정보처리 및 자연언어처리

과제3 : NaiveBayesClassifier를 이용하여 accuracy 높은 모델 만들기

경제학/빅데이터사이언스 전공 20180590 이윤주

0.Data 정제

-entity_data_utf8.txt 파일의 정보를 전체/한국어만 어휘 별로 (어절, 개체명)만 뽑아 각각 labels, kor_word로 저장한다.

```
import nltk
import konlpy
import os
import re
from konlpy.tag import Kkma
os.getcwd()
os.listdir('/Users/iyunju/Documents/대학교/21-1학기/정보처리및자연언어처리/3차과제_자연어처리')
a=open('/'.join([os.getcwd(),'entity_data_utf8.txt']), encoding='utf8').read()

#(어휘, 개체명) for all 단어
sent=a.split('\n1\t')
for i in range(0,len(sent)):
    s=sent[i].split('\t')
    if i==0:
        for j in range(1, len(s)):
            if (j%2==0):
                ss=re.sub('[\n\d]','',s[j])
                tmp=s[j-1]
                s[j-1]=ss
                s[j-2]=tmp
        sent[0]=s[:-1]

    else:
        for j in range(1, len(s)):
            if (j%2!=0):
                ss=re.sub('[\n\d]','',s[j])
                s[j]=ss
        sent[i]=s

labels=[]
for i in range(0, len(sent)):
    t=sent[i]
    label=[(t[j], t[j+1]) for j in range(0, len(sent[i]),2)]
    labels.append(label)
labels[:3] #결과 확인용
```

```
[[('비토리오', 'PER'),
  ('양일', 'DAT'),
  ('만에', '-'),
  ('영사관', 'ORG'),
  ('감호', 'CVL'),
  ('용퇴,', '-'),
  ('항룡', '-'),
  ('압력설', '-'),
  ('의심만', '-'),
  ('가율', '-')],
 [('이', '-'),
  ('음경동맥의', '-'),
  ('직경이', '-'),
  ('8', 'NUM'),
  ('19mm입니다', 'NUM'),
  ('.', '-')],
 [('9세이브로', 'NUM'),
  ('구완', '-'),
  ('30위인', 'NUM'),
  ('LG', 'ORG'),
  ('박찬형은', 'PER'),
  ('평균자책점이', '-'),
  ('16.45로', 'NUM'),
  ('준수한', '-'),
  ('편이지만', '-'),
  ('22⅓이닝', 'NUM'),
  ('동안', '-'),
  ('피홈런이', '-'),
  ('31개나', 'NUM'),
  ('된다', '-'),
  ('.', '-')]]
```

```
#(어휘, 개체명) for 한국어만
f=re.sub('\n', '',a)
z=f.split('\t')
labeled=[] #문장으로 나뉘지 않은 연속된 (어휘, 개체)
for i in range(1,len(z)-1,2):
    k=re.findall('\D+', z[i+1])
    z[i+1]=k
    labeled.append((z[i],k))
labeled=[(labeled[i][0], labeled[i][1][0]) for i in range(0,len(labeled))]

kor_words=[] #한글 단어만(개체명 x)
for x in range(0,len(labels)):
    for i in range(0,len(labels[x])):
        if labels[x][i][0].isalpha() == True:
            if len(re.findall('[^가-힣]+',labels[x][i][0]))==0:
                kor_words.append(labels[x][i][0])

#(어휘, 개체명)으로 만들기 위한 작업
dictl=dict(labeled)
kor_word=[(word, dictl[word]) for word in kor_words]
kor_word[:3]
```

```
[('비토리오', 'PER'), ('양일', 'DAT'), ('만에', '-')]
```

-이 후 각 방법(ngram, 자소분리 등등)을 위한 feature expression을 한 후,

-훈련셋과 테스트셋으로 나누어 훈련셋으로 모델을 훈련시킨다.

-테스트셋으로 정확도를 평가한다.

