언컴 기말과제_원종빈 (2018131605)

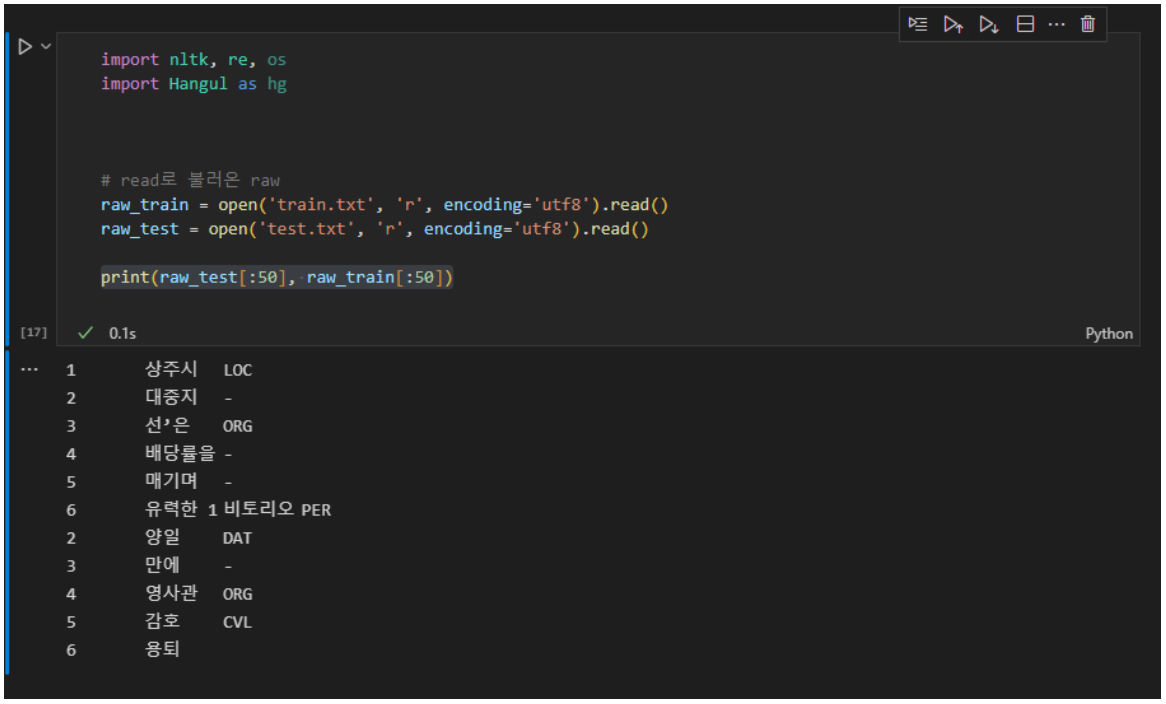
생성일	@2022년 7월 12일 오후 11:27

본 과제는 원본 데이터에 정답(라벨, 태그)이 붙어있으므로, 지도 기계학습이라고 할 수 있다.

일반적인 지도기계학습은 다음과 같은 절차를 따른다.

I. 데이터 불러오기 및 데이터 라벨

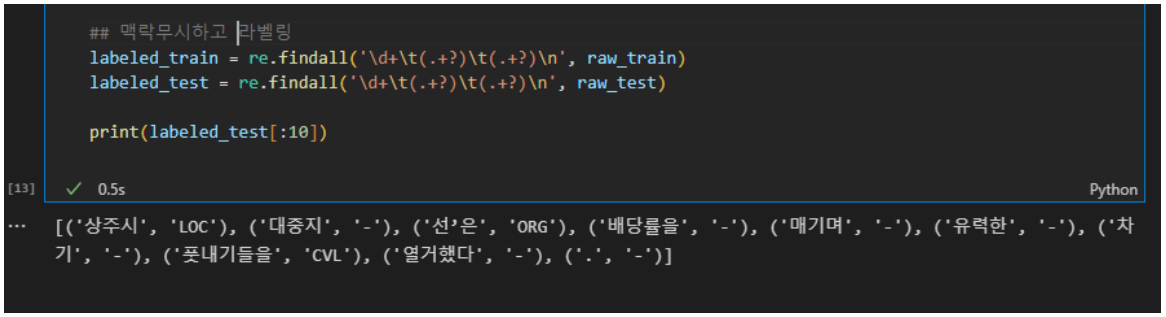
우선 주어진 데이터와 사용가능한 모듈들을 python상에 불러오는 것부터 시작한다.



다음으로 txt 형식으로 주어진 데이터들에 대해서 input값과 그에 따른 label을 구분해 리스트로 정리한다.

이 과정에서 `nltk.bigrams()`, `nltk.trigrams()`등을 활용해 표적이 되는 단어 좌우의 맥락을 고려하는 방법도 있으나, 그렇게 하면 처리해야할 데이터의 양이 최소 2배 이상 많아진다는 단점이 있다.

따라서 우선은 맥락을 고려하지 않고, 단어 하나하나에만 집중해 과제를 수행하기로 한다.



~~II. 셔플링과 training set, test set의 구분~~

원칙적으로 기계학습에서는 주어진 데이터를 무작위로 shuffling하여, 모델을 학습시키기 위한 training set, 정확도를 개선시키기 위한 devtest set, 그리고 모델의 정확도를 측정하기 위한 test set을 구분하는 과정을 거쳐야한다.

