**CYBERAFFITI 사이버래피티**

**유해도 기반의 필터링 시스템**

빅데이터 청년인재 고려대학교 3조

남궁찬 박재은 장민교

정우태 조민수 채수영

목차

Ⅰ 서론3

1. 개발 배경3
2. 시장 조사 및 필요성3

Ⅱ 본론4

1. 데이터 수집4

1.1) 수집 대상4

1.2) 수집 방법 4

1. 전처리7

2.1) 데이터 셋 구축 및 투표 기반 레이블링7

2.1.1) Data Argumentation

2.2) 토큰화8

2.2.1) 형태소 분석 기반8

2.2.2) Bype Pair Encoding8

2.2.3) nKT14

2.2.4) AUTO-LABELING18

3. 모델19

3.1) 1dCNN19

3.2) Self-Attention Bidirectional-LSTM19

4. 확장프로그램 및 UI25

1.1) CHROME 확장 프로그램 소개 및 작동 방식25

1.2) 확장프로그램 서버 연동 및 UI 구현25

Ⅲ 결론27

1. 개발 결과27
2. 기대 효과 28

부록29

**Ⅰ 서론**

1. **개발 배경**

정보화 시대의 흐름 속에서 현대인들은 여러 매체를 이용하여 다양한 정보를 얻을 수 있게 되었다. 인터넷과 모바일을 이용하면 사용자는 언제든 어디서든 원하는 정보를 쉽게 구할 수 있지만, 이러한 장점에는 그에 버금가는 부작용도 존재한다. 선정적이고 폭력적인 미디어들에 대한 접근 또한 용용 해진 것은 물론이고, 심의 규정에 따라 제재를 하는 경우에도 검색을 통해 우회하여 접속하는 방법을 쉽게 알아낼 수 있기 때문이다.

1. **시장 조사 및 필요성**

인터넷 방송 플랫폼의 경우에는 아주 최소한의 제재만이 적용되고 있다. 인터넷 방송은 인터넷 망을 사용하지 않는데, 이 경우에는 방송 구역이라는 개념이 적용이 되지 않아서 플랫폼의 자체적인 제재에만 기댈 수밖에 없기때문이다. 따라서 현재 대표적인 인터넷 방송 플랫폼으로 손 꼽히는 유투브, 아프리카티비, 트위치 등에서는 선정적이고 폭력적인 멘트 및 영상들이 인터넷 망을 통해 유포되고 있다. 문제가 되는 부분은 이 유해 영상들에 대한 필터링, 즉 플랫폼의 자체적인 제재가 제대로 수행되고 있지 않다는 점이다. 트위치 및 아프리카 티비에서는 채팅방 안에서 필터링을 수행하는 기능이 존재하지만 정확한 비속어 및 욕설을 사용해야만 필터링이 되고 있기에 변형된 욕설들에 대한 탐지 성능이 못한 편이다. 또한 유투브는 신고 기반으로 필터링이 되고 있기에 유해 영상에 대한 신고가 없이는 영상에 대해 제재를 할 수 없는 상황이다.

2018년 방송 통신위원회에서 조사한 결과에 따르면 10대 청소년 10명 중 1명이 인터넷 방송을 시청하며 평균적으로 하루에 2시간씩 시청을 한다. 이때 어린이 및 청소년들이 주로 시청하는 영상 중 27개의 콘텐츠를 선별하여 모니터링을 해보니 총 1034개의 유해한 내용들이 나타났는데, 이를 통해 방송 통신 위원회는 아직 가치관이 확립되지 않은 어린이 및 청소년이 인터넷 방송을 접했을 때 받을 수 있는 부정적인 영향력에 대한 상관관계를 입증해 냈다. 그리고 실제로 이 조사 결과를 확인해보기 위해 플랫폼 3사를 방문한 경우에도 VOD 등을 통해 유해한 컨텐츠들이 소비되고 있는 것을 쉽게 볼 수 있었다.

유투브에서는 ‘엘사게이트’로 인한 문제가 뜨거운 감자이다. 엘사게이트는 어린 아이들이 좋아하는 캐릭터로 시청을 유인하지만 실질적으로는 선정적이고 폭력적인 내용이 담겨있어 아이들로 하여금 부적절한 영상을 시청하게끔 만드는 피싱을 일컫는 단어이다. 주로 썸네일 및 제목에 아이들이 좋아하는 캐릭터를 등장시켜 클릭을 유도하는데, 막상 컨텐츠의 내용은 캐릭터의 스토리가 아닌 성폭행, 불륜, 패티시즘, 신체절단, 음주, 흡연, 납치 그리고 아동학대에 대한 것들로 구성되어 아이들의 동심을 파괴한다. 주로 겨울왕국의 엘사를 도용한 사례가 엘사게이트[[1]](#footnote-1)라 불린다

따라서 사이버 공간상에서 존재하는 유해한 컨텐츠. 일종의 낙서를 필터링하고 지워주고 싶다는 목적 하에서 본 사이버래피티 프로젝트를 진행하게 되었다.

**Ⅱ 본론**

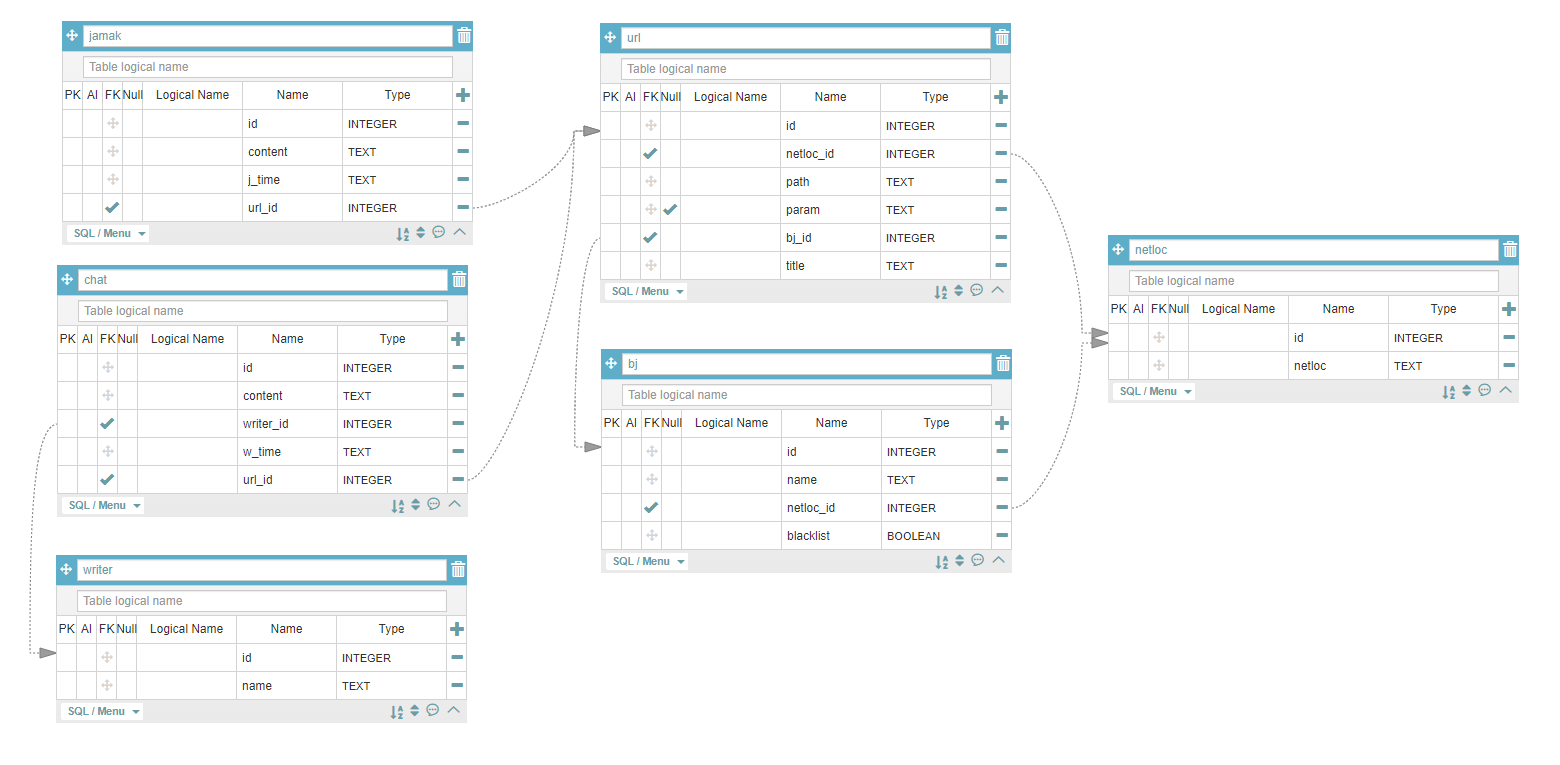
1. **데이터 수집**
   1. **수집 대상 및 수집 방법**

인터넷 방송 플랫폼으로 유명한 트위치, 아프리카티비, 유투브를 기준으로 데이터를 수집했다. 트위치와 아프리카티비는 인터넷 BJ[[2]](#footnote-2)의 채널에서 존재하는 영상들에 대한 채팅데이터를 이용했고, 유투브는 영상에 대해 자동적으로 생성되는 자막데이터를 가져왔다.

* 1. **수집 방법**

비정형 데이터인 채팅 및 자막을 수집하기 위해서 크롤링 및 스크래핑을 진행했다. 각각의

플랫폼에 대해서는 공통으로 Download 함수를 생성하여 필요한 데이터를 수집하였고, 400 및 500번대 에러가 발생한 경우에는 수집이 불가능하다는 것을 알리게끔 처리하여 상태코드가 200인 경우에만 데이터를 긁어 오도록 코드를 작성했다. 그리고 수집된 데이터는 SQLite를 이용하여 DB에 저장해 관리했다.



[그림1] 사용한 DB 테이블

계획했던 필터링 시스템의 시나리오는 다음과 같다.

