|  |
| --- |
| **자연어처리 및 정보검색**  **Project #1** |



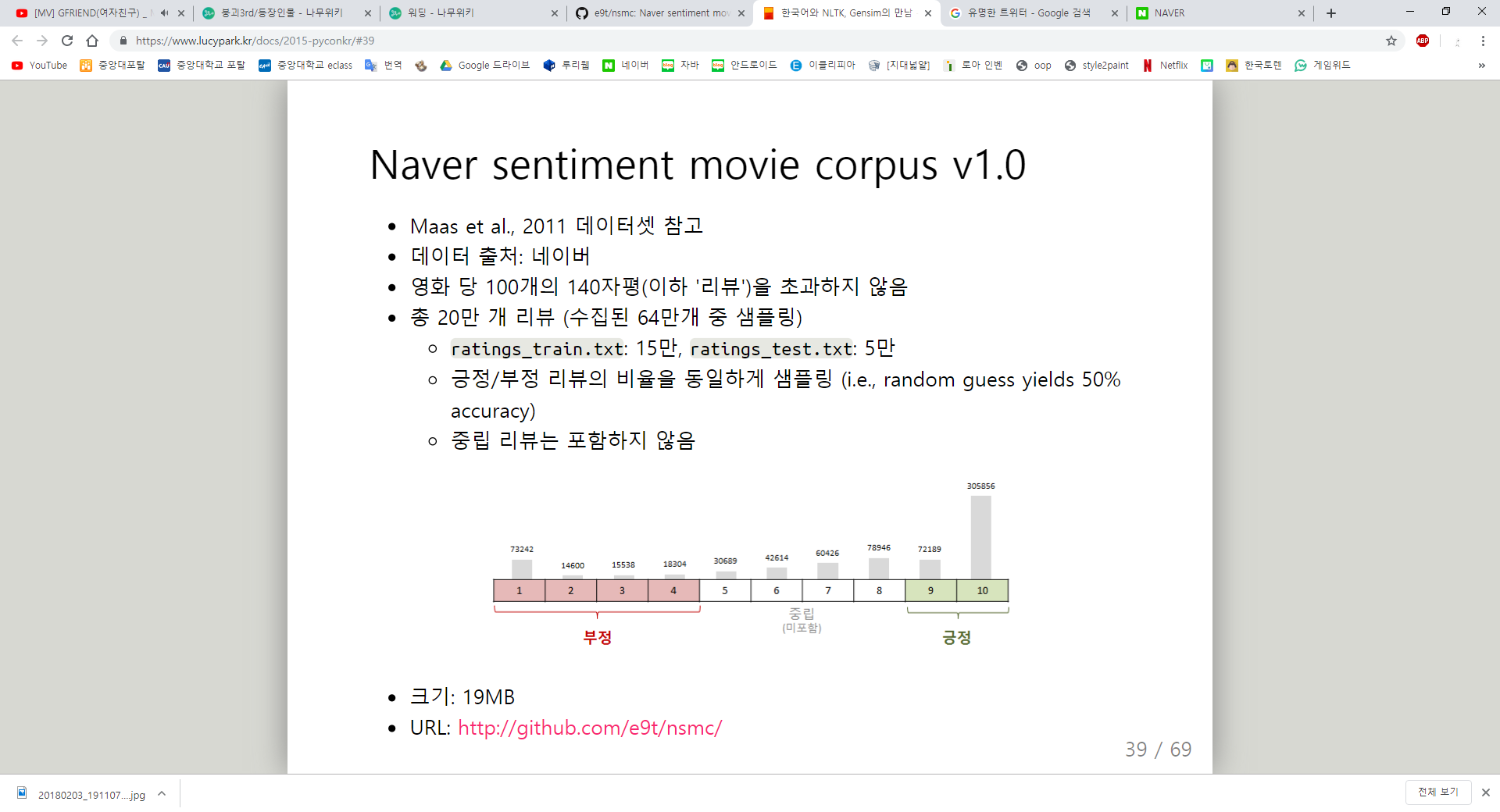
|  |  |
| --- | --- |
| **제출일** | **2019.06.15** |
| **팀** | **오늘도 밤샘각** |
| **이름** | **김현빈**  **김민조**  **백인혁**  **함지훈**  **손상혁** |
| **분반** | **01분반** |
| **담당교수** | **정재은 교수님** |

1. ***주제***

오늘도 밤샘 각의 프로젝트 주제는 영화 리뷰를 통한 감성 분석이다. 감성 분석이라 함은 주어진 문장에서 사람들의 태도, 의견, 혹은 성향과 같은 정보를 알아내는 기법이다.