이는 모델이 학습한 데이터에 대해서만 정확도가 높게 나오는 경우를 방지하기 위해서이다.

그러나 본 과제에서는 이미 train set과 test set이 구분되어 있고, devtest set은 고려하지 않으므로 이러한 절차를 생략한다.

III. 특징추출(feature extraction)

본 과제는 나이브베이즈분류기(Naive Bayes Classifier)를 활용한다.

이는 train set에서 설정된 특징들과 그에 따른 정답(라벨, 태그)을 기계학습한 뒤, 해당 특징들을 기반으로 test set의 정답(라벨, 태그)을 예측하는 식으로 진행된다.

따라서 적절한 특징을 설정하는 것이 중요하다고 할 수 있다.

0. 데이터의 특성 파악

적절한 특징을 설정하기 위해서 주어진 데이터의 특성을 살펴보기로 한다.

1. 다음은 train.txt의 112번줄~134번줄을 가져온 것이다.

```
1 22회말 NUM
2 21-21 NUM
3 동점타의  -
4 꼬냑인  CVL
5 LG  ORG
6 막둥이  CVL
7 이종열은  PER
8 "팀이  -
9 어려울  -
10  때  -
11  귀중한  -
12  타점을  -
13  올려 -
14  원기가  -
15  좋다 -
16  . -
```

```
1 ▶  -
2 케빈 PER
3 가넷,  PER
4 보스턴행  LOC
5 '확정'…올스타  CVL
6 19인방 NUM
7 '기대만발'  -
1 이상주기자  PER
2 divayuni@-  TRM
3 주소창에  -
4 '스포츠'만  -
5 치시면  -
6 유디치과그룹 ORG
7 기졸이  -
8 한눈에  ANM
9 ! -
```

원본 txt파일은 하나의 문장을 단순히 공백문자를 기준으로 단어로 나누고, 마침표(온점, 느낌표, 물음표)역시 별개의 단어로 처리한다고 예상해볼 수 있다.

곧, 단어 좌우에 붙어있는 쉼표, 말따옴표 등 마침표가 아닌 문장부호들까지 포함해 하나의 단어로 처리됨을 확인할 수 있다.

2. train set의 태그 분포를 확인해본다.

```
[2]  print(nltk.FreqDist(tag for (_, tag) in labeled_train).most_common(50))
                                                                    Python
...  [('-', 437495), ('NUM', 38910), ('CVL', 36946), ('PER', 28919), ('ORG', 27218), ('DAT', 20355), ('TRM',
     13185), ('LOC', 12722), ('EVT', 10401), ('ANM', 3935), ('AFW', 3618), ('TIM', 2690), ('FLD', 1437), ('PLT',
     180), ('MAT', 128)]
```

그 결과, 비개체명 태그(-)가 압도적으로 많이 나타났으며, 개체명 태그는 숫자(NUM), 문명 및 문화에 관련된 단어(CVL), 인물명(PER) 순으로 많이 나타나는 것을 확인할 수 있다.

1. 데이터 전처리: 좌우의 문장부호 및 조사 제거 → 88.68%

1. 선술했듯 본 데이터는 기본적으로 마침표가 아닌 문장부호는 하나의 단어로 인식한다.

   따라서 `토끼`, `토끼,`, `"토끼` 처럼 내용상으로는 같은 단어도, 문장부호 유무에 따라서 다른 단어로 인식하는 경우가 발생할 수 있다.

따라서 본격적으로 특징추출 하기 전, 다음과 같은 함수를 통해서 단어 좌우에 붙은 문장부호를 제거하는 절차를 진행한다.

```
def word_stripper(word):
    try:
        return re.search('\\b(.+)\\b', word).group(1)
    except:
        return word

for word in ['토끼,', '토끼.', "'토끼'", '.']:
    print(word_stripper(word))
```

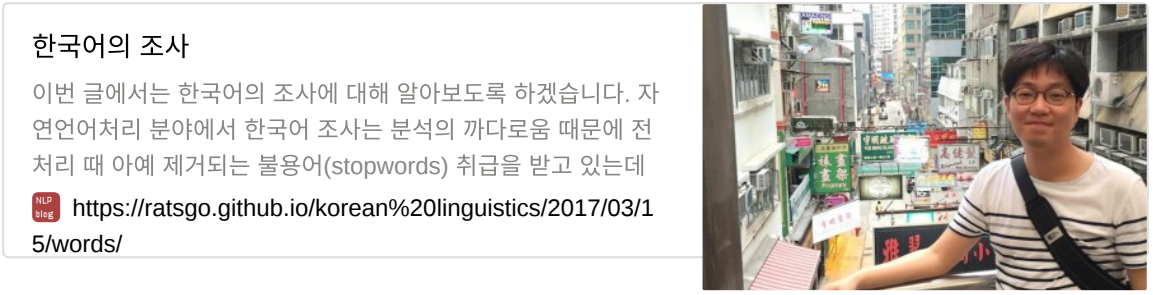
[14] ✓ 0.3s Python

```
토끼
토끼
토끼
.
```

2. 또한 `토끼를`, `토끼의`, `토끼는`처럼단어 뒤에 조사가 붙어있는 경우에도, 내용상 같은 단어를 다르게 인식할 수 있다.

   따라서 본격적으로 특징추출하기 전, 다음과 같은 조사 목록과 함수를 통하여 조사를 제거하는 절차를 진행한다.

