감성 분석

앞에서 공부한 나이브 베이즈 분류 모형을 이용하여 문서에 대한 감성 분석(sentiment analysis)를 해보자. 감성 분석이란 문서에 대해 좋다(positive) 혹은 나쁘다(negative)는 평가를 내리는 것을 말한다.

샘플 데이터로는 github에 올려져 있는 네이버 영화 감상평에 대한 감성 분석 예제를 이용한다.

- https://github.com/e9t/nsmc (https://github.com/e9t/nsmc)

데이터 전처리

우선 데이터를 다운로드 받아서 읽어보자.

In [1]:

```
%%time
!rm -f ratings_train.txt ratings_test.txt
!wget -nc https://raw.githubusercontent.com/e9t/nsmc/master/ratings_train.txt
!wget -nc https://raw.githubusercontent.com/e9t/nsmc/master/ratings_test.txt
```

```
--2018-12-06 20:07:59--  https://raw.githubusercontent.com/e9t/nsmc/master/ratings_t
rain.txt (https://raw.githubusercontent.com/e9t/nsmc/master/ratings_train.txt)
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.72.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.72.133|:
443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 14628807 (14M) [text/plain]
Saving to: ‘ratings_train.txt’

ratings_train.txt   100%[===================>]  13.95M  7.27MB/s    in 1.9s

2018-12-06 20:08:02 (7.27 MB/s) - ‘ratings_train.txt’ saved [14628807/14628807]

--2018-12-06 20:08:02--  https://raw.githubusercontent.com/e9t/nsmc/master/ratings_t
est.txt (https://raw.githubusercontent.com/e9t/nsmc/master/ratings_test.txt)
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.72.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.72.133|:
443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4893335 (4.7M) [text/plain]
Saving to: ‘ratings_test.txt’

ratings_test.txt    100%[===================>]   4.67M  5.31MB/s    in 0.9s

2018-12-06 20:08:03 (5.31 MB/s) - ‘ratings_test.txt’ saved [4893335/4893335]

CPU times: user 114 ms, sys: 52.2 ms, total: 166 ms
Wall time: 4.11 s
```

여기에서는 유니코드로 인코딩하며 읽기 위해 codecs 패키지를 사용한다. 읽어들인 결과는 유니코드 문자열이 된다.

In [2]:

```
import codecs
with codecs.open("ratings_train.txt", encoding='utf-8') as f:
    data = [line.split('₩t') for line in f.read().splitlines()]
    data = data[1:]   # header 제외
```

이 데이터는 번호, 내용, 평점으로 이루져 있으므로 내용을 X, 평점을 y로 저장한다.

In [3]:

```
from pprint import pprint
pprint(data[0])
```

```
['9976970', '아 더빙.. 진짜 짜증나네요 목소리', '0']
```

In [4]:

```
X = list(zip(*data))[1]
y = np.array(list(zip(*data))[2], dtype=int)
```

이제 이 데이터를 다항 나이브 베이즈 모형으로 학습시킨다.

In [5]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report

model1 = Pipeline([
    ('vect', CountVectorizer()),
    ('mb', MultinomialNB()),
])
```

In [6]:

```
%%time
model1.fit(X, y)
```

```
CPU times: user 2.03 s, sys: 88.5 ms, total: 2.12 s
Wall time: 2.13 s
```

Out[6]:

```
Pipeline(memory=None,
     steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='st
rict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)₩₩b₩₩w₩₩w+₩₩b',
        tokenizer=None, vocabulary=None)), ('mb', MultinomialNB(alpha=1.0, class_pri
or=None, fit_prior=True))])
```

모형의 성능을 보기 위해 테스트 데이터도 읽어들인다.

In [7]:

```
import codecs
with codecs.open("ratings_test.txt", encoding='utf-8') as f:
    data_test = [line.split('₩t') for line in f.read().splitlines()]
    data_test = data_test[1:]   # header 제외
```

In [8]:

```
X_test = list(zip(*data_test))[1]
y_test = np.array(list(zip(*data_test))[2], dtype=int)

print(classification_report(y_test, model1.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.81      0.84      0.83     24827
           1       0.84      0.81      0.82     25173

   micro avg       0.83      0.83      0.83     50000
   macro avg       0.83      0.83      0.83     50000
weighted avg       0.83      0.83      0.83     50000
```

이 결과를 Tfidf 방법을 사용했을 때와 비교해 보자.

In [9]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

model2 = Pipeline([
    ('vect', TfidfVectorizer()),
    ('mb', MultinomialNB()),
])
```

In [10]:

```
%%time
model2.fit(X, y)
```

```
CPU times: user 2.08 s, sys: 82.4 ms, total: 2.16 s
Wall time: 2.16 s
```

Out[10]:

```
Pipeline(memory=None,
     steps=[('vect', TfidfVectorizer(analyzer='word', binary=False, decode_error='st
rict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,
...True,
        vocabulary=None)), ('mb', MultinomialNB(alpha=1.0, class_prior=None, fit_pri
or=True))])
```

In [11]:

```
print(classification_report(y_test, model2.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.81      0.84      0.83     24827
           1       0.84      0.81      0.83     25173

   micro avg       0.83      0.83      0.83     50000
   macro avg       0.83      0.83      0.83     50000
weighted avg       0.83      0.83      0.83     50000
```

이번에는 형태소 분석기를 사용한 결과와 비교한다.

In [12]:

```
from konlpy.tag import Okt
pos_tagger = Okt()

def tokenize_pos(doc):
    return ['/'.join(t) for t in pos_tagger.pos(doc)]
```

In [13]:

```
model3 = Pipeline([
    ('vect', CountVectorizer(tokenizer=tokenize_pos)),
    ('mb', MultinomialNB()),
])
```

In [14]:

```
%%time
model3.fit(X, y)
```

```
CPU times: user 3min 52s, sys: 1.04 s, total: 3min 53s
Wall time: 3min 39s
```

Out[14]:

```
Pipeline(memory=None,
     steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='st
rict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=<function tokenize_pos at 0x11bbc9d90>, vocabulary=None)), ('mb',
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True))])
```

In [15]:

```
print(classification_report(y_test, model3.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.86      0.85     24827
           1       0.86      0.85      0.85     25173

   micro avg       0.85      0.85      0.85     50000
   macro avg       0.85      0.85      0.85     50000
weighted avg       0.85      0.85      0.85     50000
```

(1,2)-gram 을 사용하면 성능이 더 개선되는 것을 볼 수 있다.

In [16]:

```
model4 = Pipeline([
    ('vect', TfidfVectorizer(tokenizer=tokenize_pos, ngram_range=(1, 2))),
    ('mb', MultinomialNB()),
])
```

In [17]:

```
%%time
model4.fit(X, y)
```

```
CPU times: user 3min 44s, sys: 873 ms, total: 3min 45s
Wall time: 3min 43s
```

Out[17]:

```
Pipeline(memory=None,
     steps=[('vect', TfidfVectorizer(analyzer='word', binary=False, decode_error='st
rict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 2), norm='l2', preprocessor=None, smooth_idf=True,
...True,
        vocabulary=None)), ('mb', MultinomialNB(alpha=1.0, class_prior=None, fit_pri
or=True))])
```

In [18]:

```
print(classification_report(y_test, model4.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.86      0.87      0.87     24827
           1       0.87      0.86      0.87     25173

   micro avg       0.87      0.87      0.87     50000
   macro avg       0.87      0.87      0.87     50000
weighted avg       0.87      0.87      0.87     50000
```