**댓글분류 AI** - 나이브 베이시안 모델

2013011082 이상옥

**코드 장점**

성능: **성공률 82.54%,** 학습&분류시간 **20초** (본인 노트북 기준)

오염된 데이터도 많이 포함되어 있기 때문에 성능 90%이상은 사람이 해도 불가능해 보인다.

형태소분석기를 안 써서 비속어, 비문 등도 가리지 않음.

**코드 단점**

여러 단어가 띄어쓰기가 안 돼 있으면 구분을 못 한다. (단어 구분이 전적으로 띄어쓰기에 의존)

알고리즘 설명 / 코드 설명 / 성능향상을 위한 변화과정을 설명한다.

**1. 알고리즘적 접근**

**1.1. 전제**

각 댓글의 별점은 0 또는 1로만 분류한다. (중간단계 없음)

댓글은 여러 단어로 쪼개져 있다. 각 단어들을 독립적인 event로 보고 확률을 분석한다.

**1.2. 등장횟수가 0인 단어 처리 방법**

나이브 베이시안에서 등장횟수가 0인 단어가 나오는 경우, ‘한번도 일어나지 않은 사건’ 취급해 때문에 치명적이다. 확률 곱셈으로 계산하는데, 0이 중간에 섞이면 무슨 계산을 해도 안 통하기 때문이다.

Laplace Smoothing 대신 **[0.5 취급 정책]**을 사용한다. (이후 다양한 방법도 추가적으로 시험한다.)

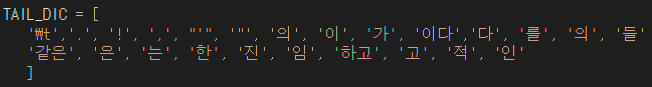
한번도 나온 적 없는 단어가 등장하는 경우, 이는 0.5번 등장한 단어로 취급하는 것이다.

예를 들어 1만번의 테스트동안 등장하지 않은 단어가 있다면, 앞으로도 1만번의 테스트를 하면 나올 확률이 있다고 판단하는 방식이다. Laplace Smoothing보다 합리적이라고 생각한다.

**1.3. 단어 split, 키워드 가공 기준**

단어 split 기준은 Spacebar만 사용한다.

“영화” 와 “영화를”을 동등한 키워드로 가공하기 위해 **[Tail Remove정책]**을 사용한다. 다음과 같은 tail이 단어 뒤에 붙어 있을 경우 해당 tail을 제거한다.



Tail을 제거 성공 시 recursive하게 Tail Remove를 시도한다.

**2. 코드 설명**

**2.1. 주요 인터페이스**

이 코드는 문장을 분류하는 나이브 베이지안 모델 분류 어디든 사용 가능하도록, 범용적으로 설계되었다. Rating이 n개여도 동작하며, 숫자일 필요도 없다. (단, 몇몇 디버그 메소드는 이를 보장하지 않는다.)

**NaiveBaisian.DefineRating(rating)**

새로운 Rating을 정의한다. Rating은 어떤 타입이든 입력 가능하다.

이미 정의된 Rating이라면, 학습 정보를 초기화한다.

**NaiveBaisian.InsertData(rating, data)**

해당 rating에 data를 입력한다. Data 문장은 단어단위로 분해되어 학습된다.

**NaiveBaisian.FindBestRatingFor(data)**

새로운 data에 대해 분류알고리즘 상 가장 확률이 높은 Rating을 return한다.

**글로벌 설정 변수**

TEST\_MODE\_OFF = True # =False면 ./ratings\_vaid.txt랑 비교하면서 정답률 체크

TEST\_NAME = "T07" # 디버그모드에서 만들어내는 결과파일 이름 앞에 붙음.

TAIL\_DIC = [ # 제거할 Tail 목록

'\t','.', '!', ',', "'", '"', '의', '이', '가', '이다','다', '를', '의', '들'

'같은', '은', '는', '한', '진', '임', '하고', '고', '적', '인', '~', 'ㅋ'

]

**3. 성능개선을 위한 변형 과정**

성능개선을 위해 학습데이터 파일을 출력하도록 했다.