1.ngram을 이용한 모델

```
#1.ngram을 통한 모델
size=int(len(labels)*0.9)
train_sents=labels[:size]
test_sents=labels[size:]
t1=nltk.UnigramTagger(train_sents)
t2=nltk.BigramTagger(train_sents, backoff=t1, cutoff=1)
t3=nltk.TrigramTagger(train_sents, backoff=t2)
t1.evaluate(test_sents), t2.evaluate(test_sents), t3.evaluate(test_sents)
```

```
(0.7338804768945704, 0.7368002395702709, 0.7365194931591458)
```

-라벨링 되어있는 배열을 90%는 훈련셋, 10%는 테스트셋으로 지정한다.

-유니그램, 바이그램, 트라이그램에 훈련셋을 넣어 학습시킨다.

-테스트셋을 넣고 정확도를 평가한다.

2.한글 자소 분리를 이용한 모델

```
#2.한글 자소 분리를 통한 모델
def jaso1(a):
    initial=['ㄱ','ㄲ','ㄴ','ㄷ','ㄸ','ㄹ','ㅁ',
             'ㅂ','ㅃ','ㅅ','ㅆ','ㅇ','ㅈ','ㅉ',
             'ㅊ','ㅋ','ㅌ','ㅍ','ㅎ']
    midial=['ㅏ','ㅐ','ㅑ','ㅒ','ㅓ','ㅔ','ㅕ','ㅖ',
            'ㅗ','ㅘ','ㅙ','ㅚ','ㅛ','ㅜ',
            'ㅝ','ㅞ','ㅟ','ㅠ','ㅡ','ㅢ','ㅣ']
    final=['','ㄱ','ㄲ','ㄱㅅ','ㄴ','ㄴㅈ','ㄴㅎ','ㄷ',
           'ㄹ','ㄹㄱ','ㄹㅁ','ㄹㅂ','ㄹㅅ','ㄹㅌ','ㄹㅍ','ㄹㅎ',
           'ㅁ','ㅂ','ㅂㅅ','ㅅ','ㅆ','ㅇ','ㅈ','ㅊ','ㅋ','ㅌ','ㅍ','ㅎ']
    chr_ord=ord(a)-44032
    ini_index=chr_ord//(21*28)
    mid_index=(chr_ord//28)%21
    final_index=chr_ord%28
    return [initial[ini_index], midial[mid_index],final[final_index]]

suffix=nltk.FreqDist()
for i in range(0, len(kor_words)):
    stems=jaso1(kor_words[i][-1])
    stem=stems[-1]
    if stem!='':
        suffix[stem]+=1
    else:
        suffix[stem]+=1

common_suffixes=[suf for (suf, count) in suffix.most_common(100)]
common_suffixes[:10]
```

```
['', 'ㄴ', 'ㄹ', 'ㅇ', 'ㄱ', 'ㅁ', 'ㅂ', 'ㅅ', 'ㅊ', 'ㅌ']
```

-한글 자음/모음을 분리하는 함수를 작성 후 kor_word에서 가장 많이 쓰이는 종성을 common_suffix에 저장한다.

```
def pos_features(word):
    features={}
    for suffix in common_suffixes:
        if suffix==jaso1(word[-1])[-1]:
            #features[f'endswith({suffix})']=dictl[word]
            features[f'종성({suffix})']=True
        else:
            features[f'종성({suffix})']=False
    return features

size=int(len(kor_word)*0.1)
featureset=[(pos_features(n),g) for (n,g) in kor_word]
featureset[:10]
```

```
    '종성(ㅅ)': False,
    '종성(ㅊ)': False,
    '종성(ㅌ)': False,
    '종성(ㅂㅅ)': False,
    '종성(ㅍ)': False,
    '종성(ㅈ)': False,
    '종성(ㄹㄱ)': False,
    '종성(ㅋ)': False,
    '종성(ㄱㅅ)': False,
    '종성(ㄹㅂ)': False,
    '종성(ㄲ)': False,
    '종성(ㄹㅁ)': False,
    '종성(ㅎ)': False,
    '종성(ㄷ)': False,
    '종성(ㅆ)': False,
    '종성(ㄹㅅ)': False},
   'PER'),
  ({'종성()': False,
    '종성(ㄴ)': False,
```