1. 우선적으로 VOD 영상에서 채팅 및 자막 데이터를 수집하며, 이 데이터를 모델에 넣었을 때 유해한 단어들이 임계치를 넘어가면 해당 방의 BJ를 블랙리스트 명단에 추가했다.
2. 그러면 앞서 설치한 크롬 확장프로그램이 블랙리스트 명단에 있는 비제이의 영상들을 화면에서 볼 수 없게 지우는 역할을 수행하는데 이 과정을 통하여 VOD를 포함한 실시간 영상에 대해서도 차단이 가능하도록 만들었다.
3. 이때 블랙리스트 명단은 일주일을 기준으로 하여 업데이트했으며, 크롬 확장프로그램이 자동적으로 업데이트를 수정하도록 하여 사용자는 설치 이외에 추가적인 작업을 하지 않아도 되게끔 하였다.

* 유투브

유튜브의 자막 데이터는 영상에 접속했을 때 나타나는 개발자 도구를 클릭하여 가져왔고, Network 페이지 중에서 timedtext로 명명된 부분을 참고하여 수집을 진행했다.

* + - * 크롤링을 하는 경우에 내용을 바로 긁어올 수가 없었는데, headers에 upgrade-insecure-requests, user-agent, x-client-data 부분을 추가하여 가져올 수 있었다.
      * BeautifulSoup을 이용하여 html을 받아올 수 있었고 signature 부분에 정규식을 사용하여 필요한 채팅 데이터만을 뽑을 수 있었다.
      * 또한 유투브는 때에 따라서는 접속이 되다가 어떤 경우에는 404 not found로 에러가 뜨곤 했기에 재귀함수를 이용했다. 경험적으로 최대 7번까지 시도했을 때 한 번은 접속이 되는 것을 확인했기에 limit을 7로 설정하여 연결 설정을 7번 이내로 시도하게 코딩했다.
      * 이때 유투브의 자막 데이터는 시간대별로 표현이 되고 있었는데, 확인해보니 생성된 자막의 성능이 좋지 않은 편이었다.
* 트위치

- 트위치 영상에 대해서는 인풋으로 파라미터 값(url 끝 부분으로 예를 들면 /2633849)을 전달해주면 채팅을 파싱해주는 식으로 함수를 작성했다.

- 트위치는 스크립트를 비동기통신으로 보내주기에 채팅을 한 번에 가져오는 것이 힘들어서, 이 부분을 어떻게 처리할지에 대한 고민이 필요했다.

- bj 영상을 틀어 둔 채로 개발자 도구를 실행시키면, 네트워크 탭에 bj의 아이디로 명명된 페이지(예, ambition\_lol)가 생성되는데, 이 곳에서 채팅데이터를 긁어올 수 있다. 이때 자막 데이터들은 일정량이 넘어가면 페이지로 구분이 되어서, 다음 페이지로 넘어가는 next가 나오지 않는 경우를 마지막 부분이라 판단하여 next가 나오지 않을 때까지의 채팅 데이터를 긁어오게 코드를 작성했다.

.

* 아프리카티비
  + - * 아프리카 티비는 영상을 실행시켰을 때 개발자도구에서 Network부분이 ChatLoad로 시작하는 페이지를 선택하고, beautifulSoup으로 html 내용을 받아와 metatag를 수정하는 식으로 데이터를 수집했다.
* 라벨링 된 데이터 셋 구축

수집한 데이터들은 바로 DB에 저장하였으며 궁국적으로는 트위치에서 채팅 bj11명과 그에 대한 채팅 데이터 2백만개, 아프리카 티비에서 bj 3만여명의 리스트를 확보할 수 있었다. 그리하여 데이터 베이스 파일에서 라벨링을 위한 데이터 셋을 모을 수 있었고 csv 파일로 저장했다.

* 힘들었던 부분
  + - * 크롤링 및 스크래핑을 수행하면서 힘들었던 점은 해당 사이트의 개발 모드로 확인하는 순간 나타나는 많은 페이지 중에서 관련 있는 request를 일이 확인해야하는 부분이었다.
      * 또한 인코딩에 관련된 지식이 부족해서 RHTODGOTEKK. open 함수를 이용하여 csv 파일을 작성할 때 encoding을 utf-8로 지정하면 에러가 발생하진 않지만 한글이 깨지는 문제가 발생하고, euc-kr을 사용하면 한글이 저장되지만 이모티콘 등이 깨지는 에러가 발생하여 csv 파일이 생성되지 않았기 때문이다. 따라서 공부를 하면서 코드를 작성해보았다. 아래는 검색을 하면서 익힌 부분들이다.
        + 대부분 한글을 출력하기 위해서는 UTF-8(UTF:Universal Coded Character Set +Transformation Format – 8 bit의 약자) 또는 EUC\_KR 인코딩을 사용한다. ECU\_KR은 완성형으로써 한글을 2바이트 취급하 많이 사용되는 한글의 자음과 모음을 조합하여 2350개의 한글을 표현해낸다. 반면 UTF-8은 조합형으로써 한글을 3바이트 취급하고 자음과 모음을 조합하여 모든 한글을 표현할 수 있다. 또한 전 세계 모든 글자를 인코딩 없이 표현 가능하며 자바의 경우에는 내부적으로 유니코드가 사용되기에 인코딩 변환이 필요 없다는 장점을 가지고 있다.
        + UTF-8의 경우 아스키 영역에서 1바이트를 그대로 사용하기에 호환이 된다. 반대로 말하면 UTF-8 문서라도 ASCII 영역에 해당되는 문자만 작성되기에 BOM이 없다면 ASCII 문서와 다를바가 없다. 그러나 UTF-16은 2바이트에서 시작하기에 호환이 되지 않는다. 그래서 UTF-16에서는 ASCII 문자를 나타낼 경우에 앞에 0x00이 붙는다. 예를 들어서 A라는 글자를 표현할 경우에 아스키 및 UTF-8에서는 0x41이라고 표현하나, UTF-16은 0x0041로 표현한다. 그래서 UTF-8은 주로 사용되는 기본 다국어 평면 문자들은 그대로 16비트로 인코딩 되며 그 외의 문자는 특별히 정해진 방식으로 32비트 인코딩이 된다.

따라서 우리의 경우에는 csv 파일을 저장할 때 utf-16으로 저장하게 하여 문제를 해결했다.

1. **전처리**
   1. **데이터셋 구축 및 투표 기반 레이블링**

우선적으로는 수집한 5만여개의 데이터들에 대한 매뉴얼 적인 레이블링을 진행했다. DB에 저장되어 있는 채팅 및 자막데이터를 뽑아내어 csv 파일로 변환했고, 파일 내의 데이터에서 비속어 및 욕설이 등장한 경우에는 유해성을 나타내기 위해 label 컬럼에 1을 작성했다. 그 중 1.5만개에 대해서는 레이블링을 vote 방식으로 진행하였는데, 3명씩 짝을 지어서 각자의 레이블링이 유효한지를 투표로써 지정하는 방식으로 레이블링을 했다. 즉 하나의 데이터에 대해 3명이 고민해보고 2명 이상이 유해하다고 결정한 데이터에 대해서만 유해도를 1로 설정함으로써 유해도에 대한 공감대를 맞췄다.

수집한 데이터 중 유해 채팅의 비율은 5% 정도로써 매우 적었으며, 데이터셋을 모델에 그대로 입력했을 때 모든 채팅이 유해하지 않다고 예측하는 등 모델이 제대로 학습하지 못하는 문제가 발생했다. 따라서 유해 채팅 데이터를 데이터에 추가하기 위해 유해 채팅의 단어 순서를 바꾸는 데이터 증식(논문각주)[[3]](#footnote-3) 기법을 적용했고, 비속어 사전을 구축한 뒤 자동으로 유해 채팅만을 레이블링하여 데이터셋에 추가하는 등의 기법을 사용해보았다.

* + 1. **Data Argumentation**

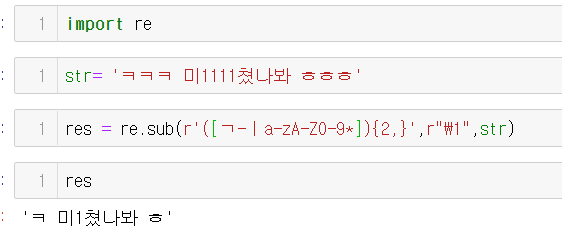
채팅 데이터를 크롤링 및 스크래핑을 시도했을 때 imbalanced한 데이터의 분포가 나타났는데, 이러한 데이터의 분포를 고려하지 않고 분석을 진행하게 되면 결과가 왜곡될 수 있는 가능성이 존재한다. 따라서 본 프로젝트에서는 데이터 증식을 통해 유해 데이터의 비율을 높이기로 하였다.

자연어처리에서 대표적인 데이터 증식 기법으로는 Synonym Replacement(SR), Random Insertion(RI), Random Swap(RS) 그리고 Random Deletion(RD) 등이 있다. 현재의 모델에서는 동의어 사전이 존재하지 않기에 SR, RI, RD 기법을 사용하는 것은 부적합하다. 따라서 RS 기법을 이용하여 문장의 순서를 랜덤으로 변경시킴으로써 데이터를 증식하였는데, 이때 모델이 적용될 도메인의 특성을 고려하여 알고리즘을 작성하였다. 예를 들면 ‘아저씨 발 냄새나요’라는 예제를 생각해보자. 이를 일상에서 접한다면 단어 그대로의 뜻으로 받아들여서 아저씨의 발에서 냄새가 나는 상황을 표현하기 위한 문장으로 받아들일 것이다. 하지만 직설적인 욕을 필터링하는 채팅 데이터의 특성을 고려해본다면 다분히 의도적인 문장으로 해석될 수도 있다. 비록 ‘씨’와 ‘발’이 띄어쓰기로 구분되어 있지만, 이 두 단어를 합치면 욕설이 되고 일반적으로 채팅 상에서 냄새를 맡고 표현하는 상황은 드물기 때문이다. 따라서 이러한 특성을 고려하여 띄어쓰기를 없애고 단어에 일정한 순서를 부여함으로써 데이터 증식을 구현했고, 위의 방법을 이용하여 유해 채팅 데이터 6만3천233개의 유해 채팅데이터와 10만1천041개의 정상 채팅 데이터로 구성된 데이터셋을 구축했다.