   조사 목록을 작성하는 데 있어서는 다음의 사이트를 참고하였다.



```python
particle_list = [
    ##빈도수 > 2000000
    '[에]?(은|는)', '(을|를)', '(이|가)', '[와과]?의', '에', '(로|으로)',
    ## 빈도수 > 500000
    '(와|과)', '도', '에서', '[이]?[지]?만',
    ## 빈도수 > 50000
    '(이나|나)', '까지', '부터', '(에게|께)', '보다', '처럼',
    '(이라도|라도)', '(으로서|로서)', '조차', '만큼', '같이', '마저', '(이나마|나마)', '(한테|더러)',
    '(에게서|한테서|께서)', '이야', '이라[야고]?',
]


def tail_maker(word):

    s_word = word_stripper(word)

    for particle in particle_list:
        if bool(re.search('[들]?'+particle + '$', s_word)) == True:
            return particle

    return ''


for word in ['토끼를', '토끼는', '토끼가']:
    print(tail_maker(word), word.rstrip(tail_maker(word)))
```

[15] ✓ 0.2s Python

```
(을|를) 토끼
[에]?(은|는) 토끼
(이|가) 토끼
```

3. 선술한 두가지 절차를 거쳐, 문장부호와 조사를 제거한 단어를 활용해 다음과 같은 특징 추출함수를 만들었다.

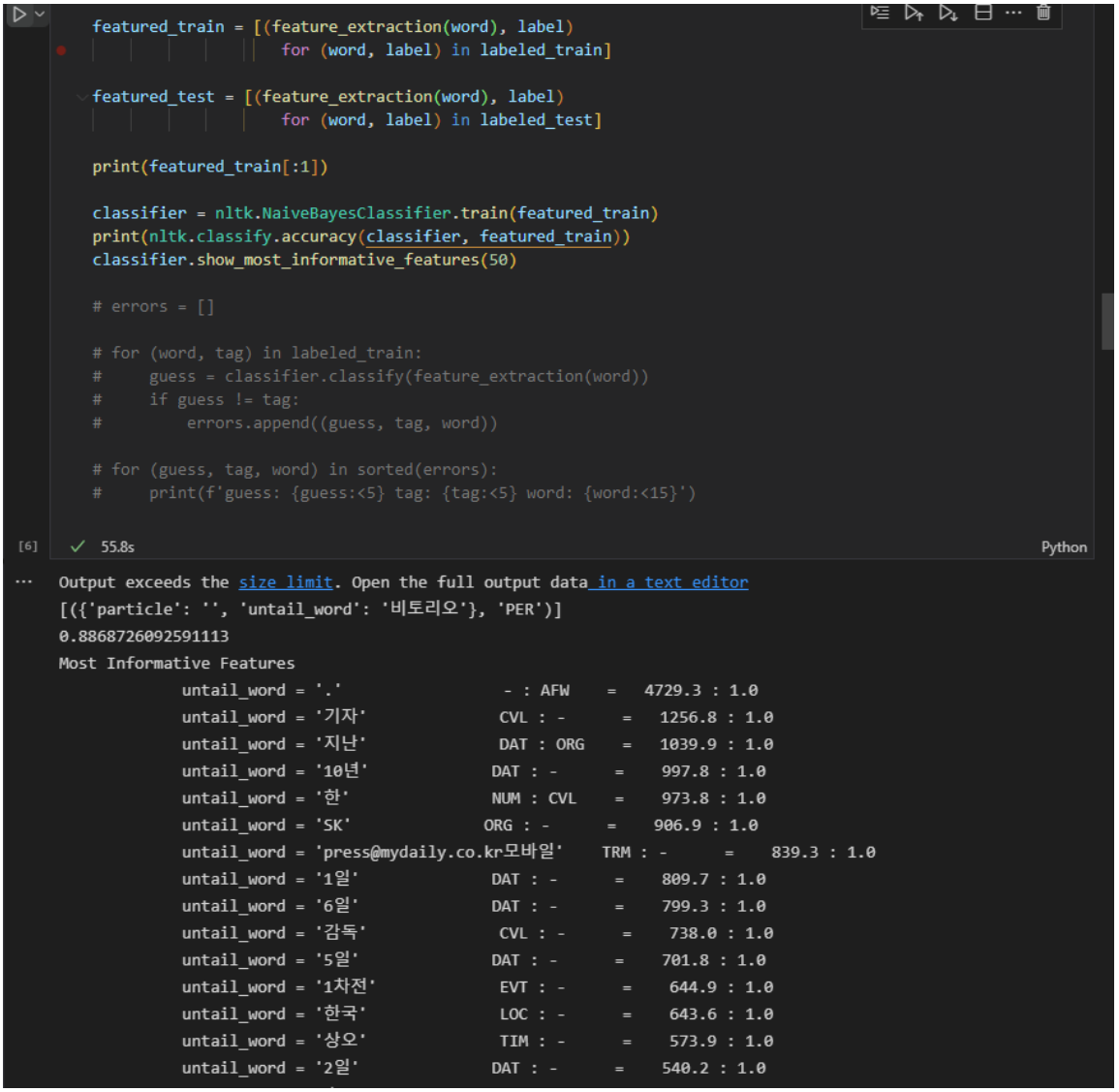
```python
def feature_extraction(word):
    features = {}

    s_word = word_stripper(word)
    word_tail = tail_maker(s_word)
    t_word = s_word.rstrip(word_tail)

    ## 양 끝 문장부호, 조사 분리
    features['particle'] = word_tail
    features['untail_word'] = t_word

    return features

print(feature_extraction("'토끼는,"))
```

[64] ✓ 0.4s Python

```
{'particle': '[에]?(은|는)', 'untail_word': '토끼'}
```

4. 본 함수를 활용해 featured train data를 만들고, 이를 활용해 모델을 학습시켰다. 그리고 동시에 featured_train을 devtest로 활용해 정확도와 유의미한 특징을 추출해보았다.