-feature extraction 진행

```
size=int(len(kor_word)*0.1)
featureset=[(pos_features(n),g) for (n,g) in kor_word]
train_set, test_set=featureset[size:], featureset[:size]
classifier=nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
```

```
0.746268842191597
```

```
classifier.show_most_informative_features(10)
```

```
Most Informative Features
                  종성(ㅅ) = True              NUM : DAT    =   1045.5 : 1.0
                 종성(ㄹㅂ) = True              NUM : -      =    479.8 : 1.0
                  종성(ㅋ) = True              TIM : -      =    389.5 : 1.0
                  종성(ㅈ) = True              TIM : -      =    159.6 : 1.0
                  종성(ㅎ) = True              TRM : -      =    152.2 : 1.0
                  종성(ㅍ) = True              PLT : ORG    =     61.2 : 1.0
                  종성(ㅂ) = True              EVT : DAT    =     53.7 : 1.0
                  종성(ㄱ) = True              FLD : DAT    =     40.4 : 1.0
                  종성(ㅁ) = True              MAT : FLD    =     20.8 : 1.0
                  종성(ㄷ) = True              DAT : -      =     20.4 : 1.0
```

-훈련셋 90%, 테스트셋 10%로 나누어 naivebayesclassifier로 훈련시킨 후 정확성을 체크한다.

3. 끝글자를 추출하여 비교

```
#3.단어별 마지막 글자를 통한 분석
def pos_features2(word):
    features={}
    suffix=word[-1]
    #features[f'endswith({suffix})']=dictl[word]
    features[f'마지막 문자']=word[-1]
    return features

size=int(len(kor_word)*0.1)
featureset=[(pos_features2(n),g) for (n,g) in kor_word]
featureset[:10]
```

```
[({'마지막 문자': '오'}, 'PER'),
 ({'마지막 문자': '일'}, 'DAT'),
 ({'마지막 문자': '에'}, '-'),
 ({'마지막 문자': '관'}, 'ORG'),
 ({'마지막 문자': '호'}, 'CVL'),
 ({'마지막 문자': '룡'}, 'NUM'),
 ({'마지막 문자': '설'}, '-'),
 ({'마지막 문자': '만'}, '-'),
 ({'마지막 문자': '율'}, '-'),
 ({'마지막 문자': '이'}, '-')]
```

```
size=int(len(kor_word)*0.1)
featureset2=[(pos_features2(n),g) for (n,g) in kor_word]
train_set2, test_set2=featureset2[size:], featureset2[:size]
classifier=nltk.NaiveBayesClassifier.train(train_set2)
nltk.classify.accuracy(classifier, test_set2)
```

```
0.7698052714642541
```

-마지막 글자를 feature로 삼아서 동일하게 진행.

3)-2. Feature 두 개를 합쳐서 시도

```
def pos_features(word):
    features={}
    for suffix in common_suffixes:
        if suffix==jaso1(word[-1])[-1]:
            #features[f'endswith({suffix})']=dictl[word]
            features[f'종성({suffix})']=True
        else:
            features[f'종성({suffix})']=False
    suffix=word[-1]
    #features[f'endswith({suffix})']=dictl[word]
    features[f'마지막 문자']=word[-1]

    return features

size=int(len(kor_word)*0.1)
featureset=[(pos_features(n),g) for (n,g) in kor_word]
featureset[:10]
```

```
  '종성(ㅈ)': False,
  '종성(ㄹㄱ)': False,
  '종성(ㅋ)': False,
  '종성(ㄱㅅ)': False,
  '종성(ㄹㅂ)': False,
  '종성(ㄲ)': False,
  '종성(ㄹㅁ)': False,
  '종성(ㅎ)': False,
  '종성(ㄷ)': False,
  '종성(ㅆ)': False,
  '종성(ㄹㅅ)': False,
  '마지막 문자': '오'},
 'PER'),
({'종성()': False,
  '종성(ㄴ)': False,
  '종성(ㄹ)': True,
  '종성(ㅇ)': False,
  '종성(ㄱ)': False,
  '종성(ㅁ)': False,
```

```
size=int(len(kor_word)*0.1)
featureset=[(pos_features(n),g) for (n,g) in kor_word]
train_set, test_set=featureset[size:], featureset[:size]
classifier=nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
```

```
0.7650209392203402
```

-2보다는 향상됐지만 3보다는 낮으므로 단일 feature가 낫다.