**2.2) 토큰화**

2.2.1) 형태소 분석 기반

한글 형태소 분석 오픈소스 라이브러리인 KoNLPy의 Twitter(Okt) 형태소 분석기를 이용하여 형태소 단위로 토큰화를 진행했다. 하지만, 몇몇 채팅 데이터에 대해서는 의미 단위로 토큰화를 할 수 없었다. 대부분이 ㅋㅋㅋ ㅎㅎㅎ 또는 숫자와 특수문자로 작성된 경우로 인해 발생했는데, 이때는 다음과 같은 정규표현식을 이용하여 전처리를 진행하여 반복된 자음들에 대해서는 한 개만은 남겨두고 그외의 중복된 값들은 날리게 하였다.. 예 re.sub(r'([ㄱ-ㅣa-zA-Z0-9]){2,}', r"\1", str)



그런데 띄어쓰기를 무시하고 표준어를 지키지 않는 채팅데이터의 특성 때문인지, 이렇게 전처리를 진행하여도 형태소 분석기를 이용했을 때의 성능이 좋지 않았다. 특히 Okt는 작은 단위의 욕설을 토큰화 하지 못했는데, 이에 따라서 byte pair encoding 즉 BPE를 이용한 전처리를 시도하게 되었다.

2.2.2) Byte Pair Encoding[[4]](#footnote-4)(이하 BPE)

주로 한글 도메인에서 나타나는 채팅 데이터를 필터링해보고자 했기에 기반이 되는 한글을 살펴보았고, 특히 한글을 구성하는 규칙에 집중해보았다. 한글을 utf-8로 인코딩 하면 결과가 b'\xe3\x84\xb1'처럼 16진수의 숫자가 3개씩 나타나는데, 이를 10진수로 변환해면 일종의 규칙들이 나타났다. 우선 자음에 경우에는 ‘ㄱ’이 (227,132,178)에 매핑 되고, ‘ㄲ’이 (227,132,179)로 매핑 된다. 즉 마지막 부분의 숫자가 1씩 증가함에 따라서 ㄱ->ㄲ->ㄳ->ㄴ->ㄵ->ㄶ->ㄷ->ㄸ->ㄹ->ㄺ->ㄻ->ㄼ->ㄽ->ㄾ->(227,132,191)ㄿ으로 나타났다.

이때 나타날 수 있는 숫자의 최대값이 191이고, 192부터는(227,133,128)->ㄹㅎ->(227,133,129)->ㅁ ....(227 133 142)->ㅎ을 나타나는 것을 발견했는데 모음의 경우에는’ㅏ’가 (227 133 143)에, ‘ㅣ’가 (227 133 163)에 연관되는 것을 확인할 수 있었다.

다음으로 ‘가’ 부터는

가:234, 176, 128,

각:234, 176, 129

갂:234,176,130

갛:234, 176, 155

개:234, 176, 156~~

갿: 234, 176, 191 끝이고

걀:234, 177, 128 ~~

걿:234, 177, 191

검:234, 178, 128~~

겿: 234, 178, 191

곀:234, 179, 128~~

곿:234, 179, 191

관:234, 180, 128~~

꿿234, 191, 191

뀀:235, 128, 128~~

뿿235, 191, 191

쀀:236,128,128 ~~

쿿:236 191 191

퀀:237 128 128 ~~

힣:237, 158, 163 로 표현되었다.

결론:

1. 첫째자리를 제외하고 모든 자리는 191이 끝이다

2. 해당자리가 191이 넘어가면 그 직전 자리가 1오른다.

3. 해당자리는 191이 넘어가면 128로 된다

4. 따라서 이것이 순환되므로 규칙은 찾을 수 있을 것이다 다만 해당 글자 한 자 음소(초 중 종성)을 하나의 숫자로 매핑은 할 수 없다. ex) ㅏ->129(x) ㅏ-> (227,133,143)(0)

bpe 모듈 bpe\_module.py

|  |
| --- |
| def str\_to\_lst(char):  """  ex)"abcde"->['a','b','c','d','e']  """  lstt = []  [lstt.append(\_) for \_ in char]  return lstt |

|  |
| --- |
| def str\_to\_sep\_utf8lst(str):  """  ex)"abc"->[[b'\xea\xb0\x80'], [b'\xea\xb0\xb8'], [b'\xea\xb1\xb0'], [b'\xea\xb2\xa8']]  """  morped =[[\_.encode('utf-8')] for \_ in str\_to\_lst(str)]  return [[x for x in \_] for \_ in morped] |

|  |
| --- |
| def str\_to\_utf8lst(str):  """  ex)"abc"->[b'\xea\xb0\x80', b'\xea\xb0\xb8', b'\xea\xb1\xb0', b'\xea\xb2\xa8']  """  morped =[\_.encode('utf-8') for \_ in str\_to\_lst(str)] |

|  |
| --- |
| def str\_to\_sep\_numlst(str):  """  ex)"abc"->[[234, 176, 128], [234, 176, 184], [234, 177, 176], [234, 178, 168]]  """  morped =[[\_.encode('utf-8'),\_] for \_ in str\_to\_lst(str)]  return [[x for x in \_[0]] for \_ in morped]  return morped |

|  |
| --- |
| def str\_to\_numlst(str):  """  ex)"abc"->[234, 176, 128,234, 176, 184,234, 177, 176,234, 178, 168]  """  li=[]  morped =[[\_.encode('utf-8'),\_] for \_ in str\_to\_lst(str)]  sep\_num\_lst =[[x for x in \_[0]] for \_ in morped]  [li.extend(lst) for lst in sep\_num\_lst]  return li |

|  |
| --- |
| def num\_to\_char(lst):  """  ex) [111,222,333,444,555]->abcde  """  byt=b''  ntoh = [byt.fromhex(t) for t in [\_.split('x')[1] for \_ in [hex(x) for x in lst]]]  byt2=b''  byt2 = byt2.join(ntoh)  return byt2.decode() |

|  |
| --- |
| def lst\_to\_num(lst):  """  ['가','나']->[1,2,3,4,5,6]  """  encoded = [str\_to\_numlst(el) for el in lst]  bytearr = np.array(encoded)  return bytearr |

|  |
| --- |
| def lst\_to\_sepsumnum(lst):  """  ['가','나']->[[1+2+3],[4+5+6]]  """  temp =[str\_to\_sep\_numlst(x,True) for x in lst]  bytearr = np.array(temp)  return bytearr |

|  |
| --- |
| def sum\_238(a):  try:  temp =238\*238\*a[0]+238\*a[1]+a[2]-12889781  except IndexError:  print("한글이 아닙니다 리스트에서 한글 아닌것을 지우세요")  return temp |

|  |
| --- |
| def findchar(x):  """  use this func when sum 3 nums to mapping one char  input is sum and output is char  """  a=[1,1,1]  temp = x+12889781  a[2] = temp%238  temp =int(temp/238)  a[1] = temp%238  a[0] =int(temp/238)  return num\_to\_char(a) |
| def sepnum\_to\_char(lst):  """  input is 2d sum of num list and out put is string  """  temp = [[findchar(x) for x in lst[i]] for i in range(len(lst))]  res = [[reduce(lambda x,y: x+y,temp[i])] for i in range(len(temp))]  return res |

위에서 본 것처럼 bpe는 글자를 숫자로 매핑한다. 따라서 한글에 유효한 모델을 4개를 만들어보았다.

(1) 채팅 데이터에서 필요 없는 영어와 특수문자 숫자를 전처리를 통해 없애고 bpe를 활용하여 한 글자당 3개의 숫자로 나누어 시험했다.

(2) 첫번째 단계와 마찬가지로 전처리를 통해 중복된 영어 및 특수문자들을 제거하고 bpe를 활용하여 한글자를 238진법 계산을 통해 한 글자를 숫자 하나로 합쳤다.

(3) 첫번째 단계처럼 전처리를 진행한 후 나온 3개의 숫자를 127차원에서 각각 1로 매핑했다.

(4) 두번째의 방법에서 나온 값 하나를 그대로 매핑하지 않고, 결과로 나온 50만 차원에 대해 vocab을 생성하여 1만 1천 153개로 줄였다.

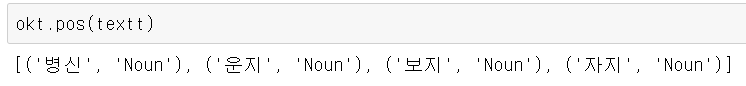
결론적으로 Byte Pair Encoding를 적용하였을 때 한 음절을 세 개의 토큰(Byte단위)으로 분리할 수 있었고, 자소가 변형된 욕설(ex. 씪빢 등) 에도 유해하다는 의미를 부여할 수 있었다. 또한 기본 형태소 분석기보다 4%정도 향상된 성능을 확인할 수 있었다.

2.2.3) nKT (nogada oKT)

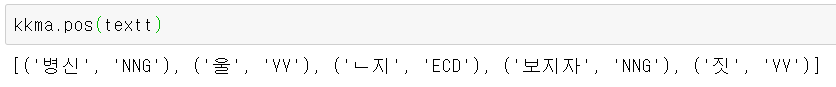
비속어를 탐지하여 영상을 필터링하기 위해서는 딥러닝 모델에 입력할 때 욕설의 특징이 잘 나타나도록 전처리를 수행해야 한다고 생각했다. 모델에 적합한 방법을 찾기 위해서는 여러가지의 형태소 분석기를 적용하여 성능을 비교해볼 필요가 있었기에, konply의 Okt와 Kkma, 그리고 카카오에서 개발한 khaiii등 대표적인 형태소 분석기를 사용해보았다.

예시로써 ‘병신운지보지자지’, 를 각각의 형태소 분석기로 돌려보았는데 그 결과는 다음과 같다.

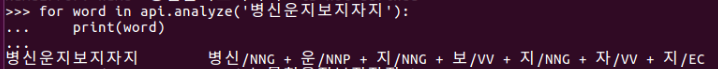
(Okt와 Kkma는 window, khaiii는 linux 환경에서 수행했다.)