그 결과, 단순히 조사와 문장부호를 제거하는 것만으로도 약 88.68%의 정확도를 확보할 수 있었다.



```
featured_train = [(feature_extraction(word), label)
                  for (word, label) in labeled_train]

featured_test = [(feature_extraction(word), label)
                 for (word, label) in labeled_test]

print(featured_train[:1])

classifier = nltk.NaiveBayesClassifier.train(featured_train)
print(nltk.classify.accuracy(classifier, featured_train))
classifier.show_most_informative_features(50)

# errors = []

# for (word, tag) in labeled_train:
#     guess = classifier.classify(feature_extraction(word))
#     if guess != tag:
#         errors.append((guess, tag, word))

# for (guess, tag, word) in sorted(errors):
#     print(f'guess: {guess:<5} tag: {tag:<5} word: {word:<15}')
```

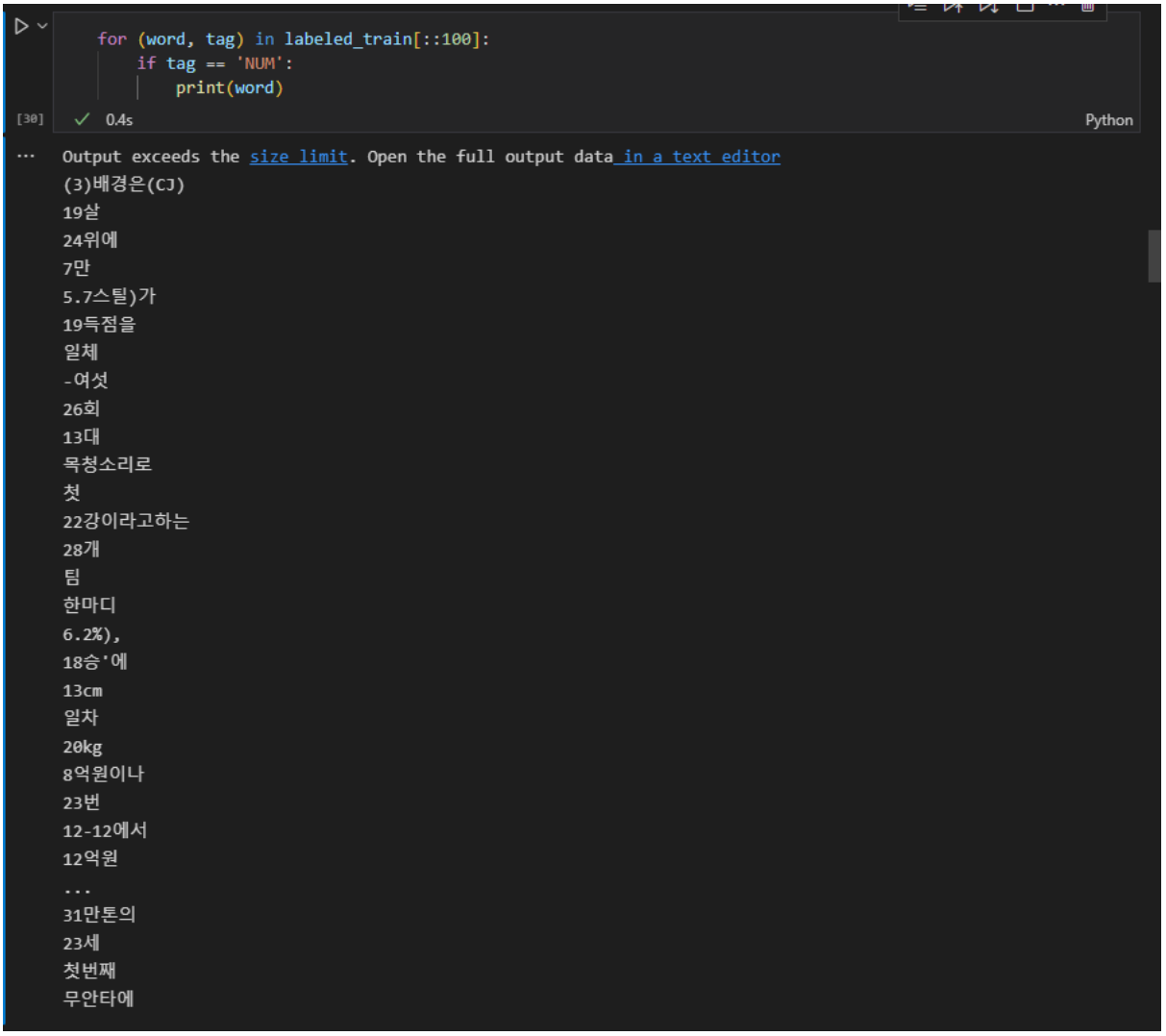
[6] ✓ 55.8s Python

```
Output exceeds the size limit. Open the full output data in a text editor
[({'particle': '', 'untail_word': '비토리오'}, 'PER')]
0.8868726092591113
Most Informative Features
             untail_word = '.'                - : AFW    =   4729.3 : 1.0
             untail_word = '기자'             CVL : -      =   1256.8 : 1.0
             untail_word = '지난'             DAT : ORG    =   1039.9 : 1.0
             untail_word = '10년'             DAT : -      =    997.8 : 1.0
             untail_word = '한'               NUM : CVL    =    973.8 : 1.0
             untail_word = 'SK'              ORG : -      =    906.9 : 1.0
             untail_word = 'press@mydaily.co.kr모바일'    TRM : -      =    839.3 : 1.0
             untail_word = '1일'              DAT : -      =    809.7 : 1.0
             untail_word = '6일'              DAT : -      =    799.3 : 1.0
             untail_word = '감독'             CVL : -      =    738.0 : 1.0
             untail_word = '5일'              DAT : -      =    701.8 : 1.0
             untail_word = '1차전'            EVT : -      =    644.9 : 1.0
             untail_word = '한국'             LOC : -      =    643.6 : 1.0
             untail_word = '상오'             TIM : -      =    573.9 : 1.0
             untail_word = '2일'              DAT : -      =    540.2 : 1.0
```

