영화 리뷰 분석하기

우선 띄어쓰기로 간단하게 테스트를 해봤고
한글 자연어 처리 라이브러리 KoNLPy를 사용했습니다.
*나이브 베이즈 분류기는 각 단어가 독립임을 가정합니다.

보고서는 크게 아래와 같이 나뉩니다.

- 파일 읽기 (파일 읽기, 파일 전처리 과정(불필요한 문자 삭제))
- Train 하기 (형태소 분석기 선택, 우도 확률과 사전확률 구하기, 단어 별 우도 table만들기)
- Valid 로 점수내기 (답안지 tag와 내가 예측한 tag 비교해보기)
- 개선해보기
- Test 해보기

파일읽기

```
def text_read(where):
    f=codecs.open("./source/"+where+".txt",'r',"utf-8")
    line = f.readline()
    line2 = f.readline()
    list1 =[]
    while line2:
        line2 = line2.replace("₩r₩n",'')
        result2 = line2.split("₩t")
        #print(result2)
        list1.append([result2[1],result2[2]])
        line2 = f.readline()
    return list1
```

- 한글 파일을 읽기 위해 import codecs를 해줬습니다.
- 전처리 과정 불필요한 부분(₩r₩n)을 공백으로 대체합니다.
- ₩t으로 잘라줍니다.
- 읽은 파일은 id와 review, tag로 구성되어 있는데, review와 tag부분만 list1에 추가해줍니다.

Train 하기

학습하기에 앞서 우선 review부분을 어떤 형태소 분석기를 사용해서 단어를 분류할건지를 선택해야 합니다. (띄어쓰기,KoNLPy)

- 띄어쓰기로 분류할 경우

```
def count_words(out_list):
    counts = defaultdict(lambda: [0, 0]) # [neg , pos] checking
    for review, tag in out_list:
        if isNumber(review) is False:
            words = review.split()
            for word in words:
                counts[word][int(tag)] +=1
    return counts
```

- Split() 함수를 이용해서 word list를 만듭니다.
- Counts dict은 python에서 제공되는 dict입니다. Lambda [0 , 0]은 word가 neg에 나왔는지 pos에 나왔는지 개수를 세주는 변수입니다.
- 예시)

```
9100399 내가 이걸 보자고 토욜밤을 허무하게 보냈나..화가난다ㅜㅜ  0
'보냈나..화가난다ㅜㅜ': [1, 0],
```

띄어쓰기로 split()을 해서 그런지 '보냈다..화가난다ㅜㅜ'가 통째로 word로 처리되어 neg 부분에 +1이 된 모습을 볼 수 있습니다. 벌써부터 띄어쓰기의 한계가 보이기 시작합니다.

```
def word_probability(word_count,negnum,posnum,k):
    return [ (w,
              (class0+k) / (negnum + 2*k),
              (class1+k) / (posnum + 2*k))
             for w,(class0, class1) in word_count.items()]
```

- 이 부분은 우도 함수 테이블을 만드는 과정입니다.
- 작은 k값을 분모와 분자에 더해줌으로써 smoothig처리를 해줬습니다. 어떤 단어가 긍정 문서에 단 한번도 쓰이지 않더라고 우도 확률이 0이 되지 않도록 적당히 작은 상수를 더해줬습니다. 이걸 안 하면 값이 모두 0 이 나와서 쓸모 없어집니다.
- 테이블은 [단어, P(단어 | neg), P(단어 | pos)] 으로 구성되어 있습니다.

```
('보냈나..화가난다ㅜㅜ', 0.1875, 0.05555555555555555),
```

저 단어는 neg tag에서 일어날 확률이 pos 보다 높습니다. 라고 판단할 수 있습니다.

점수내기

```
def scoring():
    answer_list = text_read("test")
    answernum = file_len("test")
    corret =0
    for ans in answer_list:
        prediction =classify(ans[0])
        if int(ans[1])==prediction:
            corret +=1
    return corret / answernum
```

- 우선 정답지인 text를 읽어와 answer_list를 만듭니다. review, tag으로 구성되어 있다.

```
['아버지가 생각나네요...마음이 짠합니다..', '1'],
```

- classify함수를 이용해서 review의 tag값을 예측합니다. 그 값을 prediction에 넣고 정답지인 ans[1]와 내가 예측한 값이 일치하는지 안 하는지를 체크합니다.
- 맞은 개수 / 전체 개수로 맞은 비율을 return 해줍니다.

