정보 JSON 파일을 통해 폰트가 바탕,휴먼명조인 이미지 파일의 정보를 가져옵니다. 우선, 데이터 정보 JSON 파일의 크기가 매우 크기 때문에 '시범용'으로 test.json 을 만들어서 과정을 중심으로 설명해보려 합니다. test.json 은 annotations 의 정보에서 5 개만 임의로 저장한 파일입니다. 첫번째로, JSON 파일을 MySQL Workbench 프로그램을 통해 DB 에 저장합니다.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

attributes	text	id	image_id
{"font": "만화진흥원", "type": "글자(음절)", "is_aug": false}	궰	00000000	00000000
{"font": "휴먼명조", "type": "글자(음절)", "is_aug": false}	퀔	00000007	00000007
{"font": "바탕", "type": "글자(음절)", "is_aug": false}	랒	00000018	00000018
{"font": "궁서", "type": "글자(음절)", "is_aug": false}	찑	00000019	00000019
{"font": "푸른전남", "type": "글자(음절)", "is_aug": false}	쉛	00000020	00000020

JSON 파일을 굳이 데이터베이스에 저장시키는 이유는, 폰트의 정보를 중심으로 튜플을 뽑아내기 위해서 입니다. 그렇게 된다면 어떤 이미지 파일이 어떤 폰트로 되어있는지 명시되어 있기 때문에 우리가 원하는 Dataset 을 선별해내는것에 큰 도움이 됩니다.

