13. 감성분석(Sentiment Analysis)

인터넷의 메시지나 e-mail 등은 현대사회에서 가장 유용하고 빈번하게 사용되는 문자로 표현된 문서들이다. 자연어 분석(Natural language processing)의 한 분야인 감성분석은 이러한 문서로부터 문서 작성자의 의도나 의견을 머신러닝을 통해 분류하는 러닝기법이다. 문서는 단어로 구성되어 있고 문자뿐만 아니라 숫자, 이모티콘, HTML 표식 등의 수많은 조합으로 이루어져 있다. 머신러닝은 기본적으로 수많은 문서를 통해 학습하므로 문서(documents)가 표본이 되고 문서를 구성하는 단어가 특성변수(feature variable)가 된다. 즉, n개의 문서가 있을 때 이 n개의 문서에 포함된 모든 단어는 특성변수가 되므로 문서의 특성변수의 차원은 근본적으로 클 수밖에 없다. 당연하게 감성분석을 포함한 자연어 분석에서는 개인용 컴퓨터 저장용량으로 처리할 수 없을 정도의 크기를 가진 자료(big data)를 다루게 된다.

13.1 감성분석(Sentiment Analysis)

자연어분석의 핵심은 문서를 어떻게 특성변수로 변환할지이다. 쉽게 이해하기 위해 다음의 간단한 3개의 문서를 고려해보자.

“The building is mine”
“The car is running”
“The building is mine, the car is running, and I am richier”

자연어를 분석하기 위해서는 문장을 구성하는 문자(character)나 단어(word)를 수량화가 가능하도록 분리해야 한다. 대부분의 자연어 분석에서는 단어를 수량화하므로 단어를 중심으로 설명하도록 한다. 주어진 세 개의 문서를 모두 소문자로 만든 후, ‘the’, ‘building’, ‘is’, ‘mine’, ‘the’, ‘car’, ..., ‘am’, ‘rich’로 분리한다. 유일한 단어들만 모으면 {‘am’, ‘and’, ‘building’, ‘car’, ‘is’, ‘mine’, ‘richier’, ‘running’, ‘the’}이 된다. 이를 1-gram bag-of-word라고 하며 bag을 쓴 이유는 집합이라는 의미를 강조하기 위함이다. 일반적으로 자연어의 이해나 번역을 위해서 단어의 순서가 매우 중요하지만 여기에서는 순서 정보를 완전히 무시한다는 의미에서 bag을 사용하고 있다. 자연어의 순서를 이용한 분석은 딥러닝에서 주로 다룬다. 단어의 분리를 ‘the’, ‘the building’, ‘building’, ‘building is’, ‘is’, ‘is mine’,‘mine’, ‘the’, ‘the car’, ‘car’, ‘car is’, ..., ‘am’, ‘am rich’, ‘rich’로 분리한 후, 유일한 단어분리의

집합을 2-gram bag-of-word라고 한다. 그러므로 3-gram bag-of-word는 'the', 'the building', 'building','building is', 'the building is', 'is', 'is mine', 'mine', ...으로 단어를 분리한 후 유일한 단어분리의 집합으로 정의된다. 이러한 n-gram bag-of-word를 토큰화(tokenization)라고 한다. 3-gram 또는 4-gram bag-of-word가 스팸 메일을 걸러내는(spam-mail filtering)데에 효과적이라고 알려져 있다. 그러므로 n-gram에서 n은 교차검증(cross-validation)을 통해 결정해야 할 초모수(hyperparameter)이다. 다음은 토큰화한 자료에 index를 부여하는 과정이다. 논의를 간단하게 하기 위해 1-gram bag-of-word={'am', 'and', 'building', 'car', 'is', 'mine', 'richier', 'running', 'the'}를 통해 살펴보자 'am'에 index 1을 'and'에 index 2를, 계속하여 차례대로 index를 부여하고 마지막으로 'the'에 index 9를 부여한다. 이제 각 문서별로 각 특성변수의 빈도로 구성된 array로 만든다. 즉

$$\begin{matrix}[0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1]\\ [0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1]\\ [1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 2]\end{matrix} \tag{13.1}$$