Classify 함수 분류기

```
def classify(query_review):
    return class0_probability(word_prob_table, query_review)
```

- 이 함수는 query로 들어온 review값을 파라미터로 받고 class0_probability() 함수의 리턴값을 돌려줍니다.

Class0_probability 함수

```
def class0_probability(word_prob_table, query_review):
    query_words = query_review.split()
    log_prob_if_class0 = log_prob_if_class1 =0.0
    for word, prob_if_class0, prob_if_class1 in word_prob_table:
        if word in query_words:
            log_prob_if_class0 += math.log(prob_if_class0)
            log_prob_if_class1 += math.log(prob_if_class1)

        else:
            log_prob_if_class0 += math.log(1.0 - prob_if_class0)
            log_prob_if_class1 += math.log(1.0 - prob_if_class1)

    prob_if_class0 = math.exp(log_prob_if_class0)
    prob_if_class1 = math.exp(log_prob_if_class1)

    if(prob_if_class0>prob_if_class1):
        return 0
    else:
        return 1
```

- 이 함수는 (우선 띄어쓰기)로 query_review을 나누고 각각의 단어와 매칭되는 우도 확률을 찾아 확률을 구합니다.
  - ◆ log계산을 하는 이유: 컴퓨터는 0에 가까운 부동소수점을 제대로 처리하지 못하기 때문에 우도의 곱은 log우도의 합으로 처리해줍니다. Import math를 사용합니다.

$$\prod_i P(\text{word}_i \mid \text{pos}) = \exp[\sum_i \{\log P(\text{word}_i \mid \text{pos})\}]$$

  - ◆ 추가로, 검증하려는 word가 우도 테이블에 없을 경우 해당 단어가 나오지 않을 log 확률을 더해 줌 log(1-나올 확률)로 계산해준다.
- Log 취해준 값을 exp로 원상 복귀시킨 후 class0(neg)에 가까운지 class1(pos)지 비교한다. Neg일 확률이 높으면 return 0 이고 그게 아니라면 return 1을 해준다.

결과

ratings_train.txt로 학습하고, ratings_valid.txt으로 검증해봤을 때

- 띄어쓰기:

```
C:₩Users₩rookiebox₩PycharmProjects₩review>python main.py
word_prob_table size :  414417
train_fin
answer_sheet num:  10000
calculating ....  0
calculating ....  1000
calculating ....  2000

calculating ....  7000
calculating ....  8000
calculating ....  9000
0.8107
--- {}s seconds --- 9626.195023536682
```

나름 결과가 좋게 나왔다. 80%정도 높게 측정되었다.
이제 개선해보자.

개선해보기

- Smoothing
- Log, exp
- 검증 데이터 문서에 학습데이터에 없는 단어가 있을 경우 해당 단어가 나오지 않을 확률을 더해준다.
- 구두점, 조사 등 범주 분류에 불필요하다고 판단되는 단어들은 따로 처리해서 뺀다.

시도 1

```
def tokenize(message):
    t = Twitter()
    all_words = t.nouns(message)
    return set(all_words)
```

- KoNLPy를 이용해서 Tokenize해봤다. Nouns() 함수는 명사만 추출하는 함수이다.

```
453109  저예산으로 엄청난 흥행을 거두었다는것으로 이유는 충분하다1
```

문자열을

```
{'예산', '이유', '흥행'}
```

- 이런 식으로 tokenize 한다. 명사를 생각보다 잘 뽑아 내지를 못하는 것 같다. 역시 한글은 정말 대단한 언어이다.

```
(C:₩Users₩rookiebox₩AppData₩Local₩conda₩conda₩envs₩py35) C:₩Users₩rookiebox₩PycharmProjects₩review>python main.py
C:₩Users₩rookiebox₩AppData₩Local₩conda₩conda₩envs₩py35₩lib₩site-packages₩konlpy₩tag₩_okt.py:16: UserWarning: "Twitte
r" has changed to "Okt" since KoNLPy v0.4.5.
  warn('"Twitter" has changed to "Okt" since KoNLPy v0.4.5.')
word_prob_table size :  42084
train_fin
answer_sheet num:  10000
calculating ....  0
calculating ....  4000
calculating ....  5000
calculating ....  6000
calculating ....  7000
calculating ....  8000
calculating ....  9000
0.7737
--- {}s seconds --- 922.3886814117432
```

- 명사로 뽑아서 했을 때 성능이 오히려 더 떨어졌다. 역시 명사로 평가하는 것은 한계가 있는 것 같다.