Query 1 | test

Limit to 1000 rows

```
1 • SELECT * FROM stdt080.test WHERE (attributes -> "$.font" = '바탕') or (attributes -> "$.font" = '휴먼명조');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

attributes	text	id	image_id
{"font": "휴먼명조", "type": "글자(음절)", "is_aug": false}	퀔	00000007	00000007
{"font": "바탕", "type": "글자(음절)", "is_aug": false}	랒	00000018	00000018

SELECT 문을 사용하여 WHERE 절에 attributes 속 font 중 '바탕'이거나 '휴먼명조'가 있는 튜플만을 뽑아냅니다. 그 결과 시범용으로 삽입했던 5 개의 튜플 중 '바탕'과 '휴먼명조'가 있는 2 개의 튜플만이 SELECT 되고, 가공된 table 을 기반으로 Dataset 을 재구축하면 됩니다.

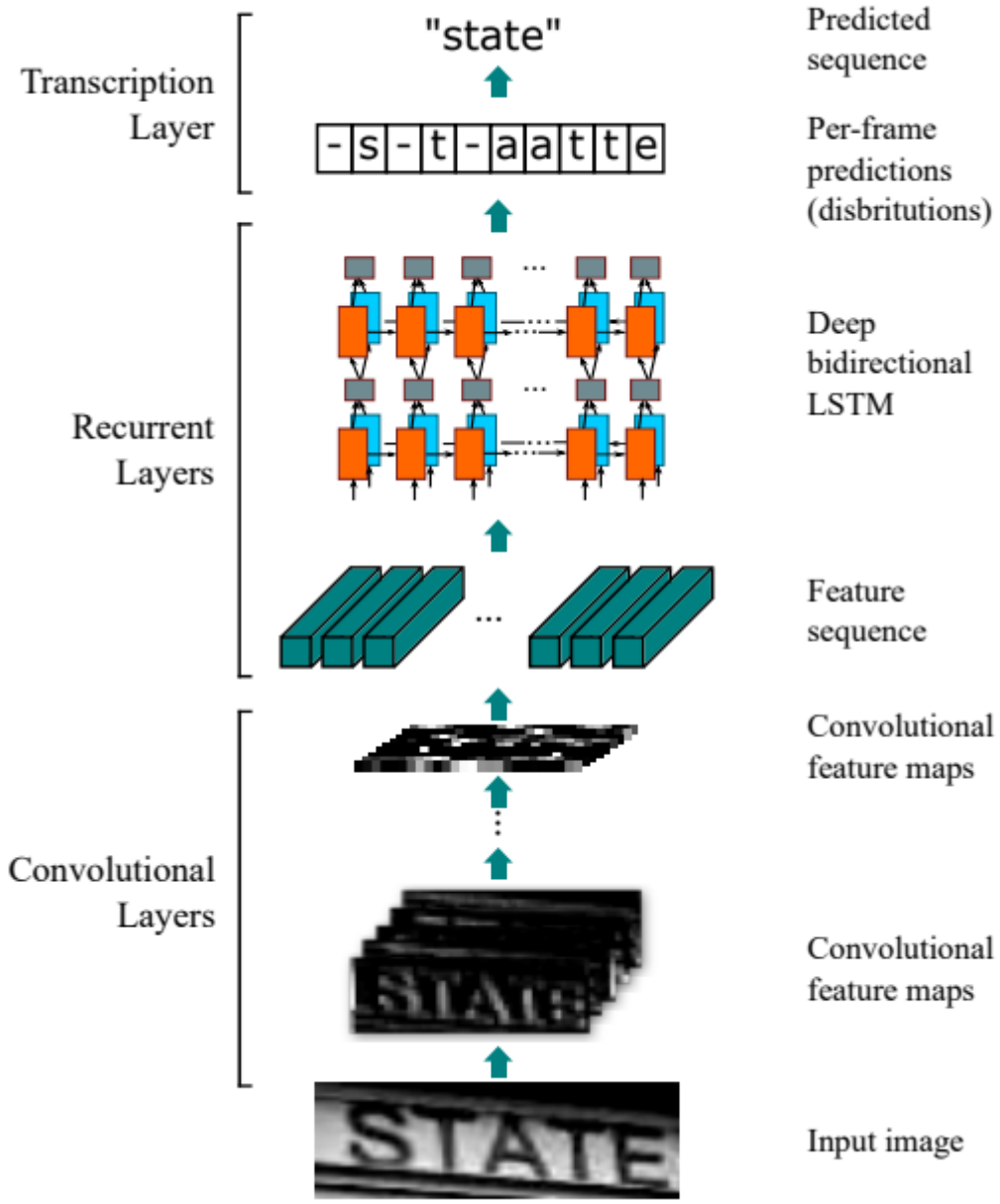
▶ 3 단계. 데이터 증강 (Data Augmentation)

판결문을 등록하는 방법에는 2 가지가 있습니다. 판결문을 카메라로 촬영하거나, 문서를 업로드하는 형식입니다. 문서를 업로드하는 방법은 OCR 에 큰 관여를 주지 않지만, 카메라를 통해 촬영할 경우 정교하게 이미지를 촬영하기 힘들기 때문에 데이터 증강(Data Augmentation)이 필요합니다. 카메라를 이용할 때 사용자 마다 이미지의 기울기, 조도의 변화, 흐릿함 등 다양한 제한 요소가 추가됩니다. 우리는 이러한 제한적 요소를 배제하기 위하여 다음과 같은 시나리오로 데이터 증강을 하려 합니다.

판결문 본문의 특징을 살펴봅니다. 글자색은 검은색, 배경도 흰색이기에 고려하지 않습니다. 하지만 카메라로 판결문을 촬영 할 때 손으로 들고 찍을 경우 글자의 기울기를 고려해야합니다. 또한 책상에 두고 촬영을 진행한다고 해도, 조명으로 인해 그림자가 생길 수 있습니다. 조도에 따른 왜곡 현상도 고려합니다. 카메라의 성능에 따라 선명도와 노이즈에서 차이가 생길 수 있기 때문에 추가합니다. 위와 같은 시나리오를 토대로 증강 변수와 증강 데이터를 생성하는 작업을 진행합니다.

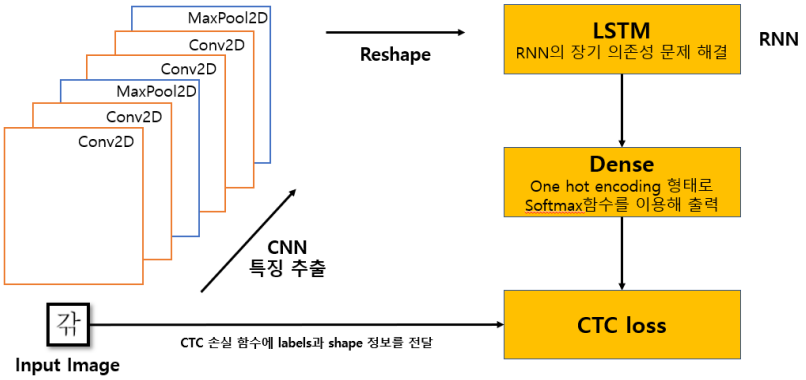
▶ 4 단계. 모델링, 분석 알고리즘

OCR 자체가 인식과 검출의 두가지를 필요로 하기 때문에 각각의 알고리즘을 사용해서 최적의 모델을 만들어내야 합니다. CNN 을 통해서 문서 내의 텍스트 특징을 추출한 다음에 RNN 을 통해 추출된 특징을 문자로 반환하는 작업이 필요합니다. 최종적인 알고리즘은 CNN 과 RNN 을 합친 'CRNN'을 사용하기로 합니다. CRNN 은 CNN 을 연산을 한 뒤에 각각의 채널의 나누어서 RNN 에 입력하는 구조입니다.



[그림 5] CRNN 구조 (출처: 하단 명시)

CRNN 의 구조는 Convolution Layer, Recurrent Layers, Transcription Layer 3 가지의 층으로 구성되어있습니다. Convolutional Layer 를 통해 입력 이미지로부터 Feature Sequence 를 추출합니다. Recurrent Layers 를 통해 추출된 Feature Sequence 를 특정 크기로 분할해 RNN 의 Input 으로 넣습니다. 최종적으로 Transcription Layer 에서 CTC(Connectionist Temporal Classification)를 사용해 Feature 별 예측을 레이블로 변환하게 됩니다. CTC 는 입력에 레이블 할당 위치를 정하기 어려울 경우 연속적인 시퀀스를 다루는 문제에 자주 사용되는 손실함수입니다. 텍스트와 같은 시계열적인 시퀀스를 사용하기 때문에 손실함수로 CTC 를 지정하는 것은 성능 향상에 도움을 줄 수 있습니다.



