**인공지능2 과제 보고서**

**영화 리뷰 긍정 부정 판별하기**

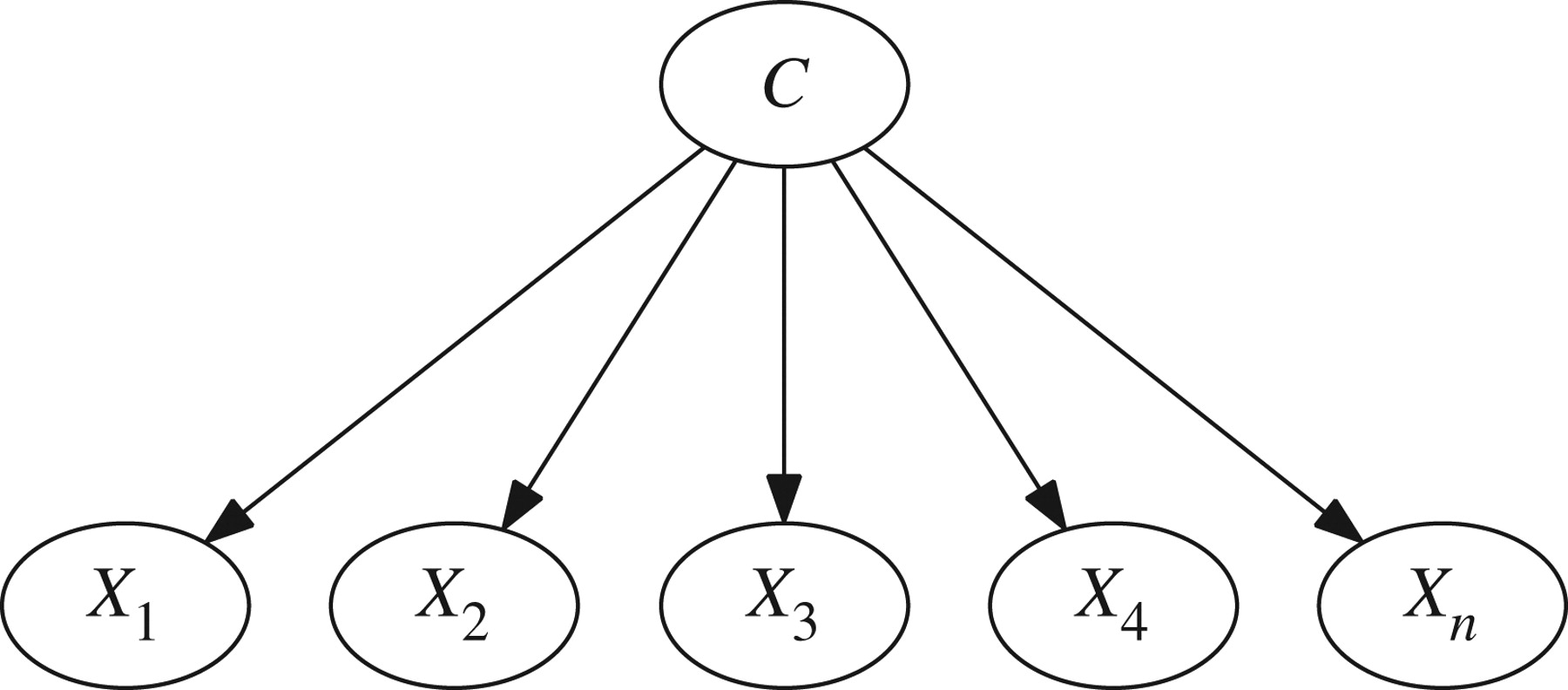
**2012003909 김승현**

1. **목표**

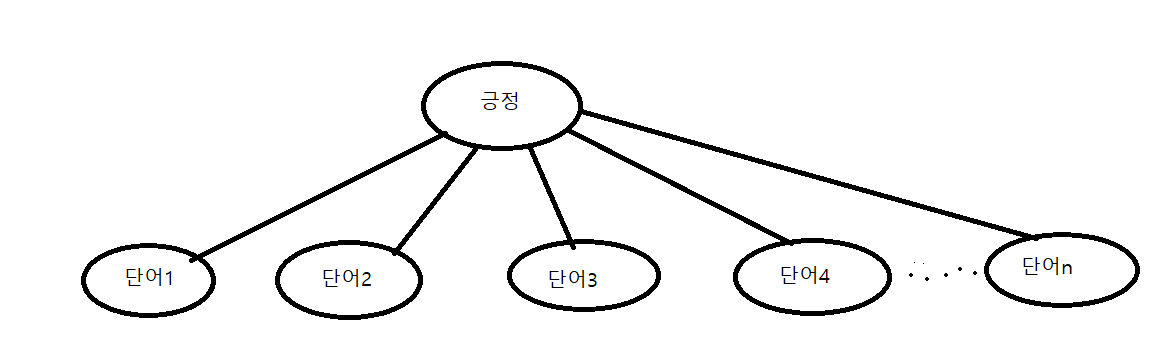
이번 과제의 목표는 대량의 학습데이터를 가지고 Naïve Bayesian 모델을 만들어 새로 들어오는 데이터의 긍정 혹은 부정을 판별하는 시스템을 구현하는 것입니다.

1. **모델 및 공식 정리**

Naïve Bayesian Model에 따르면 Cause 로 인해 발생할 수 있는 X 사건이 있으면, 각각의 X 사건은 조건부 독립을 만족합니다. 따라서 각각의 X마다 P(X|C), P(X|~C) 값과 P(C) 값을 알면 모든 확률 값을 계산할 수 있습니다.



여기서 Cause는 긍정 혹은 부정에 매칭시키고, X는 문장안에 있는 단어들로 매칭시킬 수 있습니다.



여기서 시스템의 목표는 단어1~단어n 까지의 상태가 주어지고, 이 상태를 A라고 가정할 때, P(긍정|A)와 P(부정|A)를 계산하여 이를 대소비교한 다음, 긍정 혹은 부정을 판정하는 것입니다.

그리고 그 것을 계산하는 식은 아래와 같습니다.