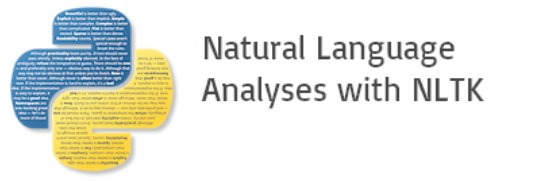
1. ***목적***

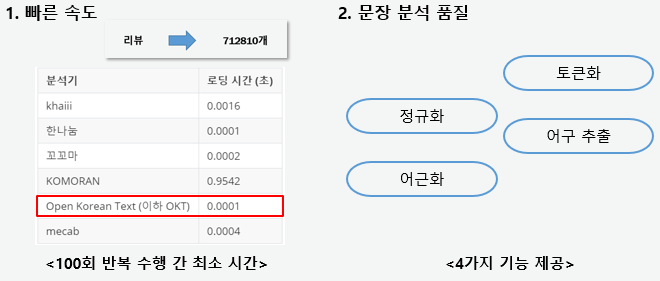
본 프로젝트의 목적은 영화 리뷰를 통해서 가장 유사한 리뷰를 찾아서 긍정 부정으로 구분해보자에 있다. 우리는 새로운 영화 리뷰가 들어왔을 때, 그 리뷰의 점수가 8점~10점일 경우 긍정으로 인지하고, 1점~3점일 경우는 부정으로 인지한다. 4점~7점은 중립인데 이는 우리가 본 프로젝트에 영향을 미치지 않도록 없는 값으로 처리했다.

1. ***데이터 및 분석***
2. **리뷰 자료 조사**

네이버 영화리뷰 데이터를 사용하여 영화 리뷰 데이터의 개수는 대략 총 71만개 정도이다.

조사한 리뷰에서 Rating 1~4 부분은 부정 5~8은 중립 그리고 9.10은 긍정으로 설정 되어 있었지만 우리는 Rating 1~3 부분은 부정으로 Rating 8~10 부분은 긍정으로 기준을 재 설정해서 리뷰가 긍정인지 부정인지를 판단했다.

1. **영화 리뷰 형태소 분석**

영화 리뷰가 한글이므로 영화 리뷰를 형태소 분석을 통해서 어떤 단어인지를 알 필요가 있어서 한글 형태소 분석인 KoNLPy을 사용하였고 여기서 Okt룰 통해서 형태소를 분해해서 저장하였다.

Okt를 선택하게 된 배경은 다음과 같이 빠른 연산속도와 정규화, 토큰화, 어근화, 어구 추출 기능을 제공해가지고 Okt방식을 사용하게 되었다. 수집한 영화 리뷰에서 형태소 분석을 통해서 생성된 단어는 총 237894개였다.-

1. **Rating 별 문서 만들기**

조사한 리뷰를 평점 1점에서 10점까지 구분해서 레이팅 별로 10개의 document text에 저장하여 10개의 텍스트 파일을 생성하였다.

1. ***진행 과정***

영화 리뷰 글을 분석했을 때 어느 모델이 더 높은 정확도를 내는지 알 수가 없으므로 테스트 데이터를 모아서 케이스를 나누고 여러 모델을 통해서 테스트를 진행해서 어떤 모델이 정확도가 가장 높게 나오는지 테스트 진행하였다.

1. **테스트 데이터**

Test Data는 6개의 Raing문서(1~3,8~10)에서 각각 1000개씩 data들을 뽑아서 총 6000개의 test data set을 생성하였다.