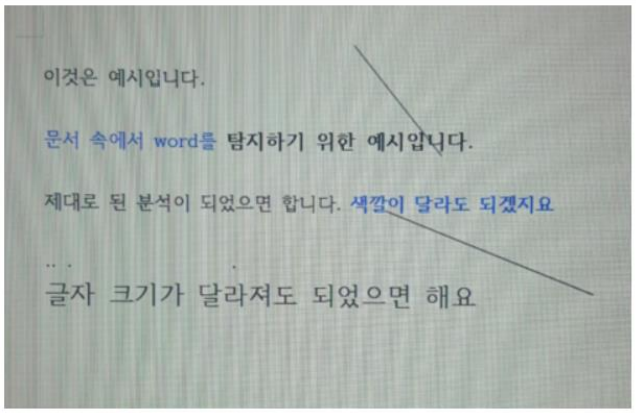
[그림 6] 글자체 이미지 CRNN 구조

최종적으로 사용하는 것은 CTC 손실 함수를 이용한 CNN 과 RNN 입니다. 이미지에서 Feature Sequence 를 추출하는 CNN 과 시계열 순차적 출력을 예측하는 RNN 을 사용하는데, 각 시간 단계에 대한 출력을 예측할 때 사용되는 CTC 손실함수를 가지고 판결문 인식 OCR 에 활용할 것입니다.

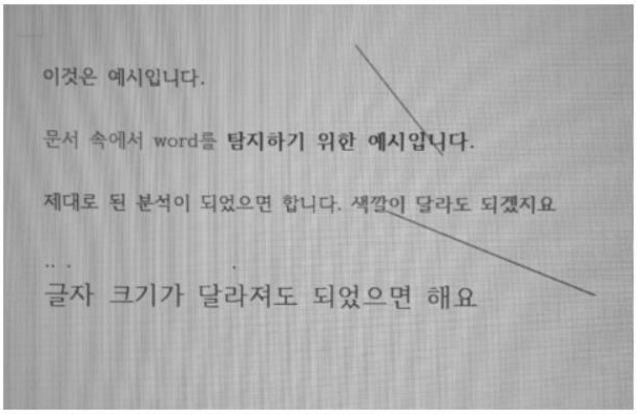
▶ 5 단계. 적용하기

판결문을 촬영 후 인식하기 전, 촬영 이미지를 전처리하는 예시입니다.

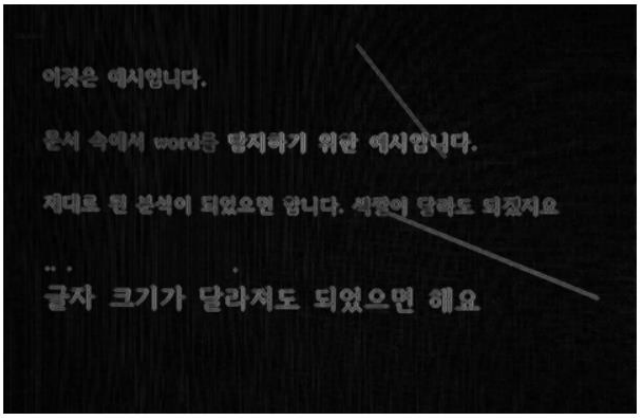
① cv2.pryDown()를 활용해 촬영 이미지의 크기를 줄입니다.



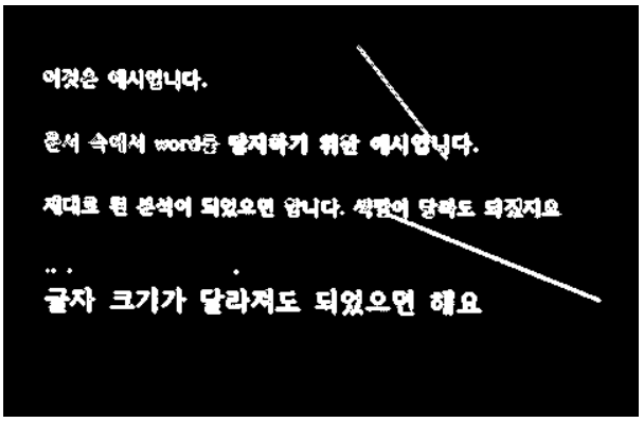
② 이미지 처리를 위해서는 grayscale 이 편리하기 때문에 cv2.cvtColor()의 cv2.COLOR_BGR2GRAY 를 활용하여 grayscale 이미지를 만듭니다.



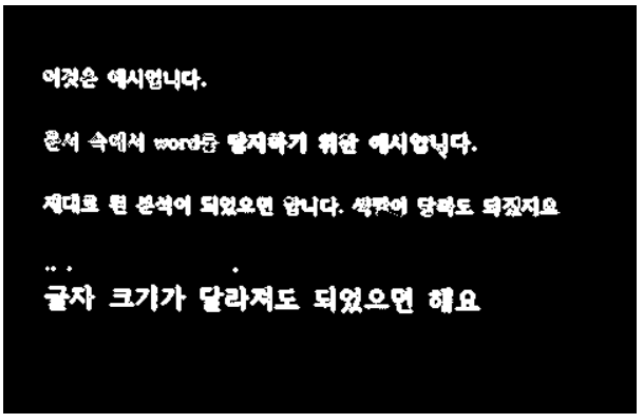
③ 이미지 팽창과 침식인 dilation, erosion 의 차이를 활용하는 cv2.MORPH_GRADIENT 를 활용하여 이미지 상태를 변화시킵니다.



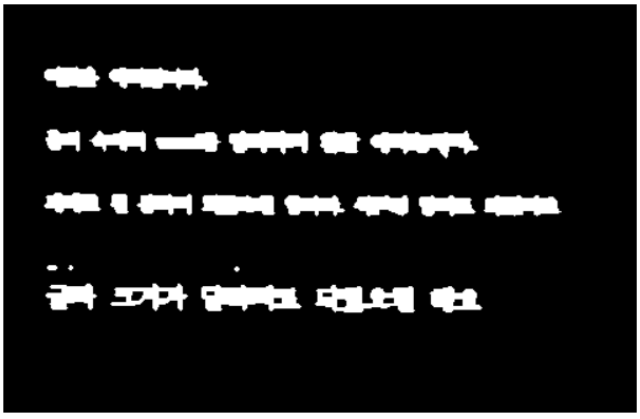
④ threshold 지정을 통해서 무엇인가가 적혀 있는 부분과 그렇지 않은 부분의 차이를 강조합니다.



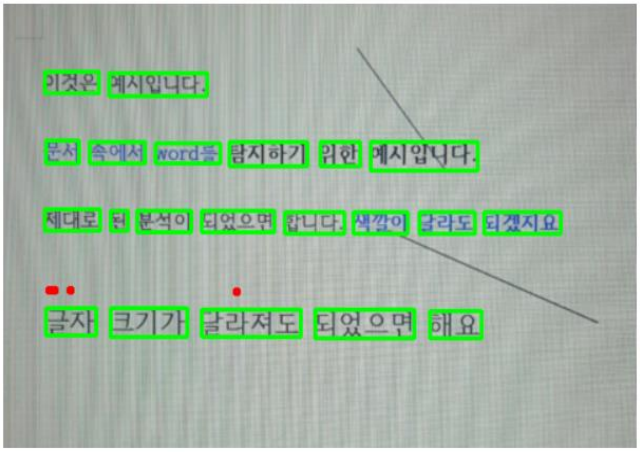
⑤ 가로선, 세로선, 혹은 노이즈 등을 제거하기 위해서 cv2.HoughLinesP()를 활용하게 되고 발견된 가로선, 세로선, 노이즈 등은 4)에서 만든 이미지에서 검은색으로 표시함으로써 제거되도록 합니다.



⑥ 띄어쓰기를 기반으로 이미지 속 텍스트를 명확히 표현하기 위해서 cv2.MORPH_CLOSE 를 이용하여 3)과 마찬가지로 dilation, erosion 을 활용합니다.



⑦ 이미지 contour 를 찾는 작업을 실행할 때, 글자 윤곽선이 아닌 띄어쓰기 기준 bounding box 를 만들기 위해 cv2.boundingRect()를 사용하고 bounding box 의 경우 원본 이미지에 cv2.rectangle()를 사용해서 그려 보여줍니다.



사용자가 촬영한 판결문을 위와 같은 전처리 과정을 진행한 후에, 최종적인 모델을 적용시켜줍니다. 이렇게 함으로써 인식을 할 때 노이즈를 감소시켜 인식률을 향상 시킬 수 있습니다.

[생성 요약 서비스]

AI Hub 에 있는 '문서 요약 텍스트' 중 '법률' 데이터를 학습 데이터로 사용합니다. 데이터의 형태는 법원 판결문 뉴스 텍스트 및 법원 주요 판결문 텍스트이며 3 만개씩의 원문과 추출 요약, 생성 요약을 포함하고 있습니다.

데이터 종류	데이터 형태	데이터 수량	제공 방식
신문기사	뉴스 텍스트	30만 세트	JSON 포맷 파일
기고문	오피니언 텍스트	6만 세트	
잡지	웹진 기사 텍스트	1만 세트	
법률	법원 판결문 텍스트	3만 세트	
합계		40만 세트	

학습 모델은 seq2seq (Sequence-to-Sequence)를 이용합니다. Seq2seq 모델은 RNN 모델에서 시작되었으며, RNN 을 사용할 땐 출력이 바로 이전 입력까지만 고려되기 때문에 정확도가 떨어진 다는 문제점을 가지고 있었습니다. 이를 해결하기 위해 전체 입력 문장을 고려한 모델이 seq2seq 입니다. 입력된 시퀀스로부터 다른 도메인의 시퀀스를 출력하는 모델로 인코더(encoder)와 디코더(decoder)로 구성되어 있습니다. 인코더는 입력 문장의 모든 단어들을 순차적으로 입력받은 뒤 컨텍스트 벡터(context vector)로 압축하고 이 벡터를 이용하여 디코더에서 결과값을 도출해 냅니다. 요약이나 음성 인식(STT) 서비스에 많이 이용되고 있습니다. 구체적인 진행 방향은 다음과 같습니다.