3)-3. 마지막 두 글자를 추출하여 비교

```
#3.단어별 마지막 글자를 통한 분석
def pos_features2(word):
    features={}
    features[f'마지막 문자']=word[-2:]
    return features

size=int(len(kor_word)*0.1)
featureset=[(pos_features2(n),g) for (n,g) in kor_word]
featureset[:10]
```

```
[({'마지막 문자': '리오'}, 'PER'),
 ({'마지막 문자': '양일'}, 'DAT'),
 ({'마지막 문자': '만에'}, '-'),
 ({'마지막 문자': '사관'}, 'ORG'),
 ({'마지막 문자': '감호'}, 'CVL'),
 ({'마지막 문자': '항룡'}, 'NUM'),
 ({'마지막 문자': '력설'}, '-'),
 ({'마지막 문자': '심만'}, '-'),
 ({'마지막 문자': '가율'}, '-'),
 ({'마지막 문자': '이'}, '-')]
```

```
size=int(len(kor_word)*0.1)
featureset2=[(pos_features2(n),g) for (n,g) in kor_word]
train_set2, test_set2=featureset2[size:], featureset2[:size]
classifier=nltk.NaiveBayesClassifier.train(train_set2)
nltk.classify.accuracy(classifier, test_set2)
```

```
0.8484671496563979
```

-가장 높은 정확도를 보인다.

4.(시도)문장 별 형태소 분석을 이용한 모델

```
kkm=[]
kkma=Kkma()
for i in range(0, 100):
    kkm.append(' '.join([labels[i][j][0] for j in range(0,len(labels[i]))]))
    sent=kkm[i]
    k=[kkma.pos(sent)[j][1] for j in range(0,len(kkma.pos(sent)))]

d=[]
for i in range(0,100):
    d.append(kkma.pos(kkm[i]))
    dictd=[dict(d[j]) for j in range(0,len(d))]

def kkma_features(word): #sent=kkm[i]. word=분해된 단어 하나
    features=[]
    if dictd[i][word] in common_suf:
        features[f'형태소({dictd[i][word]})']=True
    else:
        features[f'형태소({dictd[i][word]})']=False
    return features
```

```
dictd[:3]
```

```
[{'비': 'XPN',
  '토리': 'NNG',
  '오': 'NNG',
  '양일': 'NNG',
  '만': 'JX',
  '에': 'JKM',
  '영': 'NNG',
  '사관': 'NNG',
  '감호': 'NNG',
  '용퇴': 'NNG',
  ',': 'SP',
  '항룡': 'NNG',
  '압력': 'NNG',
```

-어휘 별로 끊어져 있는 것을 문장으로 만들어 꼬꼬마 형태소 분석기로 분석하고,
(아웃라인에 출력된 것은 꼬꼬마 형태소 분석기가 분석한 형태소)
가장 많이 쓰인 형태소와 일치하는 것을 feature로 두어 훈련을 시키려고 계획하였습니다.
그러나 코딩 과정에서 해결되지 않는 점이 있어 정확도를 산출하지 못했습니다.

5.마치며
-의외로 끝 두 글자로 학습한 것이 가장 높은 정확도를 보였습니다.
더 다양하고 높은 정확도의 feature extraction이 존재할 것인데 아직 시도해보지 못해서 아쉽습니다. 그리고 자연어처리의 가장 중요한 부분이 데이터 정제임을 깨닫게 되는 과제였습니다. 텍스트를 모델링하게 쉽게 변환하는 과정이 가장 힘들었기 때문입니다.