1. **CASE 1**

첫번째 case로는 전처리 하지 않은 237894개의 데이터를 활용해서 실험을 해 보았다.

1. **CASE 2**

첫번째 case에서 등장빈도가 50번 미만인 단어들의 경우 토큰화가 잘못된 단어일 경우가 높아서 이런 경우 역시 예외 처리가 필요했다. 등장빈도가 50번 미만인 단어들과 특수문자로 나누어진 단어들은 제거를 진행하였고 또한 의미가 중복되는 리뷰들을 찾아 분산되지 않도록 정리하는 정규화 과정이 필요해서 정규화를 진행하였다. 진행한 결과 11833개로 단어 개수가 줄어 들었다.

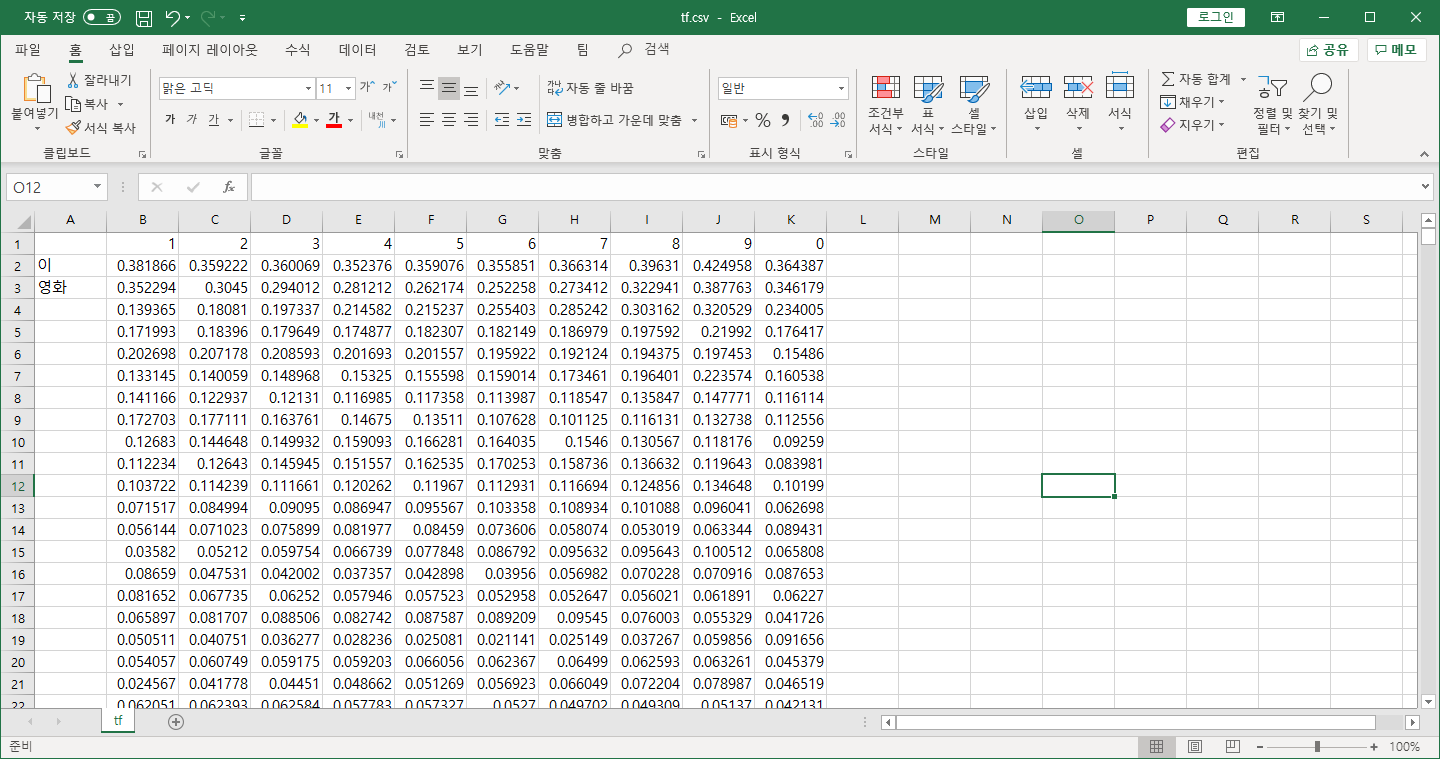
정규화 예시)