시도 2

```
def tokenize(message):
    t = Twitter()
    all_words= t.pos(message,norm=True, stem=True)
    pprint(all_words)
    return set(all_words)
```

- 이번에는 pos함수를 사용해봤다. 옵션으로 normalization 와 stem이 있는데 이것들은 각각 정규화, 어근화를 해준다.

```
9100399 내가 이걸 보자고 토욜밤을 허무하게 보냈나..화가난다ㅜㅜ  0
```
이문장을

```
[('내', 'Noun'),
 ('가', 'Josa'),
 ('이', 'Determiner'),
 ('걸', 'Noun'),
 ('보다', 'Verb'),
 ('토욜밤', 'Noun'),
 ('을', 'Josa'),
 ('허무하다', 'Adjective'),
 ('보내다', 'Verb'),
 ('..', 'Punctuation'),
 ('화가', 'Noun'),
 ('나다', 'Verb'),
 ('ㅜㅜ', 'KoreanParticle')]
[('모', 'Noun'), ('냐', 'Josa'), ('이', 'Noun'), ('게', 'Josa')]
```
이렇게 뽑아준다.

- 여기서 정말 놀라웠던 것은 '보자고'가 '보다' 로 인식되고 '허무하게'가 '허무하다' 로 인식됐다는 점이다. 이렇게 하면 정말 정확도가 굉장히 상승할 것 같다. 'ㅜㅜ'도 koreanParicle 으로 센스있게 처리하는 모습도 볼 수 있다.
- 이 함수의 return 값이 set이기 때문에 count[word] 에서의 word에 set의 원소인 ('시간', 'Noun')가 그대로 들어가게 된다. 사실은 train과 valid모두 set으로 들어간다는 가정하에 이 부분은 그대로 두어도 된다. 그리고 set으로 return해주기 때문에 한 문장 안에 중복되는 단어들도 없어진다.

결과는

```
(C:₩Users₩rookiebox₩AppData₩Local₩conda₩conda₩envs₩py35) C:₩Users₩rookiebox₩PycharmProjects₩review>python main2.py
C:₩Users₩rookiebox₩AppData₩Local₩conda₩conda₩envs₩py35₩lib₩site-packages₩konlpy₩tag₩_okt.py:16: UserWarning: "Twitte
r" has changed to "Okt" since KoNLPy v0.4.5.
  warn('"Twitter" has changed to "Okt" since KoNLPy v0.4.5.')
word_prob_table size :  54412
train_fin
answer_sheet num:  10000
calculating ....  0
calculating ....  6000
calculating ....  7000
calculating ....  8000
calculating ....  9000
0.8357
--- {}s seconds --- 1340.4375455379486
```

- 오 엄청나진 않지만 정말 조금 상승됐다. 그래도 이게 어딜까라는 생각이 든다.
  `('..', 'Punctuation'),` 이런 필요 없는 부분을 빼보면 어떨까 생각이 들어서 다시 코드를 고쳐보았다.

시도3

여기서 추가적으로 관사, 전치사 등 범주 분류 판단에 불필요한 항목을 제거해보았다.

https://docs.google.com/spreadsheets/d/1OGAjUvalBuX-oZvZ_-9tEfYD2gQe7hTGsgUpiiBSXI8/edit#gid=0

이 자료를 살펴보면, Twitter Korean Text(ntag=19)부분을 참고했다.

Noun	명사
Verb	동사
Adjective	형용사
Determiner	관형사 (ex: 새, 헌, 참, 첫, 이, 그, 저)
Adverb	부사 (ex: 잘, 매우, 빨리, 반드시, 과연)
~~Conjunction~~	~~접속사~~
Exclamation	감탄사 (ex: 헐, 어머나, 얼씨구)
~~Josa~~	~~조사 (ex: 의, 에, 에서)~~

PreEomi	선어말어미 (ex: 었)
Eomi	어미 (ex: 다, 요, 여, 하댘ㅋㅋ)
Suffix	접미사
~~Punctuation~~	~~구두점~~
Foreign	외국어, 한자 및 기타 기호
Alpha	알파벳
Number	숫자
Unkown	미등록어
KoreanParticle	(ex:ㅋㅋㅋ)
…	

과연 이렇게 빼면 성능이 좋아질까 의심이 가지만 우선, 제일 필요 없어 보이는 **구두점**과 **접속사**, **조사**를 삭제해보았다.