▶ 1 단계. 데이터 전처리

AI Hub '문서 요약 텍스트'의 법률 데이터에서 3 만개의 법원 판결문 텍스트와 3 만개의 생성요약을 사용합니다. 전체 문장을 그대로 사용할 경우 핵심 내용을 뽑고자 하는 취지와는 달리 중요하진 않지만 일반적으로 문장에서 많이 쓰이는 단어가 핵심 단어로 선정될 확률이 높아 전처리 과정을 거칩니다. 파이썬의 한글의 정보처리를 위해 KoNLPy 패키지를 이용하여 형태소를 분리합니다. 그 후 조사, 접속사, 문장부호 등 문장의 내용 분석에 도움이 되지 않는 단어들을 제거하여 불용어를 제외합니다. 조사의 경우 문장의 자연스러운 연결을 도와주지만, 이빨을 의미하는 '이'와 주격 명사 뒤에서 사용되는 조사 '이'와의 차이를 인식을 해야하는 과정이 필요합니다. 많은 신문 기사 제목이 조사가 없이 명사의 나열에 의해 구성되어도 의미상 해석이 가능한 것에서 착안하여 조사는 제거하였습니다.

[형태소 분리]

Before	"판결문은 내용이 복잡하고 어렵다.
After	['판결문', '은', '내용', '이', '복잡', '하고', '어렵다', '.']

[불용어 처리]

Before	['판결문', '은', '내용', '이', '복잡', '하고', '어렵다', '.']
After	['판결문', '내용', '복잡', '어렵다']

▶ 2 단계. 단어와 숫자 매핑

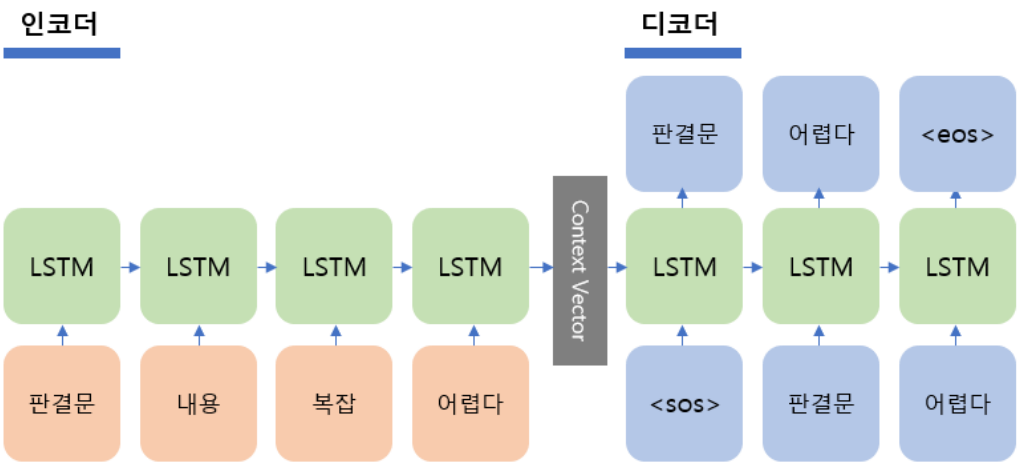
Seq2seq 모델에서 입력(input)을 숫자로 하기 위해 각각의 단어를 숫자와 매핑 시켜줍니다. 이때 매핑 정보는 학습 말뭉치에 저장됩니다. 또한 시퀀스의 길이를 생성요약의 최대 단어수로 제한합니다.

[학습 말뭉치]

{'판결문' : 6014, '내용' : 17264, '복잡' : 201, '어렵다' : 1291}

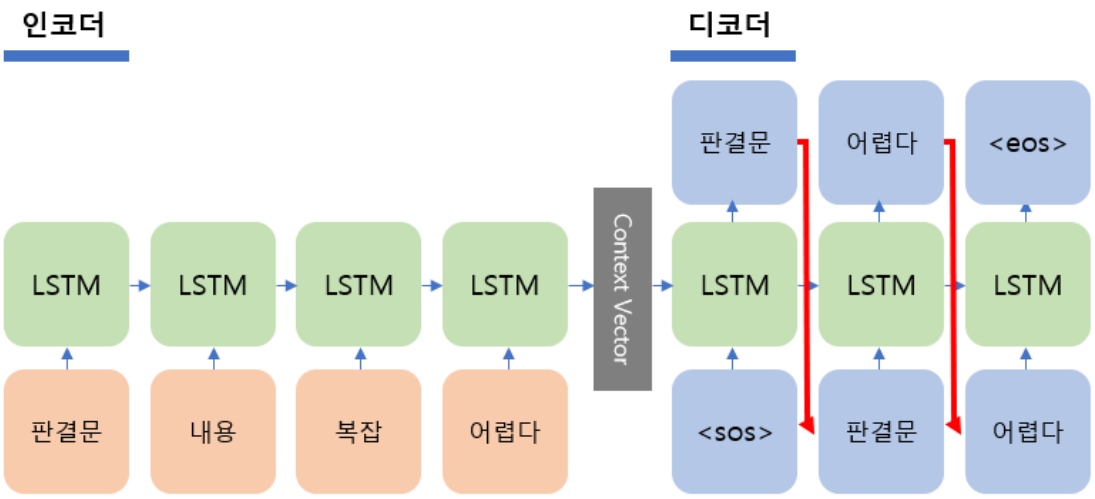
▶ 3 단계. Seq2Seq 모델 학습

먼저 인코더에서 입력 문장의 전처리가 끝난 단어들을 순차적으로 입력받아 컨텍스트 벡터를 구성합니다. 아래의 그림에서 '판결문', '내용', '복잡', '어렵다' 와 같은 단어들이 전처리가 끝난 입력 문장이고 이를 종합하여 컨텍스트 벡터가 디코더에 전달됩니다. 다음으로 디코더는 이 벡터를 이용하여 요약문을 출력합니다. 디코더에서도 입력과 출력이 나누어지는데, 디코더 입력에는 생성 요약 정답을 넣습니다. 이는 인코더에게 문제와 그에 대한 정답을 알려주며 학습을 시키는 교사 강요(teacher forcing) 과정입니다. 그림에서 [<sos>, '판결문', '어렵다']처럼 AI Hub 에서 제공된 실제 생성 요약 정답을 넣어주고, ['판결문', '어렵다', '<eos>]는 최종적으로 도출된 요약문입니다. 이 때 실제 생성 요약 정답과 도출된 요약문 간의 손실(loss)을 최소화 하는 방향으로 학습이 진행됩니다. <sos>는 'Start Of String'으로 디코더에 실제 생성 요약 정답이 입력된 다는 것을 의미하는 토큰이고, <eos>는 'End Of String'으로 디코더에 도출된 요약문이 끝났다는 것을 의미하는 토큰입니다. 추가적으로 RNN-LSTM 을 이용하여 긴 기간의 의존성(long-term dependencies)을 다룰 수 있게 합니다.



▶ 4 단계. Seq2Seq 모델을 이용한 예측

사용자가 판결문을 사진 찍으면 OCR 기능을 이용하여 판결문을 텍스트 형태로 변환하고 이를 앞서 진행한 것처럼 전처리 과정을 거칩니다. 이 단어들을 통해 만들어진 컨텍스트 벡터를 학습되어진 seq2seq 모델을 이용하여 디코더의 출력을 요구합니다. 하지만 이 때 학 습 과정에서처럼 디코더에 입력되는 생성 요약의 정답이 없습니다. 따라서 컨텍스트 벡터와 <sos>가 입력되면, 학습 결과 등장할 확률이 높은 단어를 예측합니다. 아래 그림에서는 첫 번째 단어로 '판결문'을 예측 하였습니다. 두 번째 시점부턴 디코더에 출력된 단어를 디코더 입력 값으로 하여 연쇄적으로 디코더 출력을 실시합니다. 즉 첫 번째 시점에서 디코더 출력으로 도출된 '판결문'이라는 단어가 두 번째 시점에서 디코더 입력으로 들어가게 됨을 의미합니다. 이는 디코더 출력에 <eos>가 나올 때까지 계속되며 결과적으로 디코더 출력값으로 추출된 단어들의 집합이 인코더에 입력한 판결문 원문의 생성 요약이 됩니다.