Okt

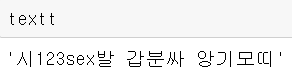


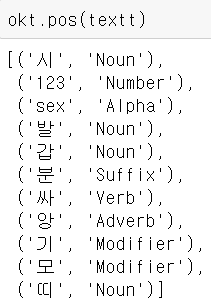
Kkma



Khaiii

위의 결과를 통해 인터넷 방송에서 빈번하게 나타나는 비속어들을 의미 단위로 분류하기 위해서는 트위터 기반의 형태소 분석기인 Okt가 적합한 것을 알 수 있다. 그러나 이 Okt조차도 실제로 사람들이 채팅방에서 사용하는 변형된 욕설을 잡는 데는 한계를 보였다.



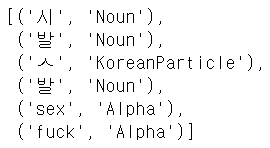


변형된 욕설을 okt에 적용한 결과

위의 결과를 통해 알 수 있듯이, ‘시발’이라는 단어 사이에 숫자나 영어가 들어가면 이를 ‘시’, ‘발’로 분리하여 원래의 의미를 잃어버리게 된다. 나름 신조어인 ‘갑분싸’, ‘앙기모띠’에 대해서는 하나의 고유명사가 아닌 각 글자별로 분석을 하고 있기에 만일 이 값들을 그대로 모델에 적용한다면 비속어에 대한 학습이 제대로 되지 않을 것이 분명해 보였다. 따라서 좀 더 비속어 및 신조어에 특화된 전처리가 필요하단 생각이 들어 Okt를 기반으로 하되 다음과 같은 기준을 만족시키는 분류 방법을 만들었다.

1. 하나의 단어 내부에 숫자나 영어, 특수문자가 들어가는 경우, 이는 대체로 해당 방송에서의 간단한 비속어 필터링을 피하기 위한 수단으로 사용되었다 간주하여 숫자는 버리고, 영어는 따로 분리하여 해당 단어와 영어를 구분한다.
2. 비속어의 학습을 용이하게 할 수 있도록 ‘조사’ 등 필요 없는 품사들을 걸러낸다.
3. 초성과 글자가 합쳐진 비속어는 KoreanParticle과 None로 분리하지 않고 ‘mixed’라는 품사의 하나의 단어로 분류한다.
4. Okt가 구분하지 못하는 신조어격의 비속어들과 비속어의 변형된 형태, 그리고 띄어쓰기로 분리된 비속어들은 ‘Except’라는 품사의 하나의 단어로 분류한다.

을 예시로 든다면 1번 조건에 의해서 숫자 123, 444는 버려지고, 한글과 영어를 구분하여 분리한다.



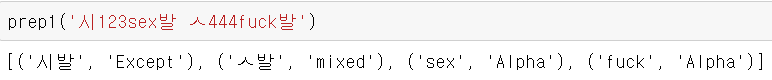
다음으로 3번 조건에 의해서 ‘ㅅ’와 ‘발’은 ‘ㅅ발’ 이라는 하나의 단어로 합쳐진다.



마지막으로 4번 조건에 의해서 띄어쓰기로 구분된 ‘시’, ‘발’을 ‘시발’로 합쳐진다.



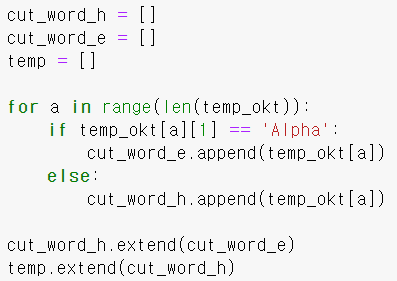
위의 4가지 조건을 처리해주는 함수를 prep1으로 정의하였다.



1번 조건은 정규식을 이용하여 구현하였다.



먼저 숫자와 특수문자를 제외해준다. 이와 별도로 ‘ㅋ’는 웃는 표현으로써 데이터의 상당부분을 차지하는 불용어로 판단했고, ‘ㅡ’, ‘ㅣ’는 단독으로 쓰일 때 의미가 없으며 주로 욕설 필터링을 피하기 위한 수단으로 사용되므로(ex: 시ㅡ발, 시ㅣ발) 같이 제외했다.



그 다음 한글과 영어를 구분하여 모든 한글을 먼저 배치한 후 그 뒤에 영어를 배치했다.

2번 조건의 경우 비속어 탐지에 필요한 품사들만 정의하고, 단독으로 쓰일 경우 조사(Josa)가 아닌 Noun, Verb로 분류되는 '은', '는', '이', '가', '자’를 제외시키기 위해 따로 정의했다.



3번 조건을 위해 초성과 글자로 이루어진 비속어 중 주로 쓰이는 단어들을 미리 저장하여 해당하는 초성과 글자가 오는 경우 이를 합치는 방식으로 구현했다. (자세한 내용은 prep1 함수 코드를 참고)



마지막으로 4번 조건은 신조어, 비속어의 변형, 띄어쓰기로 인해 비속어의 의미가 깨지는 단어들을 미리 지정함으로써 구현했다. (자세한 내용은 prep1 함수 코드를 참고)







Okt 분석 결과를 prep1 함수를 통해 비속어를 제대로 인식할 수 있게 해주는 위 모듈을 nKT 로 이름붙였다.

**2.2.4) AUTO LABELING**

위의 nKT를 이용해 사전 기반 자동 라벨링을 좀더 효과적으로 할 수 있었다. 예를 들어 ‘시1발’, ‘시 발’의 경우 사전 기반의 자동 라벨링을 하려면 사전 안에 ‘시1발’, ‘시 발’이 포함되어야 한다. 비속어의 변형이 다양한 한국어 특성상, 모든 비속어의 변형을 사전 안에 정의하는 것은 어려운 일이다. 하지만 nKT를 통해 나온 결과와 사전을 매칭시키면 ‘시1발’, ‘시 발’과 같은 단어들이 ‘시발’로 출력되기 때문에 사전에 ‘시발’ 한 단어만 정의하면 위의 예시들도 매칭시킬 수 있다. 물론 ‘시팔’, ‘씨팔’, 등의 변형된 비속어들을 사전에 정의해줘야 하는 것은 마찬가지이나, 사전에 정의해야 하는 비속어의 수를 줄일 수 있었다. (자세한 내용은 Auto labeling 코드 및 word-forbidden참고)

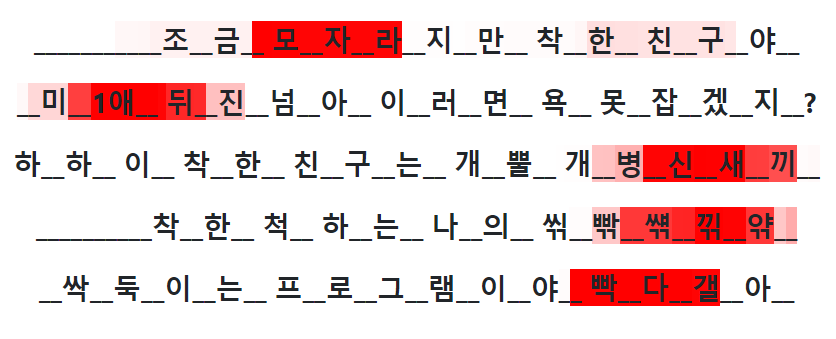
3. 모델

3.1) 1dCNN[[5]](#footnote-5)

자연어 처리에서 좋은 성능을 보이고 있는 1-dimension Convolution Neural Network를 적용했다. 이 모델을 적용하여 주변 문맥의 토큰을 바이트 단위로 이해하는 모델을 만들 수 있었다. 모델의 구조는 Embedding – Conv1D(window size : 16-8-8-4-4-2) 6개 layer – MaxPool – Dense로 구성되며 각 계층 사이에 dropout(0.5) 를 적용하여 과적합을 방지하고자 했다. 그리고 이 모델에서 Epoch을 20으로 주었을 때 Accuracy : 75%, F1-Score : 75%의 성능이 나타났다.

3.2) Self-Attention Bidirectional-LSTM[[6]](#footnote-6)

기존 1d-CNN 모델로는 모델이 제대로 유해 토큰에 대해 인식을 하고 있는지 검증하기 어려운 한계점이 있었다. 이러한 검증 문제와 함께 길이가 긴 채팅에 대해서도 성능을 높이기 위해 Self-Attention을 적용한 Bidirectional LSTM 모델을 적용해보았는데, 해당 모델의 self-attention을 이용하니 유해 토큰 제대로 인식하는지 검증할 수 있었다. 모델의 구조는 Embedding – 양방향 LSTM – Attention으로 구성되며 Attention과 LSTM hidden state의 곱이 출력 값이 된다. 수치로 성능을 살펴보니 CNN과 비슷하게 Accuracy가 73, F1-Score로는 73이 나타났다. 검증 이슈가 중요하다고 생각했기에 이 모델을 메인 모델로 삼아 사용했다.



검증 예시

4. 확장프로그램 및 UI

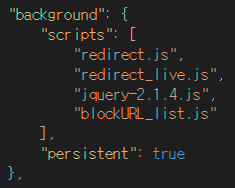
1.1) CHROME 확장 프로그램 소개 및 작동 방식

아프리카TV, 트위치, 유튜브의 각 인터넷 방송을 딥러닝 모델을 통해 분석한 결과 유해하다고 판단되는 영상들을 차단하는 기능을 수행한다.크롬 확장프로그램 ‘Cyberaffiti’는 다음과 같은 기능을 한다

1. 유해하다고 판단된 영상에 접속하지 못하도록 차단.
2. 각 플랫폼에서 유해하다고 판단된 영상들에 해당 영상의 유해도를 표시
3. 각 플랫폼에서 유해하다고 판단된 영상들이 사라지는 시각적 기법(fadeout)

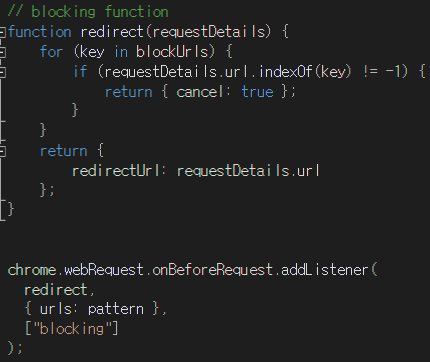
또한 1번 기능은 실시간 live영상 차단과 과거 업로드 된 영상의 차단으로 구분된다.