Tokenize(review)의 return 값은 set 형식이다.

`7312367 ...쓰레기... 0` 문장은

`{('...', 'Punctuation'), ('쓰레기', 'Noun')}` 이렇게 분류된다.

나는 Punctuation에 해당하는 '...'을 없애고 싶다. 그래서

```
    def count_words(self,out_list):
        counts = defaultdict(lambda: [0, 0])  # [neg , pos] checking
        for review, tag in out_list:
            if self.isNumber(review) is False:
                words = tokenize(review)
                #pprint(words)
                for word in words:
                    if word[1] in ['Punctuation','Conjunction','Josa']:
                        #print(word[0])
                        continue
                    counts[word][int(tag)] += 1
        return counts
```

이런 식으로 추가해줬다. 생각보다 구두점의 종류가 많았다.

우선 train과정에서 불필요한 어휘를 제거했다. (아직, valid의 구두점을 제거하지 않았다. Valid의 단어 중 우도 테이블에서 찾을 수 없으면 해당 단어가 나오지 않을 확률을 더해줌으로 확률은 구할 수 있다.)

문제가 있다면 정확한 negnum과 posnum을 측정해주지 못했다. '.....' 0 이런 경우 negnum에 -1을 해줘야 하는데 이런 케이스가 별로 없다고 생각하기에 따로 계산해주지 않았다.

```
.
!!!
...
;;
...
..
...
,
```

```
(C:₩Users₩rookiebox₩AppData₩Local₩conda₩conda₩envs₩py35) C:₩Users₩rookiebox₩PycharmProjects₩review>python main.py
C:₩Users₩rookiebox₩AppData₩Local₩conda₩conda₩envs₩py35₩lib₩site-packages₩konlpy₩tag₩_okt.py:16: UserWarning: "Twitte
r" has changed to "Okt" since KoNLPy v0.4.5.
  warn('"Twitter" has changed to "Okt" since KoNLPy v0.4.5.')
word_prob_table size :  52084
train_fin
answer_sheet num:  10000
calculating ....  0
calculating ....  5000
calculating ....  6000
calculating ....  7000
calculating ....  8000
calculating ....  9000
0.8362
--- {}s seconds --- 1509.358871936798
```

- 쓸데없는 부분이 많을 거라 생각하고 제거했는데 다행히 예상 맞았던 것 같다. (적어도 떨어지지는 않았다.) train과정에 for이 추가되는 바람에 train하는 속도가 굉장히 느려졌다.

시도 4

- 이번에는 train뿐만 아니라 valid 값도 불필요한 부분을 제거해주었다.

```
    def class0_probability(self, word_prob_table, query_review):
        log_prob_if_class0 = log_prob_if_class1 = 0.0
        query_words = tokenize(query_review)
        query_words2 = set()
        #pprint(query_words)
        #print(type(query_words))
        for d in query_words:
            if d[1] in ['Punctuation', 'Conjunction', 'Josa']:
                continue
            else:
                query_words2.add((d[0], d[1]))
        print(query_words2)
```

- tokenize의 return 값으로 set이 들어온다. for문을 돌면서 불필요한 항목을 포함하고 있는지 체

크해서 맞으면 continue 필요한 항목이면 새로운 query_word2 set에 추가해주었다.

결과

```
C:₩Users₩rookiebox₩AppData₩Local₩conda₩conda₩envs₩py35₩lib₩site-packages₩konlpy₩tag₩_okt.py:16: UserWarning: "Twitte
r" has changed to "Okt" since KoNLPy v0.4.5.
  warn('"Twitter" has changed to "Okt" since KoNLPy v0.4.5.')
word_prob_table size :  52084
train_fin
answer_sheet num:  10000
calculating ....  0
calculating ....  1000
calculating ....  6000
calculating ....  7000
calculating ....  8000
calculating ....  9000
0.8362
--- {}s seconds --- 1312.1964905261993
```

- 슬프게도 성능의 개선이 되지 않았다. 애초에 불필요한 요소의 수가 작아서 전체 확률에 크게 영향을 못 줘서 그런가 왜 확률이 이리도 똑같이 나왔는지는 모르겠다. train부분을 고쳤을 때 성능이 좋아지고 valid부분을 고쳤을 때는 train과 똑 같은 확률이 나왔으므로 이제 train의 Foreign, Unkown, Eomi 을 추가해봤다. 이건 실험적이다. 결과를 쉽게 예측할 수 없을 것 같다. 우선 어미를 빼고 나머지도 빼봐야겠다.