5. 데이터 융합을 위해 필요한 공공 데이터와 인공지능 학습용 데이터

[법률 용어 서비스]

<AI-Hub 학습용 데이터 목록>
'지식베이스-법률'의 법령용어 JSON 형식 사용

<공공 데이터 목록>
공공데이터포털-'국가법령정보 공동활용'의 법령용어 목록 조회 Open API
URL : https://www.data.go.kr/data/15057595/openapi.do
공공데이터포털-'국가법령정보 공동활용'의 법령용어 본문 조회 Open API
URL : https://www.data.go.kr/data/15057261/openapi.do

[유사 판례 추천 서비스]

<AI-Hub 학습용 데이터 목록>
'지식베이스-법률'의 관련 판례 JSON 형식 사용

<공공 데이터 목록>
공공데이터포털-'국가법령정보 공동활용' 의 '판례 본문 조회' Open API
URL : https://www.data.go.kr/data/15057123/openapi.do

6. 융합 데이터 구축 과정

[법률 용어 서비스]

① AI-Hub 의 법률용어 JSON 데이터와 공공데이터포털 국가법령정보 공동활용(이하 공공데이터)의 법령용어 목록 및 법령용어 본문 조회의 XML 파일을 융합하여 데이터를 구축합니다.

② 구체적으로는 AI-Hub 의 법률용어와 공공데이터 법령용어목록의 법령용어명을 파싱(parsing)하여 우선 DB 에 저장한 후, 중복을 제거하여 용어 DB 에 다시 저장합니다.

③ 법령용어 정의는 크롤링 한 AI-Hub 의 uri 정보 및 공공데이터 법령용어 본문 조회의 법령용어정의를 파싱(parsing)하여 우선 DB 에 저장한 후, 중복을 제거하여 용어 DB 에 다시 저장합니다.

④ 이를 통해 융합된 테이블은 column 으로 법률용어명, 법률용어 정의를 가지며, 이는 융합의 과정을 거쳐 법령의 범위를 수평적으로 넓게 확보하여 정보 탐색의 가능성을 높였다는 점에 의의가 있습니다.

[유사 판례 추천 서비스]

① AI-Hub 에서 제공하는 지식베이스 법률 분야와 공공데이터 포털에서 제공하는 판례 본문 조회 데이터를 융합하여 커스텀 판례 데이터베이스를 구성하여 사용합니다.

② 이때, AI-Hub 에서 제공하는 JSON 데이터와 공공데이터 포털에서 제공하는 XML 파일에서 각각의 key 값을 매칭하여 같은 항목임을 알 수 있도록 합니다. 매칭 결과는 다음과 같습니다.

AI-Hub JSON 데이터	공공데이터 포털 XML 데이터
caseNumber	사건번호
courtName	법원명
sentenceDate	선고일자
caseName	사건종류명(사건명)
judgementAbstract	판시사항
refArticle	참조조문
precedentText	판례내용
judgementNote	판결요지

③ 이와 같은 8 가지 항목에 대하여 매칭을 시키고 융합 데이터를 구성합니다.

우선 Json 파일과 XMl 파일 모두 DataFrame 으로 구성합니다.
이때, column 은 한글로써 사건번호, 법원명, 선고일자, 사건종류명(사건명), 판시사항, 참조조문, 판례내용, 판결요지 로 구성합니다.
위의 8 가지 column 으로 AI-Hub 학습용 데이터와 공공데이터를 합친 후, 사건번호를 통해서 중복인 경우 하나를 제거하는 방식을 통해 최종 융합 데이터를 완성합니다.

④ 이를 통해 융합된 테이블은 column으로 판례를 가지며, AI-Hub에서 제공하는 7가지 분야(교통사고, 층간소음, 창업 인허가, 퇴직금, 이혼, 한부모가족, 학교폭력)를 포함하여 사용자가 원하는 분야에 따라 더 넓고 다양한 판례를 제공할 수 있음으로써 제공 판례의 범위를 수평적으로 넓게 확보하게 됩니다. 이를 통해 사용자에게 더욱 다양하고 정확한 유사 판례 제공 기능을 제공할 수 있음에 의의를 가집니다.