먼저 확장프로그램의 background 영역에서는 다음의 스크립트를 항시 실행한다.



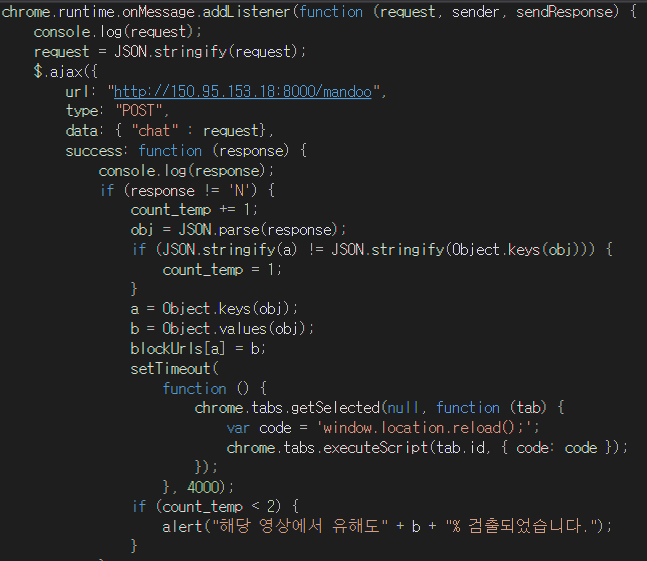
background

redirect.js는 chrome.webRequest.onBeforeRequest.addListener api를 이용하여 사용자가 영상을 클릭했을때, request를 서버에 보내기 전에 해당 request 정보가 차단할 영상의 URL과의 일치 여부를 확인하여 만약 일치한다면 이를 block하는 기능을 담당한다.

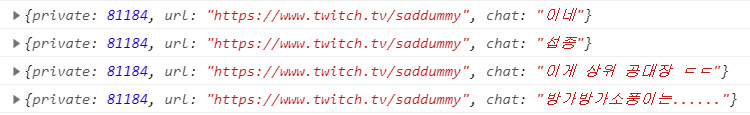


redirect.js

다음으로 redirect\_live.js는 content\_scripts의 live\_chat\_twitch.js, live\_chat\_afreeca.js에서 보내주는 실시간 live 영상의 채팅을 받아 이를 Cyberaffiti 사이트로 전달하여 딥러닝 모델에 대한 결과값을 반환 받고, 유해하다고 판단되면 blockURLs에 실시간 영상의 url 파라미터를 저장, 이를 통해 해당 live 영상을 차단하는 기능을 담당한다.

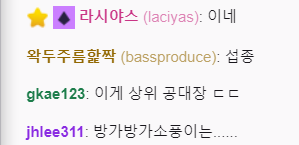


사용자가 live영상 시청 시 확장프로그램에서는 다음의 값을 서버로 보내준다.



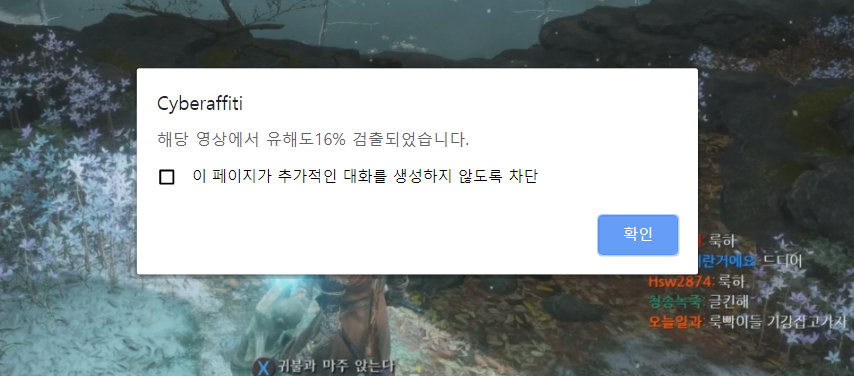
live방송시 확장프로그램이 서버에 보내는 값

아래는 live 영상의 실시간 채팅 내용이다.

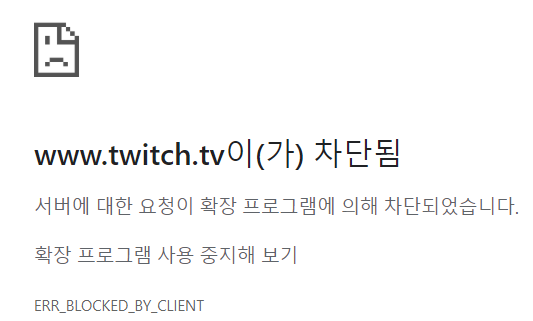


live 영상의 실시간 채팅 내용

서버에서 해당 live 영상이 유해하다는 결과를 확장프로그램으로 보내주면, 이를 사용자 화면에 표시해줌과 동시에 다음과 같이 해당 영상이 차단된다.



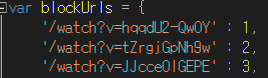
유해영상 발견시 차단 방식



유해영상 발견시 차단 방식

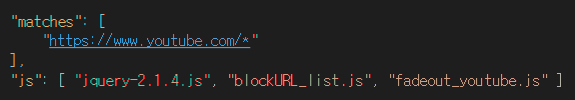
\* background 영역의 나머지 파일들 -

* jquery-2.1.4.js는 background에서 서버와의 ajax 비동기 통신을 위해 불러온다.
* blockURL\_list.js는 모델을 통해 유해도가 측정된 과거 영상들의 파라미터들을 저장하는 파일이다. key – value 방식으로써 차단할 영상의 parameter와 해당 영상의 유해도 수치를 딥러닝 모델에서 나온 결과를 서버에서 전달받는다. (2~3일 또는 일주일에 한번)

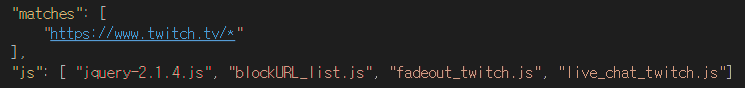


blockUrl

* content\_scripts 는 각 플랫폼별로 수행되는 스크립트를 선언해준다.



유투브



트위치

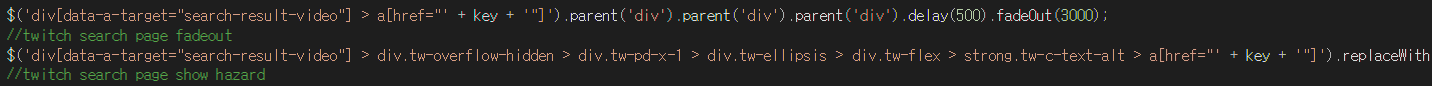


아프리카 티비

* “jquery-2.1.4.js”는 content\_script에서 시각적 효과인 fadeout 함수를 사용하기 위해 공통으로 선언했다.

- “fadeout\_youtube.js”, “fadeout\_twitch.js”, “fadeout\_afreeca.js”는 큰 틀에서는 기능이 동일하지만, 각 HTML 페이지 안에 정의된 CSS가 다르기 때문에 세부 코드는 다르게 구현된다.

- 각 플랫폼 별로 기능을 나눴기 때문에 fadeOut 스크립트의 연산 횟수를 줄여 확장프로그램 이용에 있어서 속도 저하를 최소한으로 하고자 했다.



🡪 코드가 잘 안보이니까.. (자세한 내용은 코드를 참고)



fadeout 효과가 적용되는 예시 – 유투브

플랫폼 특성상 트위치와 아프리카TV는 live 방송이 많기 때문에 live 방송의 채팅 데이터를 실시간으로 전달하기 위한 스크립트를 추가로 작성했다.

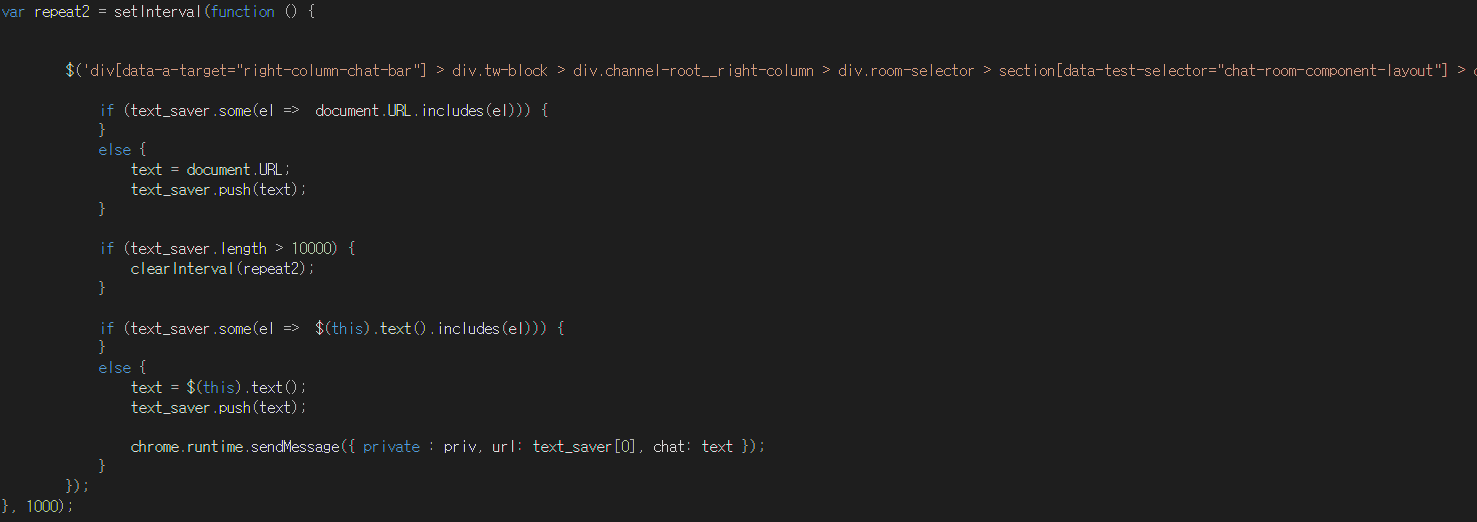
* live\_chat\_twitch.js에서 (live\_chat\_afreeca.js도 이와 같은 구조를 띈다.)