2. NUM 태그 정확도 향상

다음으로, 개체명 태그 중 가장 많은 수를 차지했던 NUM 태그에 대한 정확도를 향상시켜본다.

다음의 코드를 통해서, NUM 태그가 붙는 단어들을 살펴본다.



그 결과, NUM tag가 붙는 단어는 크게 다음의 세가지 경우로 구분할 수 있었다.

a. 아라비아 숫자가 포함되는 경우 (ex. `19살`, `24위에`, `7만`)

b. 한글 숫자 표현이 포함되는 경우 (ex. `일체`, `-여섯`, `첫`, `한마디`)

c. 앞뒤 단어가 NUM 태그로 분류되어, 본 단어 역시 NUM 태그로 분류되는 경우 (ex. `목청리로`, `팀`)

이중 3번째 경우는 전후맥락을 파악해야하므로, 논외로 하고, 첫번째와 두번째 경우에 대해서 다루기로 한다.

A. 아라비아 숫자가 포함되는 경우의 처리 → 88.40%

1. 우선 다음과 같은 함수를 활용해, 단순히 단어 안에 숫자가 포함되었는가 여부(True or False)를 판단하는 함수를 만들어보았다.

```python
def feature_extraction(word):
    features = {}

    s_word = word_stripper(word)
    word_tail = tail_maker(s_word)
    t_word = s_word.rstrip(word_tail)

    ## 양 끝 문장부호, 조사 분리
    features['untail_word'] = t_word
    features['particle or end?'] = word_tail

    ## NUM tag
    features['contain_digit'] = any(i.isdigit() for i in t_word)


    return features

for word in ['1위는', '한사람','1마리','2일']:
    print(feature_extraction(word))
```

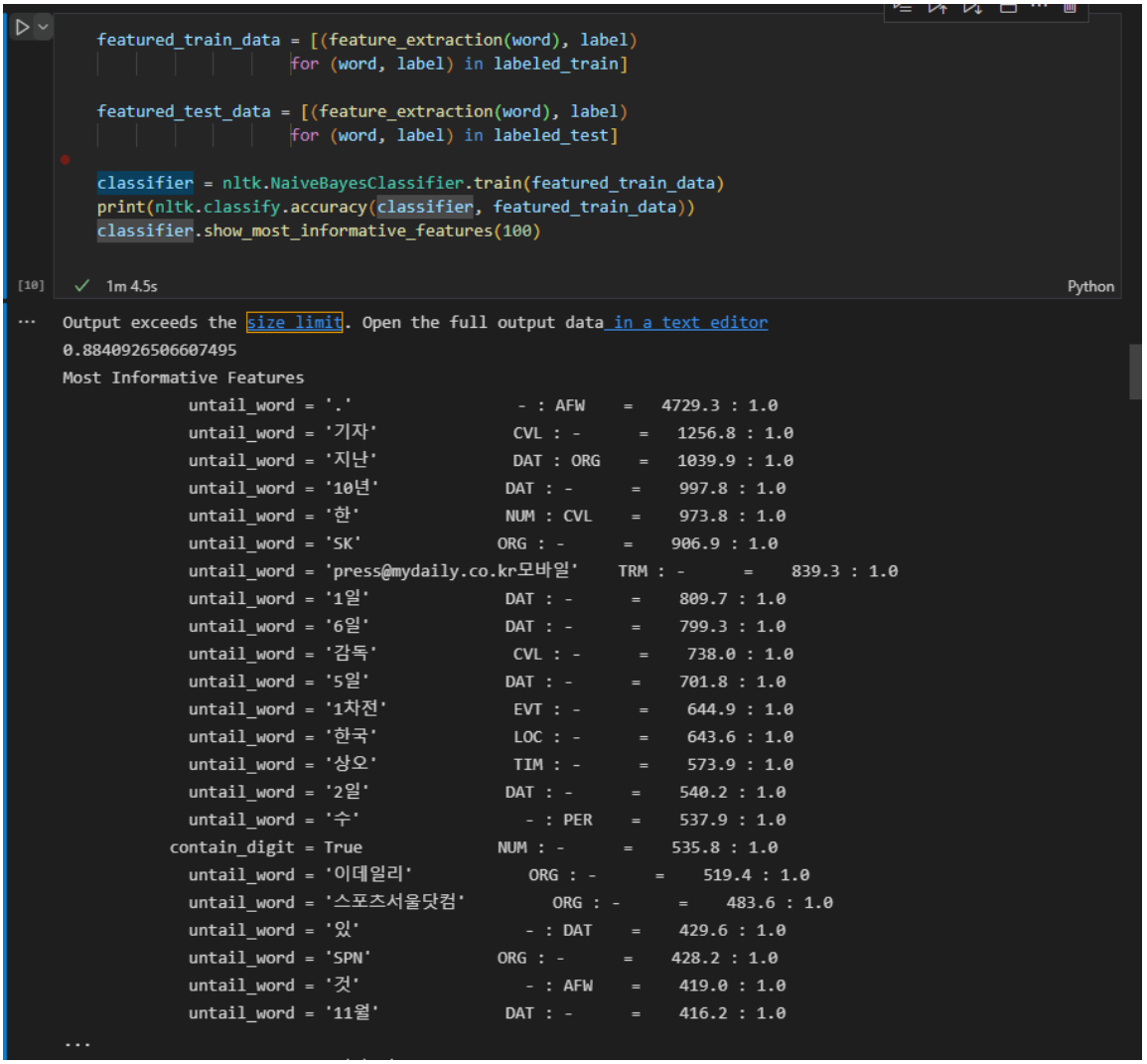
[67] ✓ 0.3s Python

```
{'untail_word': '1위', 'particle or end?': '[에]?(은|는)', 'contain_digit': True}
{'untail_word': '한사람', 'particle or end?': '', 'contain_digit': False}
{'untail_word': '1마리', 'particle or end?': '', 'contain_digit': True}
{'untail_word': '2일', 'particle or end?': '', 'contain_digit': True}
```