하지만 이 식에서 P(A)는 같은 값이므로 분자 값만 계산하면 되고, 여기서 각각의 확률 값들은 굉장히 작은 값이므로 도중에 0이 될 수 있습니다. 따라서 log 값으로 계산합니다.

1. **실행 환경 및 라이브러리**

사용한 실행 환경은 Windows 10 Pro, Python v3.7.1 에서 실험 했으며, Ubuntu 16.04 , Python 3.5.2 버전에서 실행 됨을 확인 했습니다.

또한 데이터 가공을 목적으로 외부 라이브러리인 **KoNLPY**를 사용했습니다. 이를 사용하기 위해서는 pip를 이용하여 별도의 설치가 필요합니다. 또한 KoNLPY에서는 JVM을 사용하기 때문에 이용하기 위해서는 JDK 설치와 JAVA\_HOME 설정도 필요합니다.

1. **‘구현**

구현한 python 안에는 여러가지의 함수들이 있습니다. 여기서 주목해야할 함수는 다음과 같습니다

|  |  |
| --- | --- |
| 함수이름 | 설명 |
| split | 문장을 여러 개의 단어로 나누는 함수 |
| train | 훈련할 파일을 읽어서 훈련 정보를 dictionary로 저장해 반환 |
| query | 훈련 정보를 바탕으로 한 문장의 긍부정을 판별하는 함수 |
| test | 테스트 파일을 읽어 리뷰들의 긍부정 결과를 새로운 파일로 저장하는 함수 |

구현하기 전에 앞서서 아래와 같이 고려할 것이 있습니다.