7. 융합된 데이터 학습 방법

1. 개설

광학 문자 인식(OCR)과 판결문 요약과 관련된 학습 방법은 전술하였습니다. 법률용어 서비스의 경우에는 별도의 데이터 학습을 요하지 아니하여, 본 목차에서는 유사 판례 추천 서비스의 데이터 학습 방법을 위주로 작성하였습니다.

2. 유사 판례 추천 서비스의 융합된 데이터 학습 방법

기본적으로 데이터는 위에서 서술한 가공된 융합 데이터를 사용합니다.

▶ 1 단계. 판례 데이터 1 차 선정

① 사용자가 사건번호를 선택, 유사 판례 보기 버튼을 클릭하면 알고리즘이 진행됩니다.
② 판례의 경우 사건번호에 해당하는 판례에 대해서만 알고리즘을 적용합니다.

▶ 2 단계. 판례 데이터 1 차 가공 – 기호, 불용어 제거

① 사건번호에 해당하는 판례들과 사용자가 업로드하여 요약된 판결문에 대해서 konlpy 를 사용하여 조사, 접속사, 형용사를 제거합니다. konlpy 를 사용하게 되면 형태소별로 구분되어서 나타나게 되며 조사는 'Josa', 접속사는 'Conjunction', 형용사는 'Adjective'로 나타납니다.
② 특히 konlpy.tag.Okt()를 사용하여 제거하게 됩니다.
③ 조사, 접속사, 형용사를 제거하는 이유는 유사 판례를 구하기 위해서는 키워드 기반으로 이루어져야 하는데 판결문의 경우 조사, 접속사, 형용사의 경우 키워드가 될 수 없다고 판단하여 미리 제거함으로써 정확도를 높이기 위함입니다.
④ 또한, 텍스트 속에 있는 ~ ! @ # $ % ^ & * ₩ " ' + = ` | ( ) [ ] { } : ; - _ - # & & @ § ※ 등의 기호들도 제거합니다.

▶ 3 단계. 사용자의 판결문 속 키워드 생성

① 키워드 생성에 있어서 TF-IDF 를 사용합니다. TF 란, 특정 문서 속 특정 단어의 등장 횟수를, IDF 란 특정 단어가 등장한 문서 수를 나타내는 DF 에 반비례하는 수입니다. 이 두 값을 곱한 값이 TF-IDF 로써 이를 활용하여 키워드를 생성합니다.
② 기호, 불용어가 제거된 사용자의 판결문, 융합 판례 데이터들의 TF-IDF 값을 구하게 되는데 이때 sklearn 에서 제공하는 TfidfVectorizer 를 사용하게 됩니다.
③ TfidVectorizer 를 통해서 구하게 된 텍스트 속 각 단어에 대한 TF-IDF 값을 이용합니다.
④ 사용자의 판결문 속 단어의 TF-IDF 값을 이용하는데 TF-IDF 값이 클수록 중요도가 높다는 뜻이므로 그 값이 클수록 키워드라고 간주하게 됩니다.
⑤ 이렇게 구해진 TF-IDF 값에 대해서 내림차순으로 정렬하고 상위 10 개에 대해서 순서대로 사용자에게 보여주고 키워드로써 선택할 수 있게 합니다.

▶ 4 단계. 판례 데이터 2 차 가공(키워드로만 이루어진 새로운 텍스트 생성)

① TF-IDF 를 구할 때 키워드로 보여준 건 사용자의 판결문이지만 이 뿐 아니라 융합 데이터 판결문에 대한 TF-IDF 값도 동시에 구해지게 됩니다.

② TF-IDF 값이 0.3 미만인 것에 대해 제거하고 남은 단어들로 새로운 텍스트를 구성합니다.

▶ 5 단계. 사용자 선택 키워드 기반 유사 판례 추천

① 여기서는 코사인 유사도를 사용합니다. TF-IDF 값들은 모두 벡터로 이루어진 값이기 때문에 사용 가능하며 -1~1 의 값이 나타나게 되는데 1 에 가까울수록 유사함을 뜻합니다.

② 코사인 유사도는 sklearn 에서 제공하는 linear_kernel 를 사용합니다.

③ 사용자가 키워드를 선택하게 되면(3 개 이상 선택) 선택한 키워드로만 구성된 텍스트를 만들고 그렇게 생성된 텍스트와 2 차 가공된 판례 데이터와의 TF-IDF 값을 구하고 이를 linear_kernel 에 적용하여 코사인 유사도를 구합니다.

④ 구해진 코사인 유사도 값에 대해 내림차순으로 정렬하고 유사도가 가장 높은 순서대로 유사도 값과 함께 사용자에게 보여주게 됩니다.

▶ 6 단계. 알고리즘 예시

① 예시 데이터-기존 가지고 있는 데이터