예를 들어, 맨 마지막 열은 'the'의 빈도수를 나타내는 수로, 첫 번째 문서에서는 한 번, 두 번째 문서에서는 한번, 그리고 세 번째 문서에서는 두 번 나타난 것을 보여주고 있다. 이와 같은 특성변수의 값을 raw term frequency라고 하며 $tf(t,d)$라고 표기한다. 즉 문서 d에서 단어 t가 나타난 횟수를 말한다. 분석 시 문서 각각에 공통적으로 자주 나타나는 단어(예를 들어 (13.1) 5번째 열의 'is', 마지막 열의 'the')들이 있다. 이러한 단어들은 문서의 특성을 파악하는데 필요한 정보의 질이 높지 않을 경우가 많기 때문에 가중치를 낮출 필요가 있다. 다음의 역문서빈도(inverse document frequency)를 (13.1)의 $tf(t.d)$에 곱하여 단어의 가중치를 조정한다.

$$idf(t,d) = \log\frac{n_d}{df(d,t)}$$

여기에서 n_d는 문서의 총수이고, $df(d,t)$는 단어 t를 포함하고 있는 문서의 수이다. 최종적으로, term frequency-inverse document frequency, 즉

$$tf\text{-}idf(t,d) = tf(t,d)\times(idf(t,d)+1) \tag{13.2}$$

를 계산한다. (13.2)에서 $idf(t,f)+1$을 한 이유는 $df(d,t)=n_d$이면 $idf(t,d)$=0이 되기 때문이다. (13.2)의 $idf(t,d)$ 대신

$$\log\frac{1+n_d}{1+df(d,t)} \tag{13.3}$$

로 수정하여 사용하기도 하며 (13.3)의 $idf(t,d)$를 smooth idf라고 한다. 세 번째 자료에서 'is'의 $tf = idf('is', d3)$를 구해보면, 문서의 총수 n_d=3이고 모든 세 개의 문서가 'is'를 포함하고 있으므로 $2\times(\log\frac{1+3}{1+3}+1)=2$가 된다. 세 번째 문서에 대해 다른 단어에 대해서도 구해보면 [1.69 1.69 1.29 2 1.29 1.69 1.29 2]이다.

이를 L_2 노름(norm)정규화(normalization)하면

$$\frac{[1.69\ 1.69\ 1.29\ 1.29\ 2\ 1.29\ 1.69\ 1.29\ 2]}{\sqrt{1.69^2+1.6^2+1.29^2+1.29^2+2^2+1.29^2+1.69^2+1.29^2+2^2}}$$

$$=[0.35\ 0.35\ 0.27\ 0.27\ 0.41\ 0.27\ 0.35\ 0.27\ 0.41]$$

이 된다(참고로 L_1노름 정규화는 $\sum_{i=1}^{d}|x_i|$를 말한다). 이러한 L_2 노름정규화는 첫 번째, 두 번째 문서에서도 동일하게 계산되며 결과적으로 (13.1)의 $tf(t,d)$는 $td\text{-}idf$로 9개의 특성변수의 값은

$$\begin{matrix}[0 & 0 & 0.56 & 0 & 0.43 & 0.56 & 0 & 0 & 0.43]\\ [0 & 0 & 0 & 0.56 & 0.43 & 0 & 0 & 0.56 & 0.43]\\ [0.35 & 0.35 & 0.29 & 0.27 & 0.41 & 0.27 & 0.35 & 0.27 & 0.41]\end{matrix} \tag{13.4}$$

로 전환된다. 감성분석은 (13.1)에 정의된 $tf(t,d)$를 L_1 또는 L_2 노름 정규화한 특성변수를 이용하거나 (13.2)에 정의된 $tf\text{-}idf$를 (13.4)와 같이 L_1 또는 L_2 노름 정규화한 특성변수를 이용한다.