먼저 private 값을 1 ~ 10만 사이의 랜덤 한 값으로 부여하여 확장프로그램을 사용하는 여러 사람들을 구분하고자 했다. 이를 통해 서버에서 request에 대한 response를 보낼 때 해당 private 값을 가지는 사람에게 정확하게 응답할 수 있다.

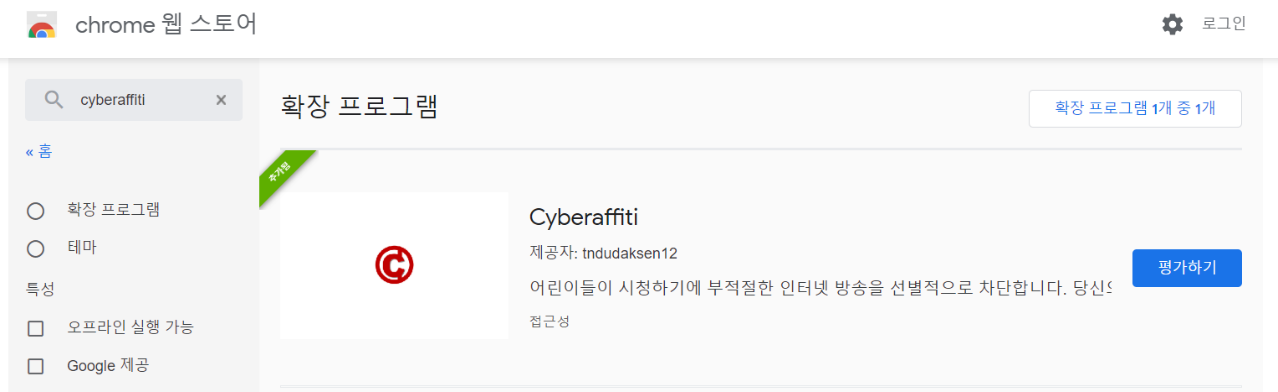


live\_chat\_twitch\_js에서 private

다음으로 1초마다 새로 갱신된 채팅 데이터를 background로 보내주는 코드이다. 맨 처음에는 해당 영상의 URL을 전송하고, 그 후에 채팅 데이터를 하나씩 보내준다.



해당 확장프로그램은 chrome 웹 스토어에 등록되었으며, Cyberaffit[[7]](#footnote-7)i 검색 후 이용해볼 수 있다.



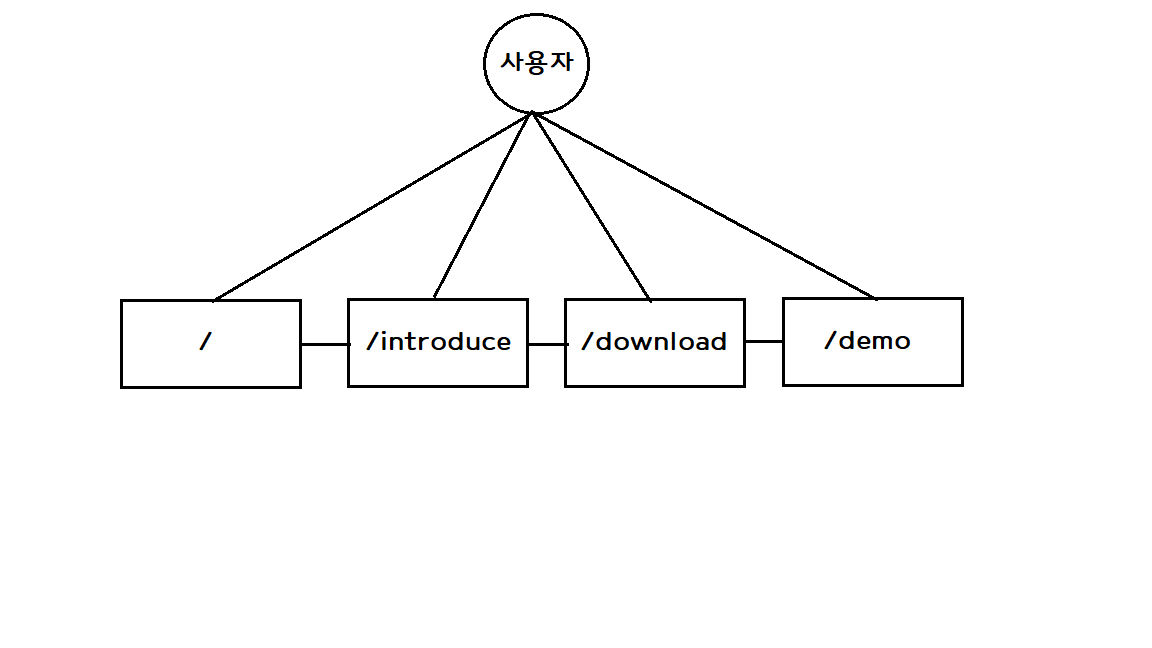
1.2) 확장프로그램 서버 연동

Chrome확장 프로그램과 서버가 Ajax 통신을 하여 확장 프로그램으로부터 사용자가 시청하고 있는 실시간 생방송도 차단을 해준다. 사용자가 시청하고 있는 생방송의 채팅을 전송해주면 서버에서 받아서 유해도를 측정하고 임계값인 18%가 넘어가면 즉시 필터링이 들어가며, 전체 채팅의 수가 100개가 넘어가기 전에 18개의 유해한 채팅이 나오는 경우에는 곧바로 영상을 차단하게 구현했다.

크롬 확장 프로그램에서 사용자에게 임의의 숫자 아이디값을 정해주고, 아이디 값과 시청하고 있는 스트리밍(생방송)의 url과, chatting 데이터를 ajax post 방식으로 전송해주면 서버에서 받아서 그 사용자가 보고 있는 영상의 채팅 데이터가 유해한지 판단하고 유해한 컨텐츠가 18% 이상 나오게 되면 즉시 차단하게 차단 url과 유해 정도를 전송해준다. (유해하지 않은 경우에는 N으로 신호를 보낸다.)

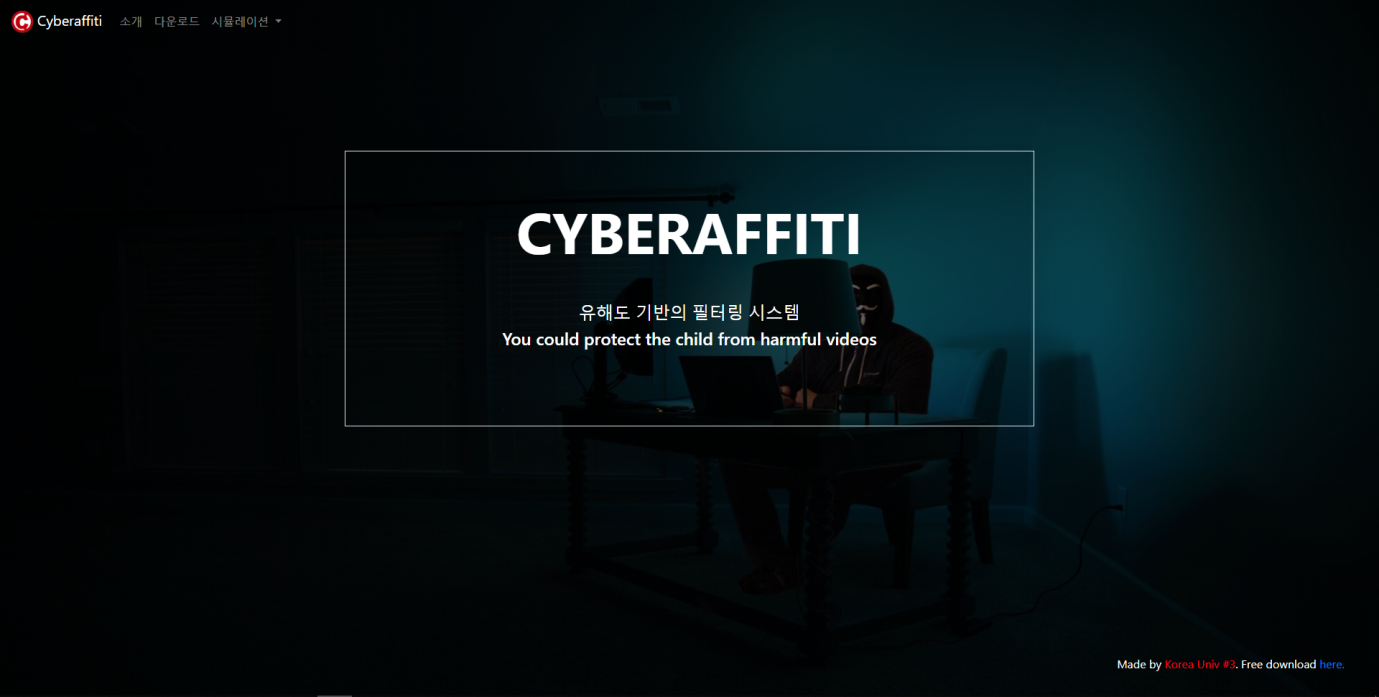
확장 프로그램의 경우 서버에서 주기적으로 검사를 진행하면서 차단 영상 목록을 업데이트하며, 관리자가 그 파일을 적용하는 방식으로 진행된다. 사용자의 입장에서 ‘사이버래피티’에 대한 UI를 사용하는 방식은 다음과 같다.