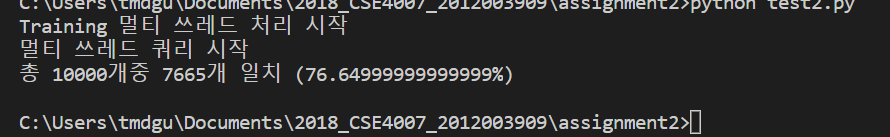
* 문장을 어떤 방법으로 단어로 나눌 것인가
* 긍정 혹은 부정 한쪽에서 아예 나오지 않는 단어는 어떻게 처리할 것인가 ( log(0) 값은 구할 수 없으므로..)
* 학습데이터가 많은데 최적화 처리는 어떻게 할 것인가

1. 문장을 나누는 방법

저는 여러가지 방법으로 문장을 나누는 방법을 시도 해보았습니다. 그 결과는 아래와 같습니다

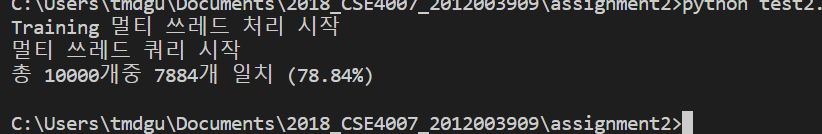
1. 가장 단순하게 공백 문자열로 나누는 방법

가장 간단하게 공백 문자열로 나누는 경우 ratings\_valid.txt 로 테스트 한 결과 대략 76.65% 정도의 정확도를 보였습니다.



1. ‘.’ 와 ‘,’를 무시하는 방법

사람에 따라 마침표나 쉼표를 무시하는 경우가 있고 아닌 경우가 있습니다. 따라서 이를 고려 했을 때 그것을 무시하는 것이 정확도를 높일 것이라 생각 했습니다.

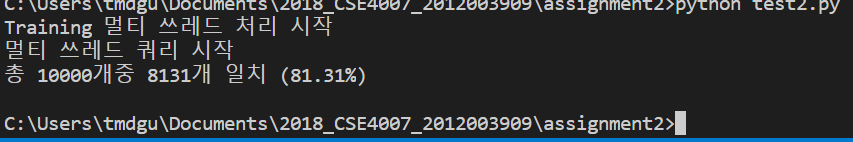


1. 여러가지 룰을 적용하는 방법

1-2와 비슷하게 문장을 쓸 때 구분되는 것들을 제 나름대로 정리하여 적용해보았습니다. 아래는 그 소스코드와 정리한 내용입니다.

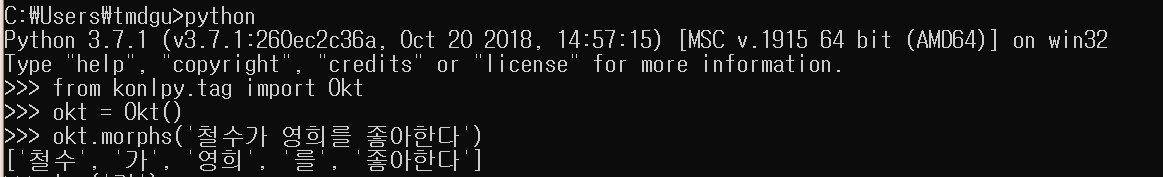
|  |  |
| --- | --- |
| 구분자 | 내용 |
| ㄱ-ㅎ (한글 초성만 쓴 경우) | 문장을 적을 때 초성만 써서 단어를 구성하는 경우가 많음 |
| 마침표, 쉼표 | 무시하도록 적용 |
| 완전한 한글 단어 | 구분자로 사용 |
| !나 ? | 구분자로 사용 |
| ; 의 경우 | 구분자로 사용 |

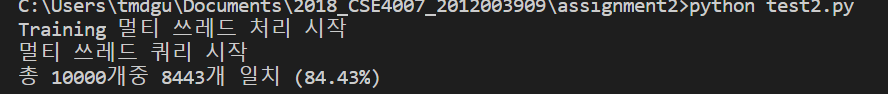
해당 규칙을 적용했을 때의 결과로는 81.31%의 정확도가 나왔습니다.



1. KoNLPY 라이브러리를 통해 단어를 나누는 경우

한글 형태소 분석기로 유명한 KoNLPY를 이용하여 문장을 형태소 별로 나눌 수 있습니다.

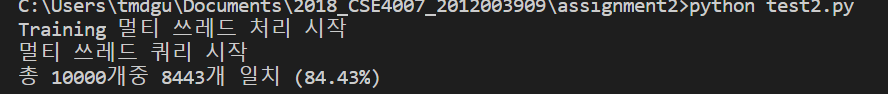




시간은 많이 걸리지만 84.43%로 다른 방법에 비해 훨씬 효과적임을 알 수 있습니다.