13.2 파이썬을 이용한 사례 분석

이 절에서는 일부 프로그램과 해석이 추가되었으나 근본적으로 Raschka & Mirjalili (2017)의 프로그램을 큰 수정 없이 옮겨 놓은 것임을 밝혀둔다. 감성분석을 위해 50,000개의 영화 평론을 담은 internet movie database (IMDb)를 "http://ai.stamford.eud/~amaas/data/sentiment"로부터 내려받은 후 다음의 파이썬 프로그램을 수행한다.

```
import tarfile
tar = tarfile.open('C://Users//Desktop//aclImdb_v1.tar.gz','r:gz')
tar.extractall()
```

여기에서 path는 download 받은 aclImdb_v1.tar가 저장된 경로를 말한다. IMDb는 매우 큰 자료이므로 이 과정은 20~30분 정도가 소요된다. 다음은 자료의 download와 자료를 행렬 형태로 바꾸기 위한 과정이다.

```
import pyprind
import pandas as pd
import os
basepath = 'C://Users//Desktop//aclImdb'
labels = {'pos':1, 'neg':0}
pbar = pyprind.ProgBar(50000)
df = pd.DataFrame()
for s in ('test', 'train'):
    for l in ('pos', 'neg'):
        path = os.path.join(basepath, s, l)
        for file in os.listdir(path):
            with open(os.path.join(path, file), 'r', encoding='utf-8') as infile:
                txt = infile.read()
            df = df.append([[txt, labels[l]]], ignore_index=True)
            pbar.update()
df.colums = ['review', 'sentiment']
```

위 프로그램도 10여분 정도의 시간이 소요되며 행렬 형태로 정리된 자료를 df로 정의하였다. df의 행의 수는 50,000, 열의 수는 2개로 구성되어 있고 첫 번째 열은 영화에 대한 평론으로 다음 코딩결과로 볼 수 있듯이 상당히 긴 글로 구성되어 있다. 열의 두 번째 변수명은 review로 영화에 대한 평가 관련 변수이며 긍정(positive), 부정(negative)(긍정은 1 부정은 0)으로 범주화 되어있다. 또한 자료가 50,000개이므로 pbar를 50,000으로 부여했다.

```
import numpy as np
np.random.seed(0)
df = df.reindex(np.random.permutation(df.index))
df.to_csv('C://Users//Desktop//movie_data.csv', index = False, encoding='utf-8')
```

위 프로그램은 IMDb가 ‘pos’와 ‘neg’ 순으로 정리가 돼 있으므로, random

permutation을 이용하여 자료의 순서를 임의로(random) 뒤섞을 필요가 있다. 실제 분석에서 이러한 사전정보가 없을 경우에도 사전적으로 자료의 뒤섞음(shuffle)하는 것을 권장한다. df.to_csv는 자료를 csv형태로 저장하라는 명령어이며 path는 저장 경로이다.

```
df = pd.read_csv('C://Users//Desktop//movie_data.csv', encoding='utf-8')
df.head(3)
```

```
(out)
0  In 1974, the teenager Martha Moxley (Maggie Gr...  1
1  OK... so... I really like Kris Kristofferson a...  0
2  ***SPOILER*** Do not read this, if you think a...  0
```

앞의 세 개의 관측치를 출력하여 살펴보면 sentiment는 1(긍정)) 또는 0(부정)으로 구분되어 있고 review는 구체적인 영화 평론이 나열되어 있다.