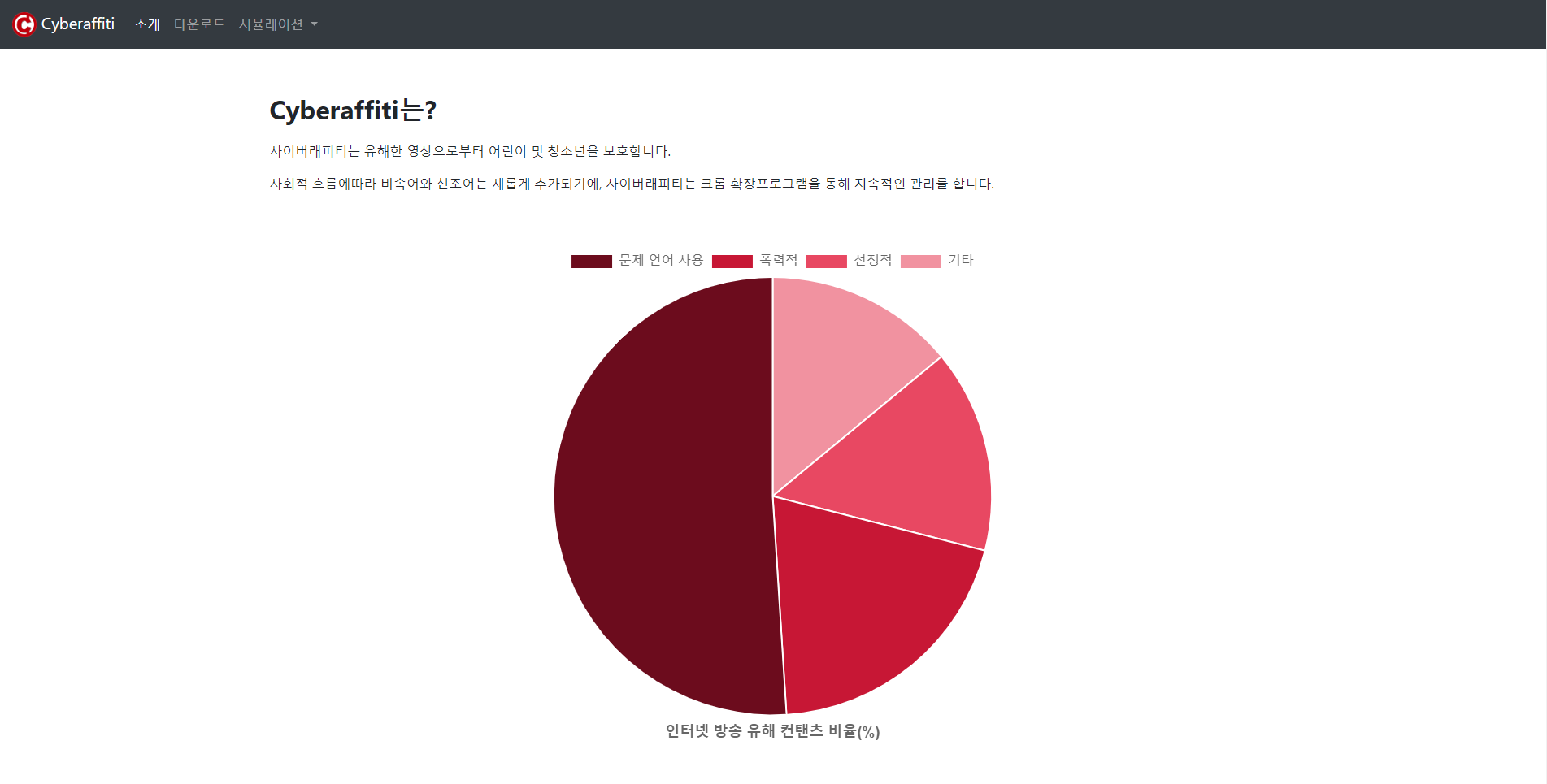
- (1) 150.95.153.18:8000으로 UI에 접속할 수 있다.

- (2) 메인 페이지에 접속한 후 최 상단에 있는 메뉴 바를 이용하여 원하는 페이지로 이동할 수 있다

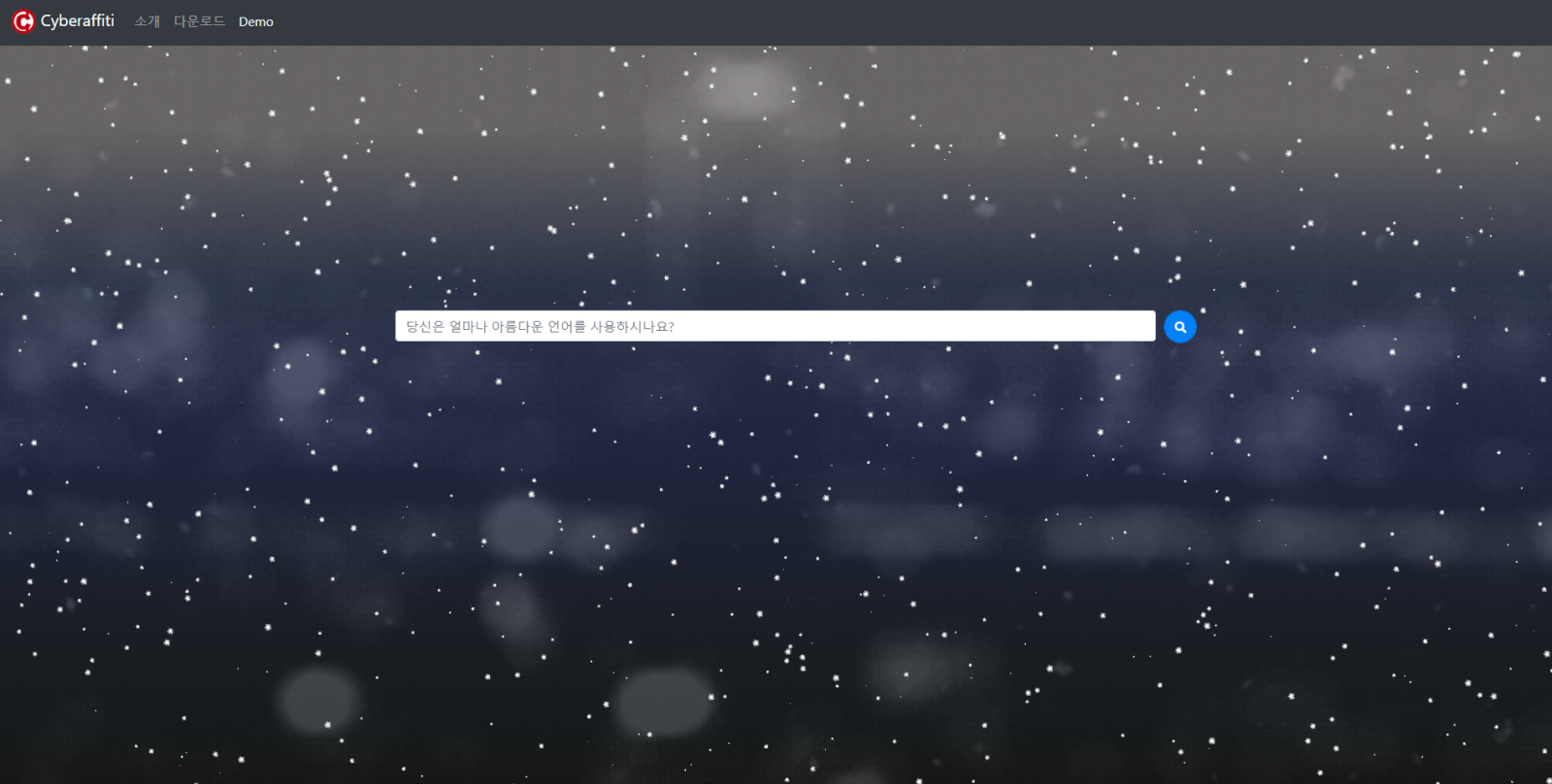
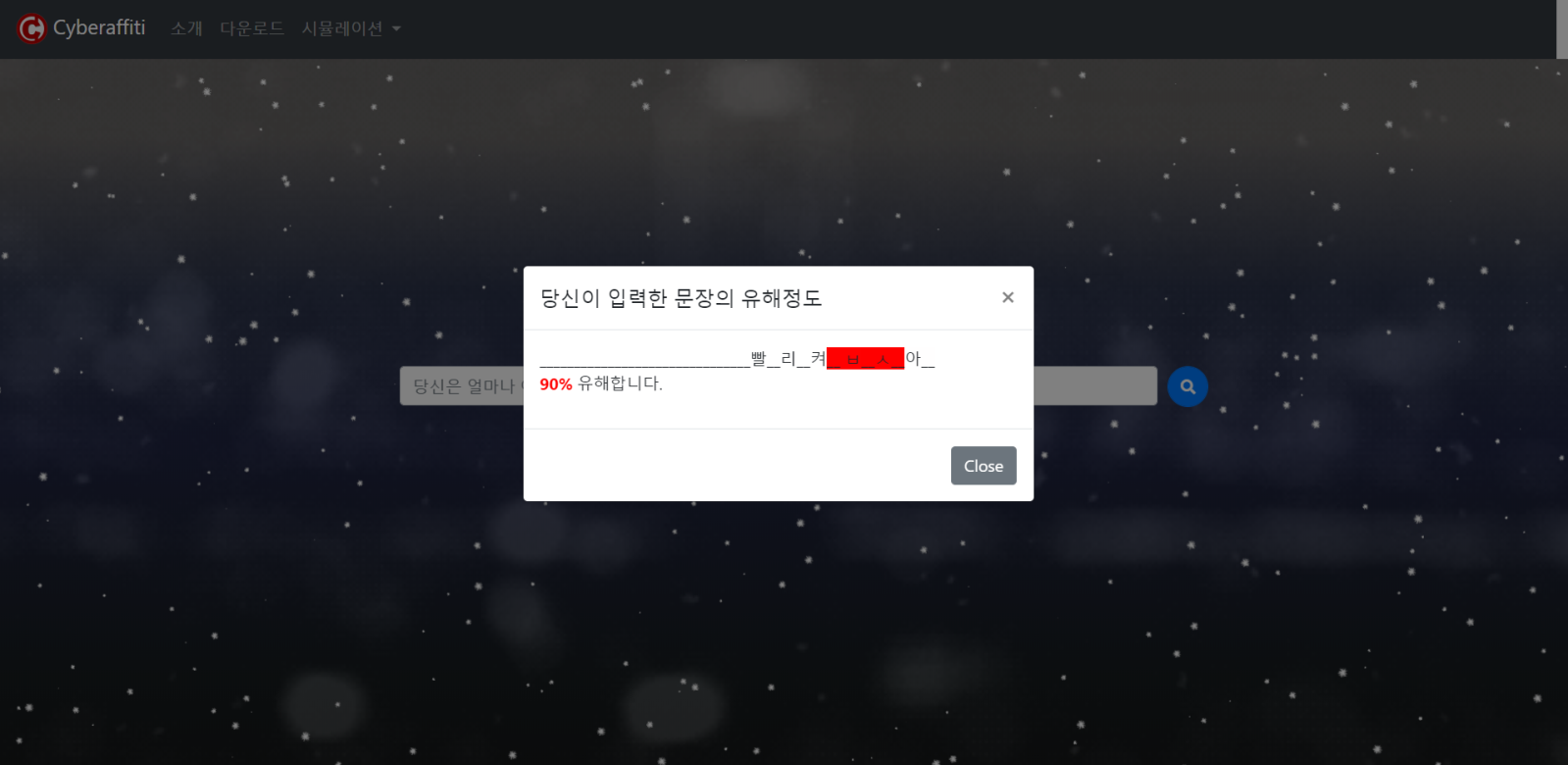
사이트 사용자 흐름도

-(3) 데모 페이지를 통해 문장이 얼만큼의 유해도로 필터링 되는지를 확인해볼 수 있다.