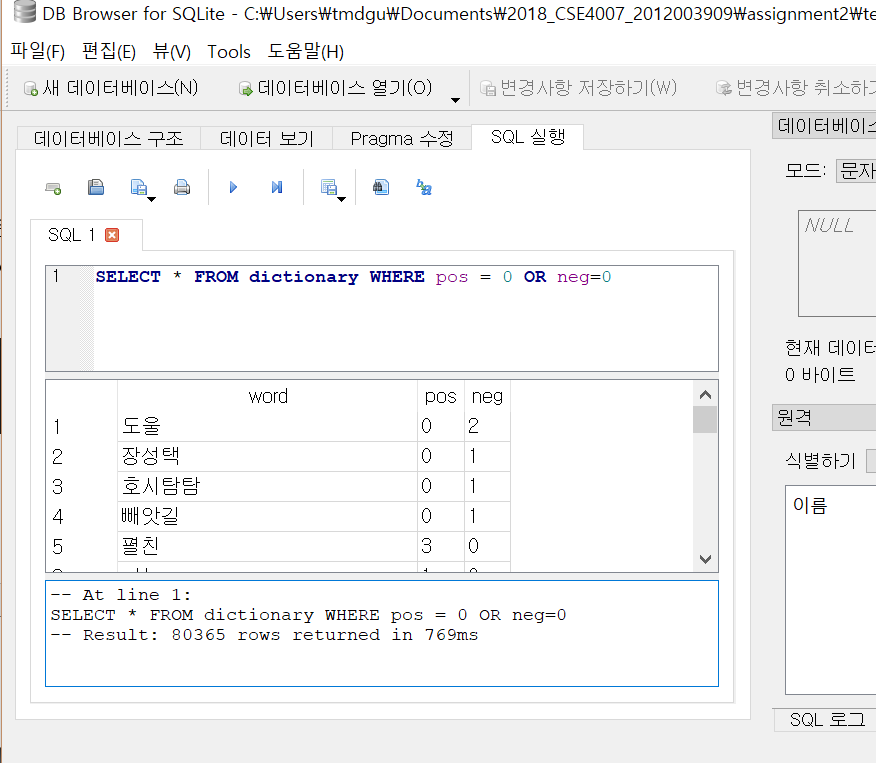
1. 0이 나오는 경우의 처리
2. 무시하는 경우

특정 단어가 부정에 나온 적 있는데 긍정에 나온 적 없는 경우, 혹은 긍정에 나온 적 있는데 부정에 나온 적 없는 경우, 가장 간단하게 생각해서 해당 단어는 Model 에서 제외하는 방법이 있습니다. 이 방법은 (1)에서 기본적으로 사용한 방법이므로 결과 값은 84.43%로 같습니다.

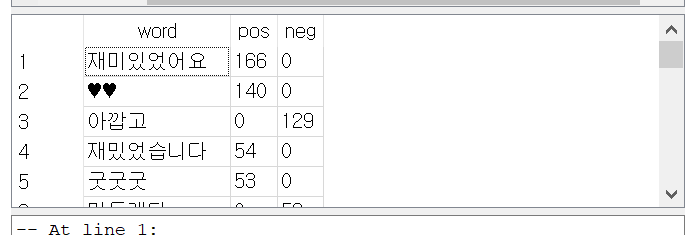


1. 상대적으로 작은 값 적용

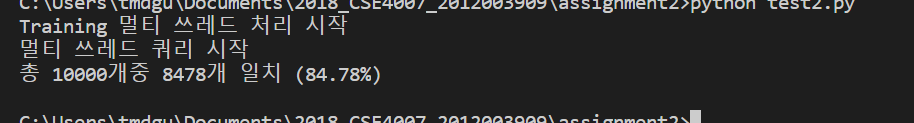
2-1번의 경우도 꽤나 좋은 성능을 보여주지만, 훈련된 데이터들(단어 들)을 SQLite로 저장하고 조회했을 때, 115991개의 단어 중 80365개라는 많은 단어들이 긍정 혹은 부정에서 한번도 나온 적이 없다는 걸 볼 수 있었습니다.