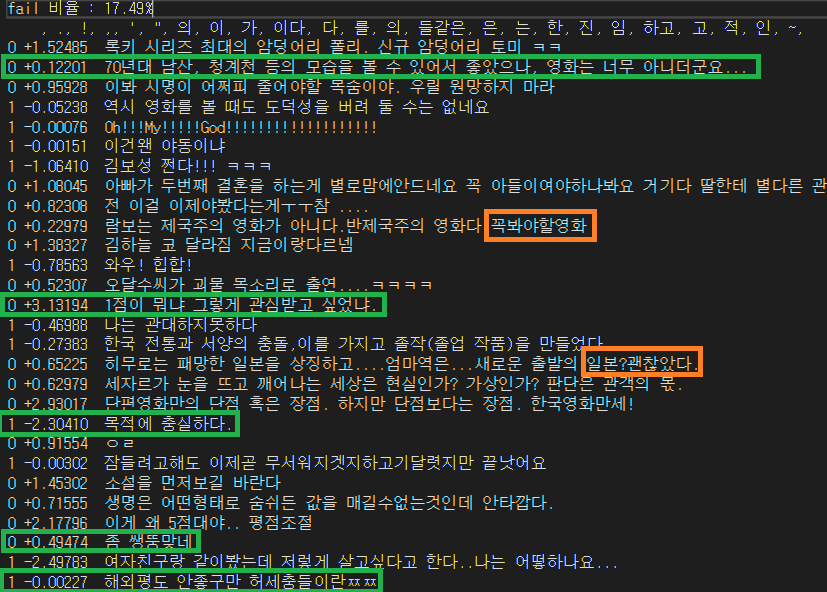
오른쪽은 최종 버전에서의 키워드 분석결과이다

Rate0와 rate1 두 경우의 키워드를, 발생 빈도수와 함께 나열해다. 오른쪽 키워드들을 바탕으로 여러 번 반복 사용되는 무의미한 tail을 수작업으로 추출해 제거했다.

[영화는] -> [영화] 같은 식으로 제거가 가능하다.

아래는 실패한 분석 결과들이다.

왼쪽부터 오답 / 확신률 / 원문 메시지다.

오염 데이터는 초록색으로 마킹했다.

**3.2. 변화과정**

|  |  |  |
| --- | --- | --- |
| Test | 성공률 | 실험조건 |
| - | 80.20 | 최초 빌드 |
| T01 | 82.30 | 최초 빌드를 바탕으로 tail 수동입력 작업을 반복하며 best solution 찾음. |
| T02 | 81.66 | T01 + 양쪽 레이팅에 모두 등장하지 않은 경우, 해당 키워드를 무시하도록 함 |
| T03 | 81.76 | T02 + T04 |
| T04 | 82.40 | T01 + 키워드가 Tail 자체인 경우 Tail을 제거하지 않도록 설정 |
| T05 | 81.77 | T04 + 양쪽 레이팅에 모두 등장하지 않은 경우, 레이팅 비율에 따라 달라지도록 함. |
| T06 | 82.38 | T04를 베이스로 [0.5취급 정책]에서 0.5를 변동시켜 보았다.  .50-> 82.40%  .45-> 82.37%  .42-> 82.42%  .41-> 82.43%  .40-> 82.43% best solution = 0.4  .39-> 82.38% T06모델에는 0.4를 적용한다.  .35-> 82.38%  .30-> 82.37%  .20-> 82.23% |
| T07 | 82.54 | T06을 바탕으로 tail 수동입력 작업을 반복하며 best solution 찾음. |

**3.3. 가능성 높은 개선방향들**

개발기간, 다른 과제, 시간자원 한계, 게으름으로 만들어 보지 못한 변형들.

**3.3.1. 띄어쓰기 외에 다른 특수문자로도 문장 구분**

[스펙타클한!!!!영화였다!!]라는 단어가 있다고 치자. 본 프로그램은 [스펙타클한!!!!영화]까지 밖에 처리를 하지 못한다. [스펙타클] [!!!] [영화였다] [!!]로 단어를 나눌 수 있게 하면, TailRemover가 [스펙타클] [!] [영화] [!]로 분류해 줬을 것이다. 이는 모두 빈번히 등장하는 키워드로, 분류에 도움된다.

**3.3.2. Training 데이터 중 등장 빈도수가 낮고 5글자 이상인 키워드를 형태소 분석기로 재가공**

외부 라이브러리로 한글 형태소 분석기를 쓴 친구들 말을 들어보면, 생각보다 정확하다고 한다. (정작 그러고도 성능은 크게 차이 안 나지만, 서로 장점이 다른 건 확실하다)

본 프로그램은 다음과 같은 단어를 [정말쓰래기같은영화돈이아깝다.]를 [정말쓰래기같은영화돈이아깝] 까지만 구분할 수 있다. (마침표 제거) 이런 경우에 형태소분석기를 사용해 키워드를 쪼개는 것도 좋을 것 같다. 재가공 시점은 Training 종료 후로 한다.