**150.95.153.18:8000로 접속한 메인 페이지**

사이버래피티 소개 페이지



유해도 측정에 대한 데모 화면

Ⅲ 결론

1. 개발 결과

트위치, 유투브, 아프리카 티비에서 bj와 영상들을 긁어오는 과정을 통해 비정형 데이터를 자유롭게 다룰 수 있게 되었고 전처리 과정을 통하여 어떻게 하면 성능을 높이는데 도움이 될 수 있을지에 대해 고민할 수 있었다. 이번 프로젝트를 수행하면서 가장 보람찼던 부분은 전체 각각의 과정속에서 점차 발전되어가는 모습을 볼 수 있었던 점이라고 생각한다.

* 데이터 수집을 함에 있어서는 아프리카, 유투브, 트위치 등에서 필요한 영상 및 bj를 긁어오면서 정규식과 map-filter-reduce를 자유자재로 사용할 수 있게 되었고, 전처리의 과정에서는 데이터 불균형을 잡아보기위해 업샘플링 및 다운샘플링을 시도하였으며 궁극적으로는 data argumentation을 통해 문제를 해결할 수 있었다.
* 토크나이즈 과정에서 okt, konlpy,khaii등의 대표적인 형태소 분석기의 사용을 통해 각각의 특성을 파악할 수 있었고 더 나아가서는 현재 도메인이 필요로 하는 채팅 데이터용 분석기를 구현하기 위한 nogada okt 및 bpe 방식을 고안해낼 수 있었다.
* 모델에서는 문맥 파악을 위한 1dcnn, 어떠한 단어로 인해 욕설이 탐지했는지 볼 수 있게 해준 attention, 한글 자소 단위를 쪼갬으로써 신조어 및 변형된 욕설도 탐지 할 수 있게 해주는 bpe를 구현할 수 있었고, 이로 인해 중간평가에서 56퍼센트 가량으로 측정되던 상과가 75퍼센트까지 향상되는 것을 볼 수 있었다.
* flask를 이용하여 UI를 개발하고 chrome 확장프로그램을 만들면서 사용자에게 유해 영상에 대한 필터링을 실질적으로 구현할 수 있게 되었다.

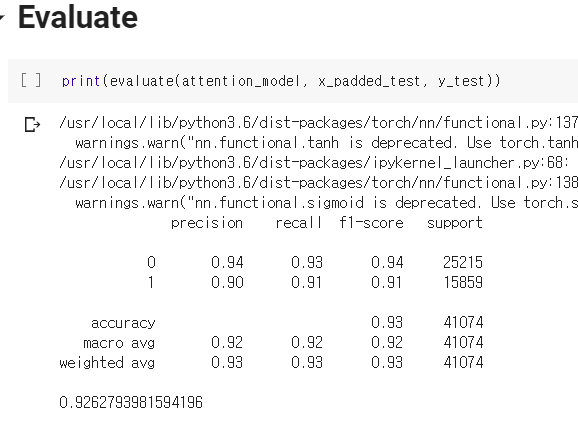
사실 초반에는 vod 영상에 대해서만 필터링을 수행하여 bj에 대한 블랙리스트 명단을 만들고, 이 리스트에 있는 사람들의 영상을 차단하여 실시간으로 방송되는 부분에 대해서도 필터링을 수행하고자 했었다. 그러나 프로젝트가 진행되면서 조원들의 실력이 향상되고 열정도 커져갔기에 불가능하다고 생각했던 실시간 스트리밍에 대해서도 서버와의 비동기 통신을 통하여 필터링을 수행할 수 있게 되었다. 게다가 무엇보다도 함께 만든 산출물이 크롬 확장프로그램 사이트에 탑재되어 배포되는 것을 볼 수 있는 것이 뿌듯한 것 같다. 이제 앞으로는 리뷰를 기반으로 하여 피드백을 수용하면서 점차적으로 형상관리를 해 나갈 예정이다.

1. **기대 효과**

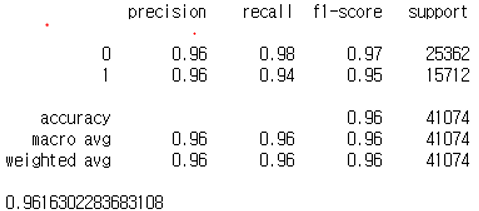
이번 프로젝트는 인터넷 방송 심의에 걸리지 않더라도 가치관이 아직 확립되지 않은 아이들이 시청하기에는 부적절한 영상들을 판별하여 필터링 해주기 위해 시작하였다. 이 확장프로그램을 이용한다면 자녀들이 건전하고 올바른 내용이 담긴 영상을 접할 수 있기에, 혹여나 아이가 유해한 영상에 접속하여 부정적인 영향을 받을 까봐 속앓이를 하시는 부모님의 걱정을 덜어드리는데 도움이 될 것이다. 현재는 데이터 수집이 초기 단계인지라 몇몇 방송인의 영상만 분석했지만, 2~3일에 한번씩 지속적으로 데이터를 수집하여 필터링 영상의 목록을 업데이트할 예정이기에 성능과 적용대상이 더 늘어날 것이다. 따라서 Cyberaffity 확장프로그램을 통하여 올바른 인터넷 방송 문화가 정착될 수 있기를 바라본다.

**부록**

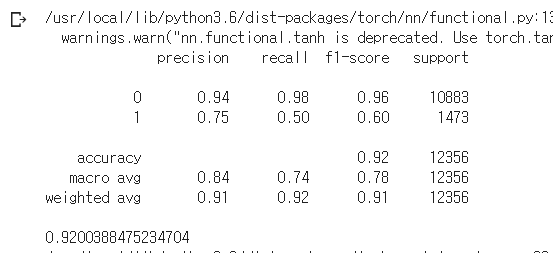
* BPE 테스트 결과



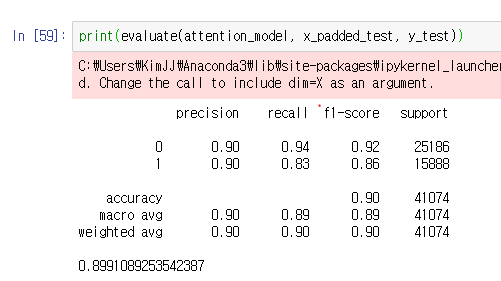
전처리 없이 에폭5로 bpe 돌린 경우



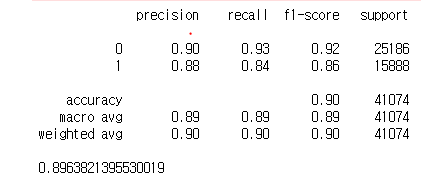
bp3의 결과로 나온 3개의 숫자를 하나로 합쳐서 에폭 5로 돌렸을때 임베딩



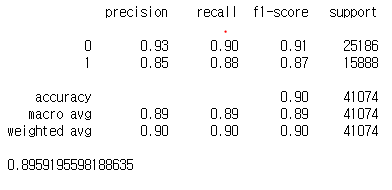
전처리 진행하고 에폭 5로 돌린 경우



전처리 진행후 에폭 10으로 bpe 진행한 경우



전처리 진행후 에폭 15로 bpe 진행한 경우



전처리 진행후 에폭20으로 bpe 진행한 경우

1. [https://namu.wiki/w/엘사게이트](https://namu.wiki/w/%EC%97%98%EC%82%AC%EA%B2%8C%EC%9D%B4%ED%8A%B8) [↑](#footnote-ref-1)
2. BJ: **B**roadcasting **J**ockey의 약자로써 방장의 개념. 아프리카티비에서 방송하는 스트리머들을 지칭하는 표현. [↑](#footnote-ref-2)
3. Sennrich et al.(2015) Sennrich, R., Haddow, B., Birch, A. 2015.*Neural Machine Translation of Rare Words with Subword Units.* arXiv:1508.07909. [↑](#footnote-ref-3)
4. Sennrich et al.(2015) Sennrich, R., Haddow, B., Birch, A. 2015.*Neural Machine Translation of Rare Words with Subword Units.* arXiv:1508.07909. [↑](#footnote-ref-4)
5. Kim(2014) Kim, Y. 2014. *Convolutional Neural Networks for Sentence Classification.* arXiv:1408.5882. [↑](#footnote-ref-5)
6. Lin et al.(2017) Lin, Z., and 6 colleagues 2017. *A Structured Self-attentive Sentence Embedding.* arXiv:1703.03130. [↑](#footnote-ref-6)
7. <https://chrome.google.com/webstore/detail/cyberaffiti/iglebngmpbfnkjcccpkdleefmdpekkoe?hl=ko> [↑](#footnote-ref-7)