문서를 bag-of-words 모형으로 전환하기 위해 필수적인 과정은, 불필요한 기호나 꺽쇠 또는 HTML표식, 그리고 하나의 알파벳으로 이루어진 단어가 아닌 특수문자 등, 분류나 회귀를 위한 정보를 가지고 있지 않은 것으로 판단되는 것을 사전에 정리를 하여야 한다. 따라서 다음의 코딩은 자연어 분석 시 가급적 실행할 것을 권장한다.

```
import re
def preprocessor(text):
    text = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:₩)|₩(|D|P)',
    text)
    text = (re.sub('[₩W]+', ' ', text.lower())
    +' '.join(emoticons).replace('-', ''))
    return text
```

위 프로그램에서 re.sub(pattern, rl, string)의 형태는 string에서 pattern과 일치하는 text를 찾아서 rl로 바꾸라는 명령어로 첫 번째 re.sub는 text에서 HTML표식과 일치하는 것은 모두 빈 공간으로 대체하라는 말이고 두 번째 re.sub에 있는 '[₩W]+'는 단어가 아닌 모든 기호는 모두 빈 공간으로 대체하고 emoticons은 이 빈공간 뒤에 배치하라는 명령어이다. text.lower()에 의해 text의 대문자는 모두 소문자로 전환하였다. 대문자가 정보를 갖지 않을 것으로 판단하였기 때문이다. 모든 emotion은 정보를 일반적으로 가지고 있으므로 emoticons에 저장하여 문서의 뒤에 정리하고 있

다.

```
df['review'] = df['review'].apply(preprocessor)
```

문서의 사전정리과정을 위와 같이 끝나면 문서의 단어를 분류하는 작업에 들어가게 된다.

```
def tokenizer(text):
    return text.split()
tokenizer('runners like running and thus they run')
```

```
(out) ['runners','like','running','and','thus','they','run']
```

ToKenizer 과정에서 유용한 기술 중의 하나는 변형해서 사용되는 단어의 원뿌리로 재 표현해주는 word stemming이 있다. 예를 들어 'runners'는 'runner'로, 'running'은 'run'으로 바꾸어 주는 과정을 말한다.

```
from nltk.stem.porter import PorterStemmer
porter = PorterStemmer()
def tokenizer_porter(text):
    return [porter.stem(word) for word in text.split()]
tokenizer_porter('runners like running and thus they run')
```

```
(out) ['runner', 'like', 'run', 'and', 'thu', 'they', 'run']
```

nltk는 Natural language Toolkit의 약자로 실행 결과 runners는 runner로, running은 run으로 thus는 thu로 재정의 되었음을 알 수 있다. nltk에는 또 다른 중요한 기능을 가지고 있다. 영어문장에서 너무나 자주 쓰이는, 예를 들어, 'and', 'is'등 정보가 거의 없는 것으로 판단되는 소위 stopwords가 내장되어 있는데 다음과 같은 code를 이용하여 stopword를 제거할 수 있다.

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop = stopwords.words('english')
[w for w in tokenizer_porter('a runner likes running and runs alot')[-10:] if w not
in stop]
```

```
(out) ['runner', 'like', 'run', 'run', 'alot']
```

이제 IMDb 자료를 이용하여 지금까지 설명한 ToKenizer, stopword등을 적용한 후

분류(즉, 영화에 대한 평가를 positive or negative로 분류)을 위한 머신러닝을 실시하여 보자. 자료의 크기가 크기 때문에 학습데이터와 시험데이터를 반씩으로 우선 나누고자 한다.

```
X_train = df.loc[:25000, 'review'].values
y_train = df.loc[:25000, 'sentiment'].values
X_test = df.loc[25000:, 'review'].values
y_test = df.loc[25000:, 'sentiment'].values