시도 5

- 우선 어미를 빼보았다. 우도 테이블의 개수가 감소했다.
- 생각해보니 우도 테이블의 크기가 감소하는데 성능이 올라가는걸 보면 이것은 엄청난 것 같다. 띄어쓰기 경우 테이블 크기가 압도적으로 많지만 성능은 80에서 머물고 tokenize을 잘 할수록 성능이 상승하는 모습을 볼 수 있었다.

```
(C:₩Users₩rookiebox₩AppData₩Local₩conda₩conda₩envs₩py35) C:₩Users₩rookiebox₩PycharmProjects₩review>python main.py
C:₩Users₩rookiebox₩AppData₩Local₩conda₩conda₩envs₩py35₩lib₩site-packages₩konlpy₩tag₩_okt.py:16: UserWarning: "Twitte
r" has changed to "Okt" since KoNLPy v0.4.5.
  warn('"Twitter" has changed to "Okt" since KoNLPy v0.4.5.')
word_prob_table size :  52011
train_fin
calculating ....  6000
calculating ....  7000
calculating ....  8000
calculating ....  9000
0.8362
--- {}s seconds --- 1218.2846035957336
```

더 이상 요소를 빼서 확률의 상승을 기대하기는 어려울 것 같다. 여기서 마치겠다.
하지만 우도 테이블의 크기를 감소시키고(시간도 감소됨) 성능을 유지했으므로 이는 의미가 있는

행위라고 생각된다.

시도	우도 테이블 크기	시간	성능
띄어쓰기	414417	9626	0.8107
1차 시도	42084	922	0.7737
2차 시도	54412	1340	0.8357
3차 시도	52084	1509	0.8362
4차 시도	52084	1312	0.8362
5차 시도	52011	1218	0.8362