```
docs = [
  '그런데 먹고 싶은 사과',
  '먹고 싶은 바나나',
  '길고 노란 바나나 바나나',
  '저는 과일이 좋아요',
  '먹고 싶은 사과',
  '사과는 영어로 apple 입니다'
]
```

② 예시 데이터-사용자가 가지고 있는 데이터

```
want="내가 먹고 싶은 것은 바나나 입니다. 바나나 좋아요"
```

③ 불용어 제거

```
import konlpy.tag
docs = [
  '그런데 먹고 싶은 사과',
  '먹고 싶은 바나나',
  '길고 노란 바나나 바나나',
  '저는 과일이 좋아요',
  '먹고 싶은 사과',
  '사과는 영어로 apple 입니다'
]
after_docs=[]
for now in docs:
  Okt=konlpy.tag.Okt()
  Okt_morphs=Okt.pos(now)
  temp=[]
  for word, pos in Okt_morphs:
    if pos=="Josa" or pos=="Conjunction" or pos=="Adjective":
      continue
    else:
      temp.append(word)
  newsent=" ".join(temp)
  after_docs.append(newsent)
print(after_docs)
```

['먹고 싶은 사과', '먹고 싶은 바나나', '바나나 바나나', '저 과일', '먹고 싶은 사과', '사과 영어 apple']

④ TF-IDF 사용 키워드 생성