2. 이를 활용해 featured train data, featured test data를 만들고, 정확도를 측정해보았다.

   그 결과, 약 88.40%로 되려 정확도가 소폭 감소한 것을 확인할 수 있었다.

   아마 DAT 태그들도 NUM태그로 분류하기에 발생한 문제라고 생각해, 차후 DAT 태그들을 별도의 특징으로 구별하기로 한다.



```
featured_train_data = [(feature_extraction(word), label)
                       for (word, label) in labeled_train]

featured_test_data = [(feature_extraction(word), label)
                      for (word, label) in labeled_test]

classifier = nltk.NaiveBayesClassifier.train(featured_train_data)
print(nltk.classify.accuracy(classifier, featured_train_data))
classifier.show_most_informative_features(100)
```

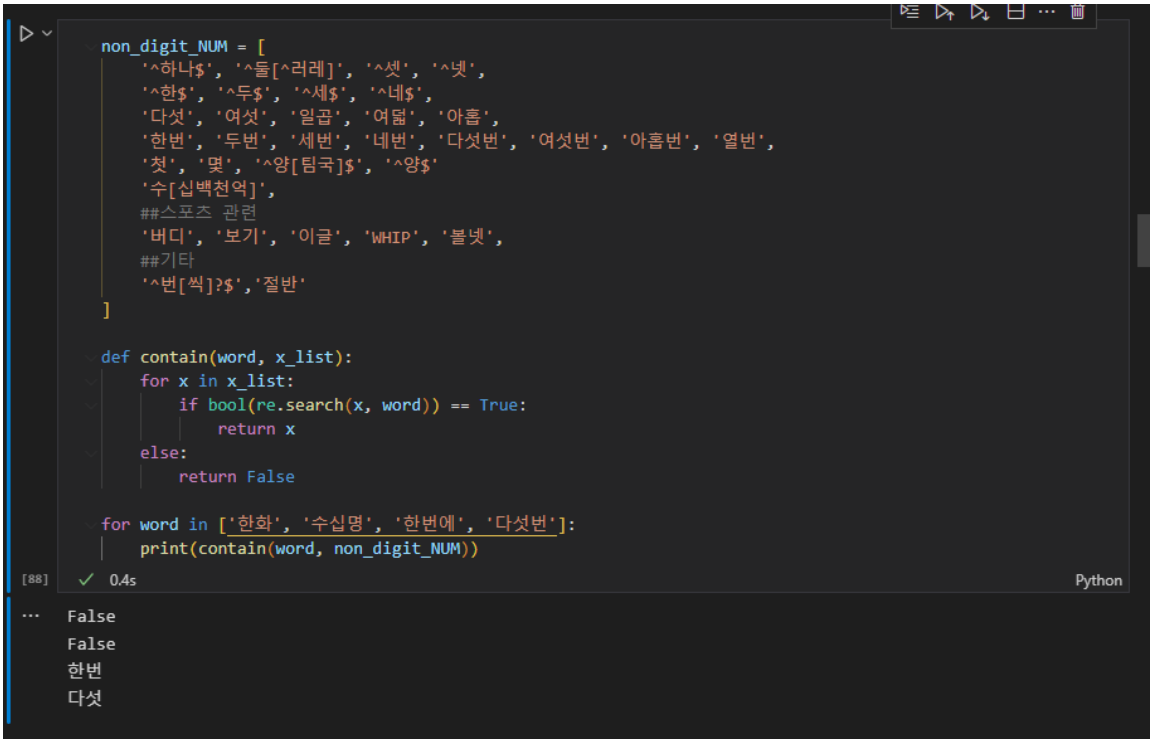
[10] ✓ 1m 4.5s Python

```
Output exceeds the size limit. Open the full output data in a text editor
0.8840926506607495
Most Informative Features
             untail_word = '.'                - : AFW    =   4729.3 : 1.0
             untail_word = '기자'             CVL : -      =   1256.8 : 1.0
             untail_word = '지난'             DAT : ORG    =   1039.9 : 1.0
             untail_word = '10년'             DAT : -      =    997.8 : 1.0
             untail_word = '한'               NUM : CVL    =    973.8 : 1.0
             untail_word = 'SK'              ORG : -      =    906.9 : 1.0
             untail_word = 'press@mydaily.co.kr모바일'    TRM : -      =    839.3 : 1.0
             untail_word = '1일'              DAT : -      =    809.7 : 1.0
             untail_word = '6일'              DAT : -      =    799.3 : 1.0
             untail_word = '감독'             CVL : -      =    738.0 : 1.0
             untail_word = '5일'              DAT : -      =    701.8 : 1.0
             untail_word = '1차전'            EVT : -      =    644.9 : 1.0
             untail_word = '한국'             LOC : -      =    643.6 : 1.0
             untail_word = '상오'             TIM : -      =    573.9 : 1.0
             untail_word = '2일'              DAT : -      =    540.2 : 1.0
             untail_word = '수'                - : PER    =    537.9 : 1.0
           contain_digit = True              NUM : -      =    535.8 : 1.0
             untail_word = '이데일리'          ORG : -      =    519.4 : 1.0
             untail_word = '스포츠서울닷컴'       ORG : -      =    483.6 : 1.0
             untail_word = '있'                - : DAT    =    429.6 : 1.0
             untail_word = 'SPN'             ORG : -      =    428.2 : 1.0
             untail_word = '것'                - : AFW    =    419.0 : 1.0
             untail_word = '11월'             DAT : -      =    416.2 : 1.0
...
```