Test 해보기

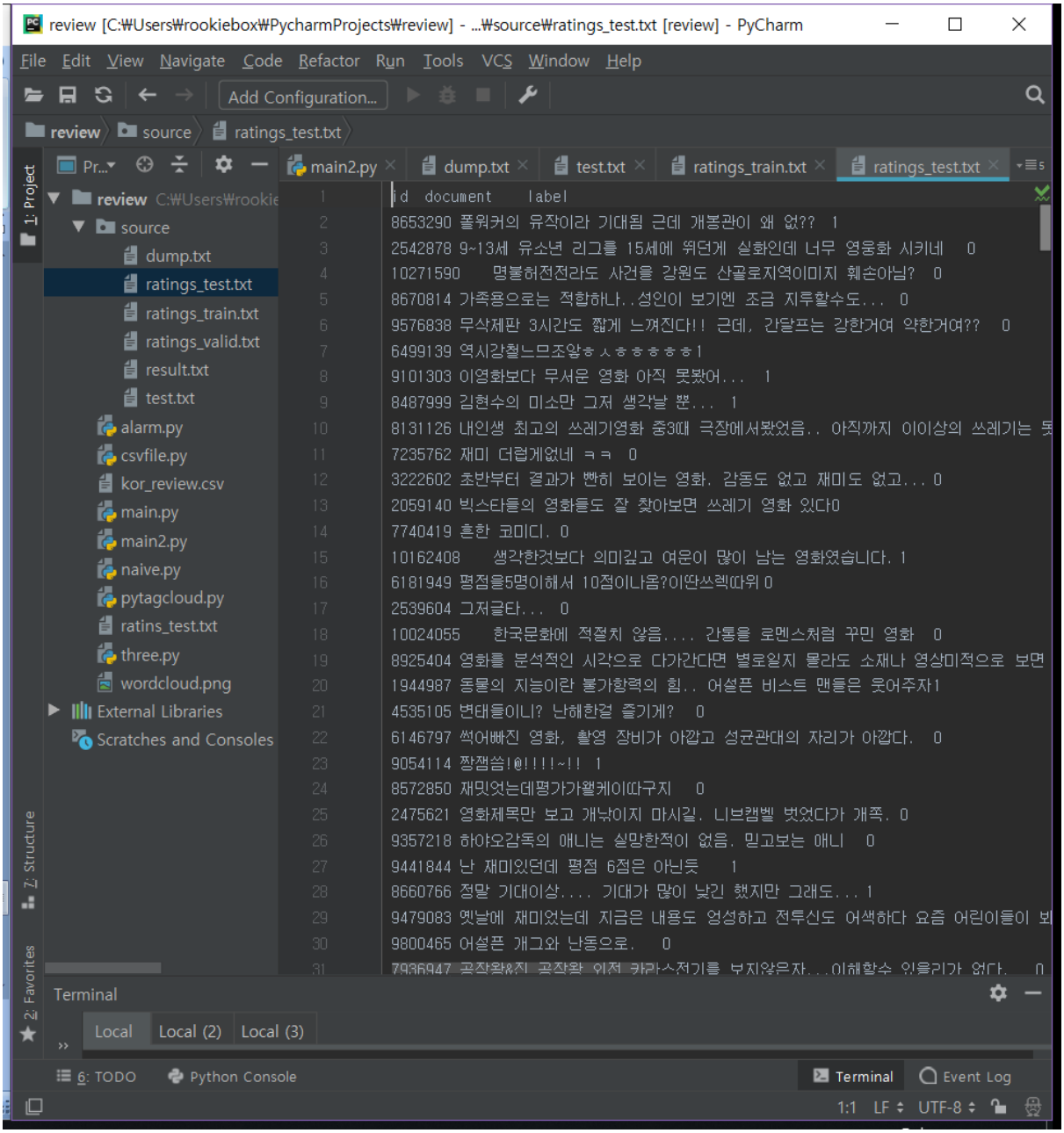
Train이 완료된 우도 함수 테이블을 이용해서, test을 해보았다. result함수는 scoring함수와 유사했다. classify함수를 이용해서 예측 tag값을 구하고 그 값을 append해주면 된다.

```
    def result(self,where):
        f = codecs.open("./source/" + where + ".txt", 'r', "utf-8")

        line = f.readline()
        line2 = f.readline()
        list2 = []
        while line2:
            line2 = line2.replace("₩r₩n", '')
            result2 = line2.split("₩t")
            #pprint(result2)
            list2.append([result2[0], result2[1]])
            line2 = f.readline()
        f.close()

        # pprint(list1)
        f2 = codecs.open("./source/" + where + ".txt", 'w+', "utf-8")
        string3 = 'id   document    label₩n'
        f2.write(string3)
```

```
        print('documnet writing...')
        counting2 =0
        for ans in list2:
            if counting2 % 1000 == 0:
                print("calculating .... ", counting2)
            prediction = self.classify(ans[1])
            ans.append(str(prediction))
            #print(ans)
            string2 = ans[0]+'₩t'+ans[1]+'₩t'+ans[2]+'₩n'
            f2.write(string2)
            counting2 += 1
        f2.close()
```

```
C:₩Users₩rookiebox₩AppData₩Local₩conda₩conda₩envs₩py35₩lib₩site-packages₩konlpy₩tag₩_okt.py:16: UserWarning: "Twitte
r" has changed to "Okt" since KoNLPy v0.4.5.
  warn('"Twitter" has changed to "Okt" since KoNLPy v0.4.5.')
word_prob_table size :  52084
train_fin
documnet writing...
calculating ....  0
calculating ....  5000
calculating ....  6000
calculating ....  7000
calculating ....  8000
calculating ....  9000
--- {}s seconds --- 2106.465080976486
```

결과로 확인할 수 있는 ratings_test:




```
3222602 초반부터 결과가 빤히 보이는 영화. 감동도 없고 재미도 없고... 0
2059140 빅스타들의 영화들도 잘 찾아보면 쓰레기 영화 있다0
7740419 흔한 코미디. 0
10162408    생각한것보다 의미깊고 여운이 많이 남는 영화였습니다. 1
```

비교적 잘 나온 것 같아 기쁘다.

```
9357218 하야오감독의 애니는 실망한적이 없음. 믿고보는 애니  0
```

이런 부분은 실망이라는 단어 때문에 부정으로 측정된 것 같다. 100%로 정확하진 않지만 어느 정도 성능을 보이고 있다는 점에서 굉장한 알고리즘이라고 생각된다.