1~2번 쓰인 단어 같은 경우는 무시해도 그렇게 결과 값에 영향이 많이 없을 거라 생각했지만, 아래의 사진과 같이 한 부분이 굉장히 많이 언급된 단어들은 무시하면 고려가 되지 않는다는 문제가 있었습니다.

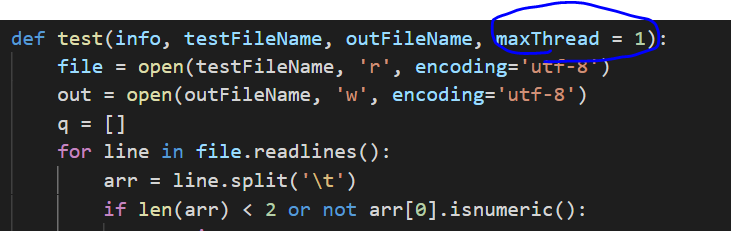


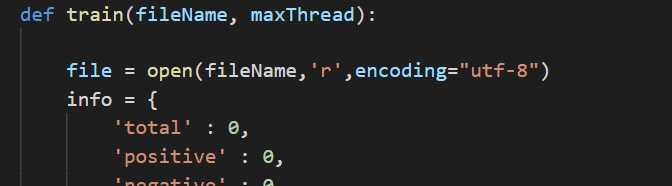
저는 이를 해결하기 위해 긍정 혹은 부정의 10% (0.1배)를 적용하는 방법, 혹은 그냥 1만 올려주는 방법으로 시도 해보았습니다. 그리고 10%만 적용하는 방법은 84.78%, 그냥 1만 올려주는 방법은 85.03%로 1만 적용해주는 방법이 더 효율적이라는 것을 알 수 있었습니다.



1. 학습데이터가 많아서 시간이 너무 드는 경우

처음에는 학습 데이터가 너무 많아서 학습하고 긍정/부정을 판별하는데 대략 10분이 넘게 걸리는 문제가 있었습니다. 따라서 저는 이걸 해결하기 위하여 멀티 쓰레드로 구현하도록 하였습니다.





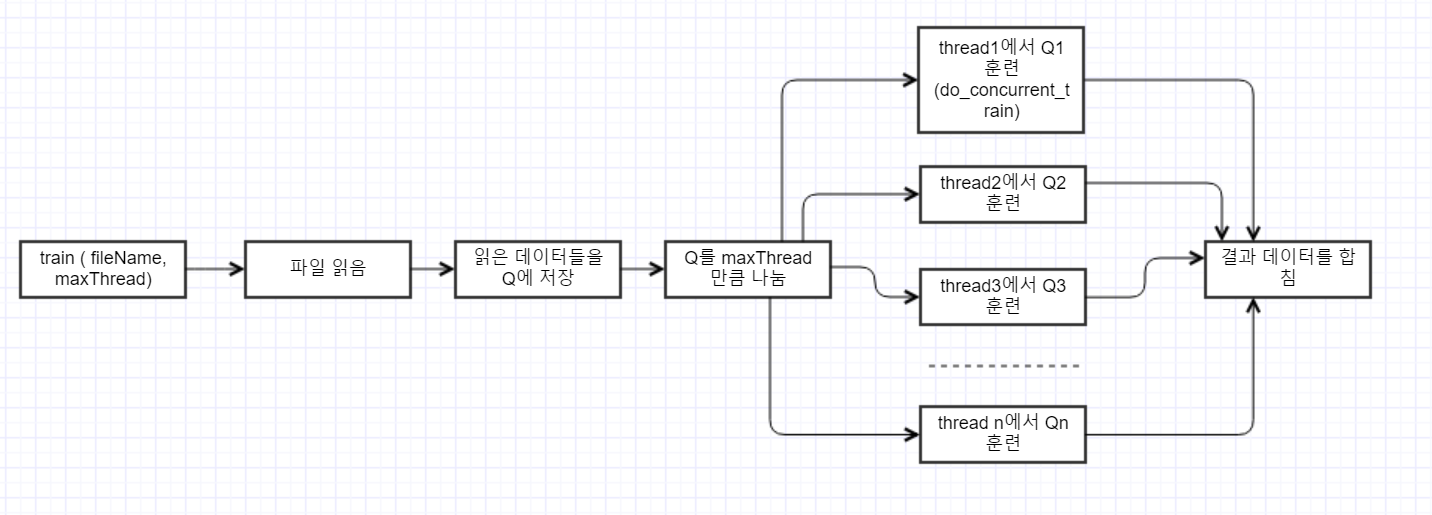
Test와 train 함수 안에 있는 maxThread 인자를 이용하여 쓰레드 수를 조정할 수 있습니다.