(ㅋㅋㅋ -> ㅋ) (ㅠㅠㅠㅠ -> ㅠ) 로 하나의 단어로 생성

1. **CASE 3**

두번째 case에서 Rating과 상관없이 유사하게 나타나는 단어들을 제거했다. 이런 단어들을 제거한 결과 총 5826개의 단어가 되었다.

제거 예시)



단어 ‘이’와 ‘영화’ 같은 경우 모든 문서에서 높은 등장빈도를 가지고 있으므로 VSM에서 제외 시켰다.

‘이’ 같은 경우 모든 문서에서 35~ 42%의 빈도를 보였고

‘영화’ 같은 경우 모든 문서 문서에서 25~ 38%의 빈도를 보였다.

1. **유사도 분석**

처음에는 TF-IDF 방식에서 Cosine Similarity방식을 사용하려고 하였으나 단어의 80%가 모든 단어에서 등장하게 되면서 idf의 의미가 없어졌다. 그래서 기존의 tf-idf값들 중 tf값만 사용하되, 모든 문서에서 공통으로 등장하는 단어들을 제외한 vsm을 생성하였다.

1. ***결과***

본 프로젝트는 TF(Term Frequency)방식에 따른 정확도를 확인해 보고자, TF 모델의 n(natural)방식과 a(augmented) 방식 2가지를 선택하여 비교 분석해 보았다.

Case1의 전처리 하지 않은 경우에는 TF의 n(natural)방식을 사용 했을 때는 67.8%의 정확도를 보였고 TF의 a(augmented)방식을 사용 했을 때는 66%의 정확도를 보였다. 비슷하지만 TF의 n 방식이 약간의 더 좋은 정확도를 보여주었다.

|  |  |
| --- | --- |
| **TF 모델** | **정확도 ACC** |
| Tf = n | 67.8% |
| Tf = a | 66% |

**<표1 - CASE1의 TF 방식에 따른 정확도>**

**<그래프1 - CASE1의 결과 그래프>**

Case2의 특수문자 제거 및 정규화 처리를 한 경우 TF의 n(natural)방식을 사용 했을 때 68.5의 정확도를 보였고 TF의 a(augmented)방식을 사용 했을 때는 68.8%의 정확도를 보여주었다. 2 방식의 차이는 아주 미묘한 차이를 보였다.

|  |  |
| --- | --- |
| **TF 모델** | **정확도 ACC** |
| Tf = n | 68.5% |
| Tf = a | 68.8% |

**<표2 - CASE2의 TF 방식에 따른 정확도>**

**<그래프2 - CASE2의 결과 그래프>**

Case3는 case2에서 레이팅과 상관없이 유사하게 나타난 단어들을 제거한 경우 인데 n 모델에서는 74.4%의 정확도를 보였고 a 모델에서는 75.1%의 정확도를 보였다.

|  |  |
| --- | --- |
| **TF 모델** | **정확도 ACC** |
| Tf = n | 74.4% |
| Tf = a | 75.1% |

**<표3 – CASE3의 TF 방식에 따른 정확도>**

**<그래프3 – CASE3의 결과 그래프>**

Case 1과 Case 2의 경우, TF-IDF 값을 적용하지 않고 TF값만을 이용하고 50번 이상 등장한 단어들만 모은 경우다 보니, 단순 TF값을 비교한다는 한계점을 보여주게 되었다. 또한 VSM(Vector Space Model)을 단순하게 빈도수를 이용해 줄인 건 정확도 면에서 크게 의미가 없었다. 반면에 case 3의 경우 IDF값을 사용하지 않는 대신, 모든 문서에서 공통적으로 비슷한 비율로 등장한 단어를 찾아 제거해 준 결과 case1,2에 비해서 정확도가 약 7% 상승하는 효과가 존재했다.