```
now_docs=after_docs
now_docs.append(after_want)
tfidfv_recommend=TfidfVectorizer().fit(after_docs)
tfidf_recommend=tfidfv_recommend.transform(after_docs).toarray()
print(tfidf_recommend)
print(tfidfv_recommend.vocabulary_)
cosine_recommend=linear_kernel(tfidf_recommend, tfidf_recommend)
print(cosine_recommend)
```

```
[[0.         0.         0.54827342 0.         0.63150021 0.54827342
  0.        ]
 [0.         0.         0.54827342 0.63150021 0.         0.54827342
  0.        ]
 [0.         0.         0.         1.         0.         0.
  0.        ]
 [0.         1.         0.         0.         0.         0.
  0.        ]
 [0.         0.         0.54827342 0.         0.63150021 0.54827342
  0.        ]
 [0.63202178 0.         0.         0.         0.44843834 0.
  0.63202178]
 [0.         0.         0.36995057 0.85221661 0.         0.36995057
  0.        ]]
{'먹고': 2, '싶은': 5, '사과': 4, '바나나': 3, '과일': 1, '영어': 6, 'apple': 0}
[[1.         0.60120749 0.         0.         1.         0.28318891
  0.40566812]
 [0.60120749 1.         0.63150021 0.         0.60120749 0.
  0.94384309]
 [0.         0.63150021 1.         0.         0.         0.
  0.85221661]
 [0.         0.         0.         1.         0.         0.
  0.        ]
 [1.         0.60120749 0.         0.         1.         0.28318891
  0.40566812]
 [0.28318891 0.         0.         0.         0.28318891 1.
  0.        ]
 [0.40566812 0.94384309 0.85221661 0.         0.40566812 0.
  1.        ]]
```

```
wordlist=tfidfv.vocabulary_
wordlist=dict(zip(wordlist.values(),wordlist.keys()))
#total words
each_words=[]
each_words_percent=[]
keyword_text=[]
for i in range(len(tfidf)):
  now=tfidf[i]
  keyword_idx=np.argsort(-tfidf[i])
  words=[]
  percent=[]
  for j in range(len(keyword_idx)):
    if now[keyword_idx[j]]==0:
      break
    percent.append(now[keyword_idx[j]]*100)
    words.append(wordlist[keyword_idx[j]])
  keyword_text.append(" ".join(words))
  each_words.append(words)
  each_words_percent.append(percent)
print(keyword_text)
#keyword_text is the text that only contains keywords
for i in range(len(each_words)):
  print(each_words[i])
  print(each_words_percent[i])
#keywords of total text
#have to work and store before user using keyword system
```

```
['먹고 사과 싶은', '바나나 먹고 싶은', '바나나', '과일', '먹고 사과 싶은', 'apple 영어 사과']
['먹고', '사과', '싶은']
[57.73502691896258, 57.73502691896258, 57.73502691896258]
['바나나', '먹고', '싶은']
[64.2084608164228, 54.20919460564738, 54.20919460564738]
['바나나']
[100.0]
['과일']
[100.0]
['먹고', '사과', '싶은']
[57.73502691896258, 57.73502691896258, 57.73502691896258]
['apple', '영어', '사과']
[63.509072052150486, 63.509072052150486, 43.96811952027871]
```

```
now_keyword=tfidf_recommend[len(tfidf_recommend)-1]
keyword_idx=np.argsort(-now_keyword)
now_keyword_text=[]
for i in range(7):
  if now_keyword[keyword_idx[i]]*100==0:
    break
  print(wordlist[keyword_idx[i]], now_keyword[keyword_idx[i]]*100)
  now_keyword_text.append(wordlist[keyword_idx[i]])
print(now_keyword_text)
now_keyword_text=" ".join(now_keyword_text)
#now_keyword_text is the text that chosen text but only contains keywords
print(now_keyword_text)
#keywords of text that want
```

```
바나나 85.22166141189456
먹고 36.99505660352747
싶은 36.99505660352747
['바나나', '먹고', '싶은']
바나나 먹고 싶은
```

⑤ 코사인 유사도 이용 추천

```
total_keyword_text=keyword_text
total_keyword_text.append(now_keyword_text)

tfidfv_keyword=TfidfVectorizer().fit(total_keyword_text)
tfidf_keyword=tfidfv_keyword.transform(total_keyword_text).toarray()
cosine_keyword=linear_kernel(tfidf_keyword, tfidf_keyword)
keyword_percent=cosine_keyword[len(cosine_keyword)-1]
keyword_recommend=np.argsort(-keyword_percent)
for i in range(len(keyword_recommend)):
  if after_want==now_docs[keyword_recommend[i]]:
    continue
  print(now_docs[keyword_recommend[i]], keyword_percent[keyword_recommend[i]]*100)
```

```
먹고 싶은 바나나 99.99999999999999
바나나 바나나 63.150020738605114
먹고 싶은 사과 60.12074880713742
먹고 싶은 사과 60.12074880713742
저 과일 0.0
사과 영어 apple 0.0
```

8. 참고자료

[그림 5] 출처: Baoguang Shi, Xiang Bai and Cong Yao,"An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition", Huazhong University of Science and Technology, 2015, p2