아직 여전히 6~7분 정도로 오랜 시간이 걸리지만 많은 시간을 절약할 수 있었습니다.

또한 한번 학습된 데이터는 trained.json으로 저장하고, 다시 실행할 때 이를 읽어서 학습은 하지 않고, 테스트 파일을 분류만 하는 작업만 수행하게 했습니다. 학습까지 하게 하려면 **trained.json을 지우고 수행**하면 됩니다.

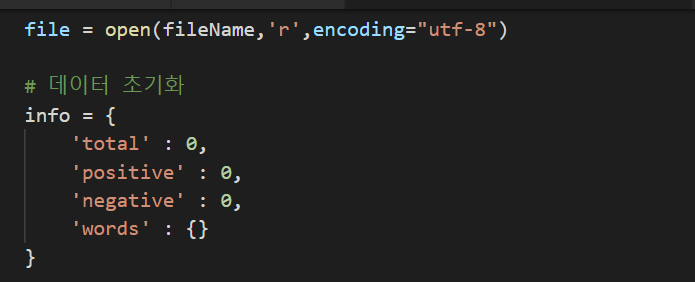
1. **Python 코드 설명**

파이썬 파일에는 기본적인 주석으로 설명이 되어 있기는 하지만, 가장 중요한 함수는 train과 test함수 입니다. train은 파일을 읽어서 학습데이터를 만들고, test함수는 테스트 파일을 읽어서 긍정/부정을 판별해 결과 값을 파일로 저장하는 함수 입니다.

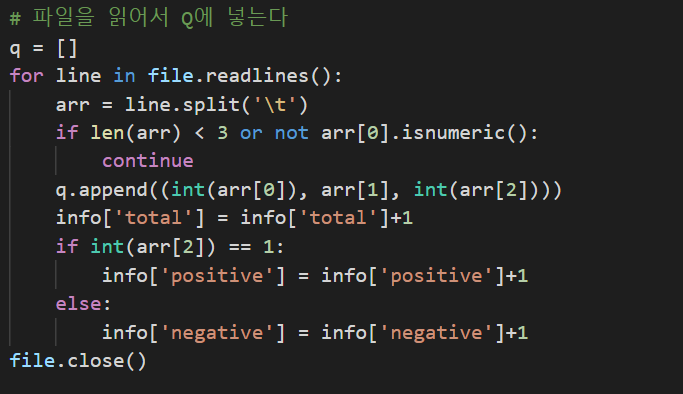


train 함수 같은 경우 위와 같은 순서도를 가지고 있습니다.

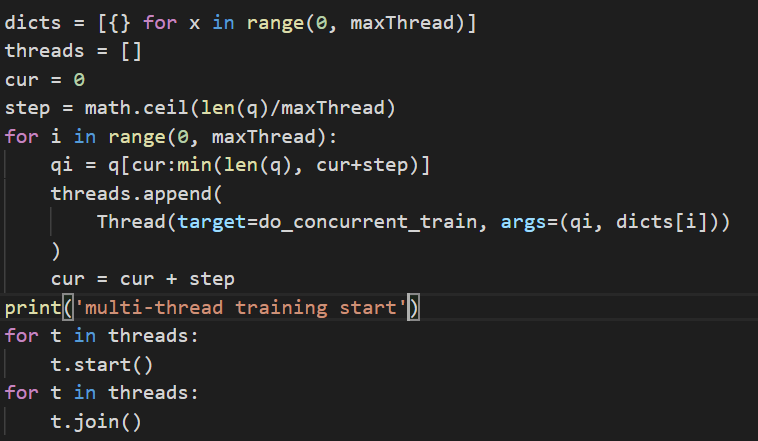
* 파일을 열고 학습 데이터 초기화 합니다.