B. 아라비아 숫자가 포함되지 않은 경우의 처리 → 88.56%

1. 다음으로, 하나, 둘처럼 한글 숫자표현을 리스트로 정리해, 해당 단어들이 포함되는 경우를 고려해보았다.

   이때, `한`, `두`, `세`, `네`처럼 광범위하게 매치될 수 있는 단어들의 경우, 검색조건을 엄격하게 하여 `한화증권`, `두산중공업`, `세밀하게`등 숫자와 관련없는 단어들을 배제하도록 설정하였다.



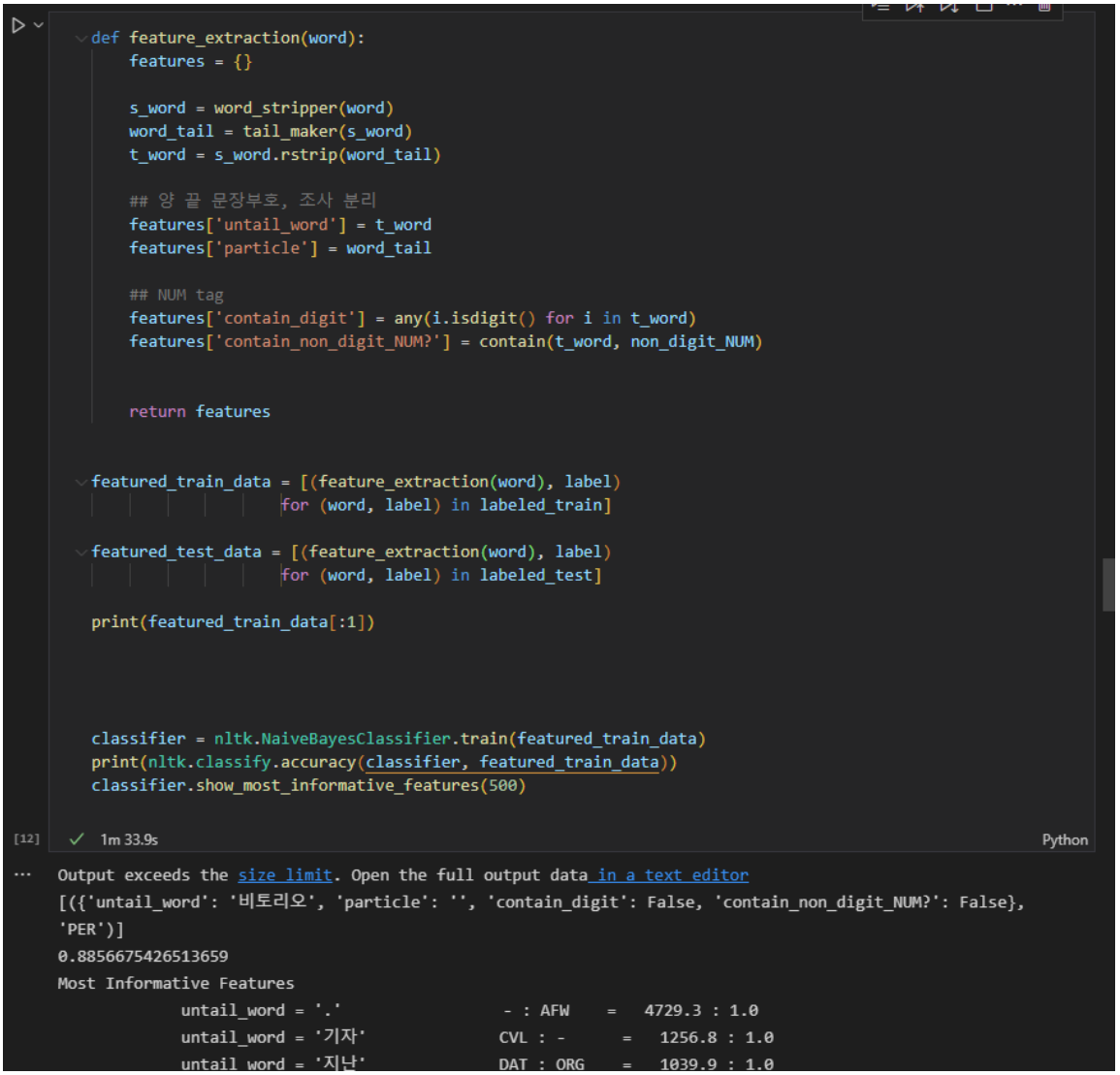
```python
non_digit_NUM = [
    '^하나$', '^둘[^러레]', '^셋', '^넷',
    '^한$', '^두$', '^세$', '^네$',
    '다섯', '여섯', '일곱', '여덟', '아홉',
    '한번', '두번', '세번', '네번', '다섯번', '여섯번', '아홉번', '열번',
    '첫', '몇', '^양[팀국]$', '^양$'
    '수[십백천억]',
    ##스포츠 관련
    '버디', '보기', '이글', 'WHIP', '볼넷',
    ##기타
    '^번[씩]?$','절반'
]

def contain(word, x_list):
    for x in x_list:
        if bool(re.search(x, word)) == True:
            return x
    else:
        return False

for word in ['한화', '수십명', '한번에', '다섯번']:
    print(contain(word, non_digit_NUM))
```