from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(strip_accents=None,lowercase=False,preprocessor=None)
param_grid = [{'vect__ngram_range': [(1,1)],'vect__stop_words': [stop,
                  None],'vect__tokenizer': [tokenizer,tokenizer_porter],'clf__penalty':
                  ['l1', 'l2'],'clf__C': [1.0, 10.0, 100.0]},
                 {'vect__ngram_range': [(1,1)],'vect__stop_words': [stop,
                  None],'vect__tokenizer': [tokenizer,
                  tokenizer_porter],'vect__use_idf':[False],
                  'vect__norm':[None], 'clf__penalty': ['l1', 'l2'],'clf__C': [1.0, 10.0,
                  100.0]}]
lr_tfidf = Pipeline([('vect', tfidf),('clf', LogisticRegression(random_state=0))])
gs_lr_tfidf = GridSearchCV(lr_tfidf, param_grid,scoring='accuracy',cv=5,
verbose=1,n_jobs=1)
gs_lr_tfidf.fit(X_train, y_train)
```

위 프로그램은 bag-of-words 모형을 적용하기 위해 TfidVectoriger 클래스를 호출하여 실행하고 있다. TfidVectoriger 클래스는 특성변수의 값을 정의하기 위해 smooth-idf를 디폴트로 사용한다. 또한 idf의 표준화로 L_2 노름을 디폴트로 한다. smooth-idf=false로 지정하면 smooth-idf를 사용하지 않고, norm='l1'으로 지정하면 L_1 노름으로 정규화하며 norm=false로 지정하면 정규화(normalization)을 하지 않는다. 분류기법으로 로지스틱회귀를 사용하고 bag-of-words 모형과 로지스틱회귀의 초모수를 5개의 부분집합으로 구성하고 5-분할 교차검증(cross-validation)을 이용하여 초모수를 선택하고 있다.

pipeline은 Tfidf(bag-of-words 모형)에 vect 라는 이름을 부여했고, 로지스틱회귀에는 clf 라는 이름을 부여하였다. 이 두 개의 이름을 이용하여 TfidVectoriger와

LogisticRegression의 다양한 옵션을 param-grid에 지정하였고 이 다양한 옵션 중에 최선의 조합을 GridsearchCV를 통해 찾을 수 있도록 하고 있다. param-grid에 지정된 다양한 옵션을 좀 더 살펴보도록 하자. 먼저 TfidVectorisger는 ngram-range=(1,1)으로 지정하여 1-gram 단어 분해(word split)를 하며 만약 2-gram단어 분해를 원하면 ngram-range=(2,2)로 지정하면 된다. stopwords=stop이면 stopwords를 이용하여 stopword를 제거하겠다는 뜻이고 stop-words=None으로 지정하면 stop-word를 제거하지 않는다는 뜻이다. ToKenizer=tokenizer 또는 tokenizer_porter를 사용하여 단어의 원래 뿌리로 전환할 것인지를 지정할 수 있다. ToKenizer는 callable이므로 자체적으로 def에 의해 정의한 ToKenizer와 ToKenizer_porter를 불러서(called)사용할 수 있다. use_idf=False로 지정하면 inverse document frequency를 사용하지 않고 오직 $tf(t.d)$만 사용한다는 의미이다. tfid에 'vect'라는 이름을 부여했으므로 이 모든 옵션 앞에 vect__ 를 덧붙여주어야 한다. LogistcRegression에 대한 옵션을 규제화(regularization)(penalty='l1' 또는 'l2')와 규제화의 강도를 조절하는 C가 부여되어 있다. GridsearchCV의 결과는

```
print('Best parameter set: %s ' % gs_lr_tfidf.best_params_)
```

```
(out) Best parameter set: {'clf__C': 10.0, 'vect__stop_words': None, 'clf__penalty': 'l2', 'vect__tokenizer': <function tokenizer at 0x7f6c704948c8>, 'vect__ngram_range': (1, 1)}
```

로 정리할 수 있다. 또한 이러한 결과를 도출한 교차검증의 검증데이터의 정밀도는 다음과 같다.

```
print('CV Accuracy: %.3f' % gs_lr_tfidf.best_score_)
```

```
(out) CV Accuracy: 0.892
```

으로, 최상의 초모수를 이용한 시험데이터에 적용한 로지스틱회귀 결과는 다음과 같다.

```
print('Test Accuracy: %.3f' % clf.score(X_test, y_test))
```

```
(out) Test Accuracy: 0.899
```