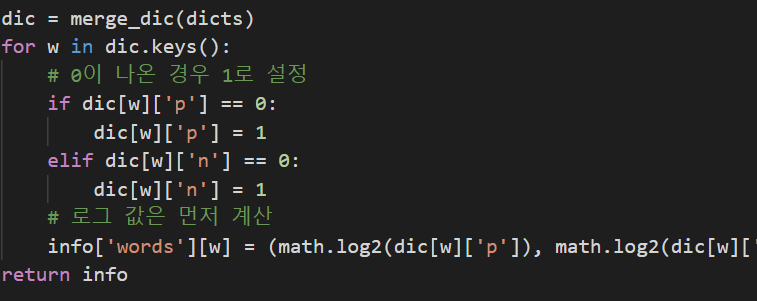
* 파일을 읽고 q에 넣습니다

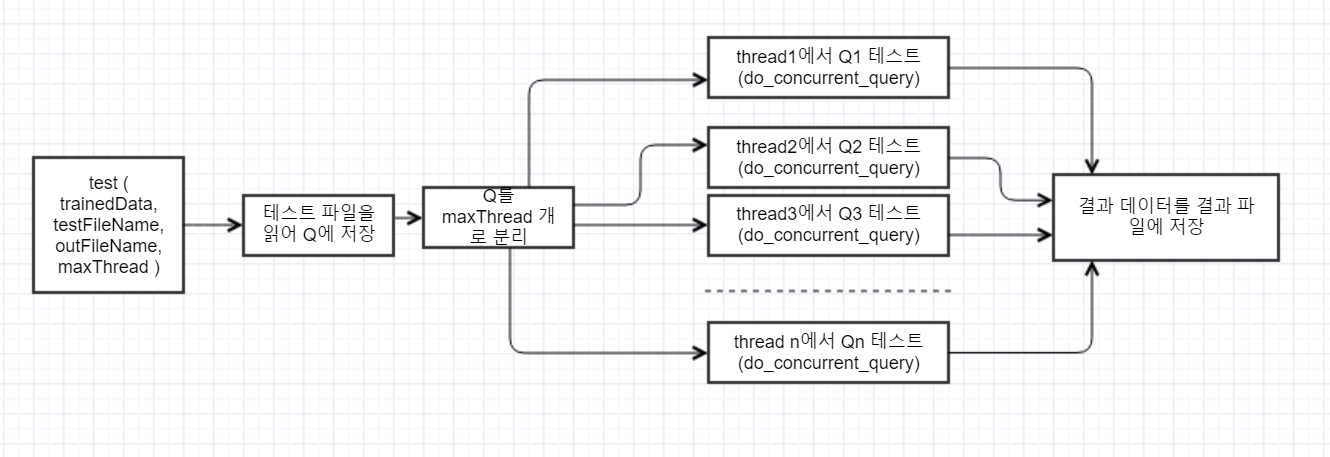


* q를 maxThread만큼 분리하여 쓰레드를 생성한 뒤 병렬처리 하도록 합니다. 병렬처리는 do\_concurrent\_train 함수에서 처리합니다.



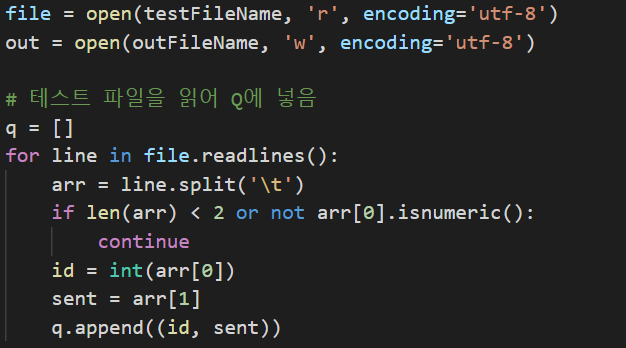
* 각각 병렬 처리한 값들을 합친 다음 결과를 반환합니다



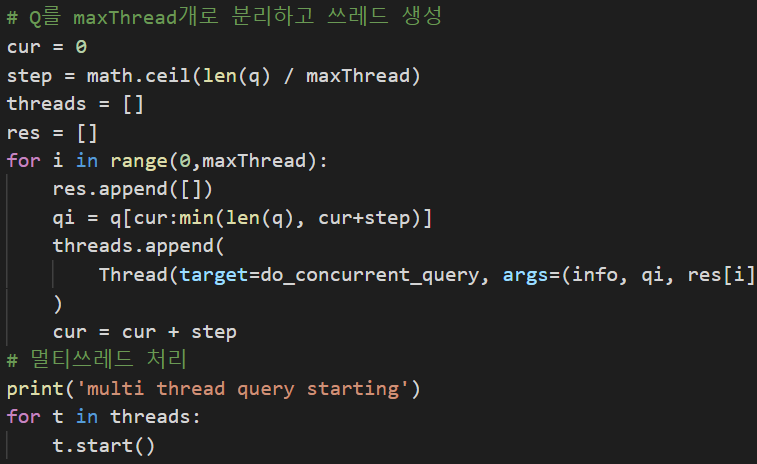


test 함수 같은 경우 위와 같은 순서도를 가지고 있습니다.

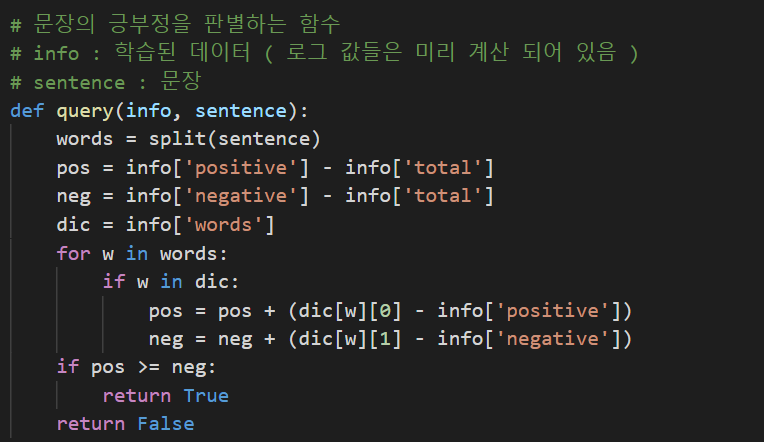
* 파일을 읽어 Q에 저장합니다.



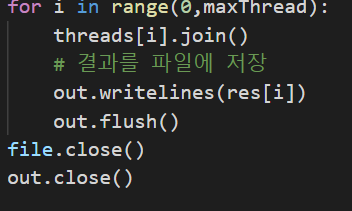
* Q를 maxThread만큼 분리한 뒤, 쓰레드를 생성해 병렬 처리하도록 합니다. 병렬 처리는 do\_concurrent\_query 함수를 이용합니다.



* Do\_concurrent\_query 함수 안에는 query 함수를 이용해 문장마다 긍정/부정을 판별합니다. query함수는 split 함수를 이용해 단어를 나누고 단어마다 log값을 계산해 긍정 값과 부정 값을 비교해, 긍정이면 True, 부정이면 False 값을 반환합니다.



* 병렬처리한 결과 값들을 결과파일에 저장합니다.



1. **결론**

수행 결과를 바탕으로 알게 된 사실은 Naïve Bayesian 모델을 이용하여 구현을 해도 85%라는 높은 정확도를 보여줍니다. 또한 중요한 것은 어떤 식으로 단어를 분리하냐가 정확도에 많은 영향을 준다는 것을 알게 되었습니다. 뿐만 아니라 이에 관하여 여러가지 방법을 찾아본 결과, Naïve Bayesian 모델 뿐만이 아니라, 아직 배우진 않았지만, SGDClassifier, doc2vec의 Logistic Regression, MLPClassifier, DNN (딥러닝), CNN 등을 활용하면 Naïve Bayesian 모델을 활용한 것보다 더 좋은 정확도를 낼 수 있다는 것을 찾아낼 수 있었습니다.