[88] ✓ 0.4s Python

```
False
False
한번
다섯
```

2. 이 함수를 포함해 특징을 추출-학습시켰다,

   그 결과 정확도가 88.56%로 0.16%p가량 상승한 것을 확인할 수 있었다.



```
def feature_extraction(word):
    features = {}

    s_word = word_stripper(word)
    word_tail = tail_maker(s_word)
    t_word = s_word.rstrip(word_tail)

    ## 양 끝 문장부호, 조사 분리
    features['untail_word'] = t_word
    features['particle'] = word_tail

    ## NUM tag
    features['contain_digit'] = any(i.isdigit() for i in t_word)
    features['contain_non_digit_NUM?'] = contain(t_word, non_digit_NUM)


    return features


featured_train_data = [(feature_extraction(word), label)
                       for (word, label) in labeled_train]

featured_test_data = [(feature_extraction(word), label)
                      for (word, label) in labeled_test]

print(featured_train_data[:1])



classifier = nltk.NaiveBayesClassifier.train(featured_train_data)
print(nltk.classify.accuracy(classifier, featured_train_data))
classifier.show_most_informative_features(500)
```

[12] ✓ 1m 33.9s Python

```
Output exceeds the size limit. Open the full output data in a text editor
[({'untail_word': '비토리오', 'particle': '', 'contain_digit': False, 'contain_non_digit_NUM?': False},
'PER')]
0.8856675426513659
Most Informative Features
            untail_word = '.'              - : AFW    =   4729.3 : 1.0
            untail_word = '기자'           CVL : -      =   1256.8 : 1.0
            untail_word = '지난'           DAT : ORG    =   1039.9 : 1.0
```

3. DAT 태그 목록 활용 → 88.54%

1. 다음과 같은 코드를 통해 지금까지 설계한 모델에서 발생한 분류오류를 파악해본다.

```
errors = []

for (word, tag) in labeled_train:
    guess = classifier.classify(feature_extraction(word))
    if guess != tag:
        errors.append((guess, tag, word))

nltk.ConditionalFreqDist((guess, tag) for (guess, tag, word) in errors).tabulate()
```

[19] ✓ 1m 26.2s Python

	-	AFW	ANM	CVL	DAT	EVT	FLD	LOC	MAT	NUM	ORG	PER	PLT	TIM	TRM
-	0	976	503	2392	1840	1329	576	650	82	3154	1116	968	109	192	2013
AFW	96	0	0	4	1	4	2	8	0	2	14	2	0	0	2
ANM	554	4	0	11	0	2	5	2	0	40	1	9	0	0	20
CVL	4891	258	116	0	38	498	130	175	17	180	525	276	24	5	301
DAT	1589	77	1	75	0	71	4	72	2	219	105	111	1	77	72
EVT	679	69	4	117	6	0	5	109	0	52	194	38	1	5	43
FLD	129	1	0	5	0	1	0	0	0	2	1	2	0	0	3
LOC	637	158	28	149	13	289	12	0	1	30	257	65	4	2	61
NUM	2028	54	19	307	1068	272	3	88	3	0	130	69	0	419	133
ORG	1470	330	40	433	38	1067	68	737	6	26	0	180	4	0	291
PER	1473	150	37	124	12	80	11	219	10	32	233	0	27	5	93
TIM	147	4	0	10	11	11	0	1	0	27	2	6	0	0	2
TRM	1570	106	34	149	7	146	60	43	5	234	152	56	1	0	0

그 결과 비개체명 태그를 제외하면, DAT태그를 NUM태그로 분류한 오류가 가장 많이 발생했음을 파악할 수 있었다.

<u>이는 앞서 단어 안에 숫자가 판단되는지 여부를 특징으로 잡았기, 숫자가 포함된 DAT 단어들을 NUM 태그로 분류해 이러한 오류가 많이 발생했으리라 추측해볼 수 있다.</u>

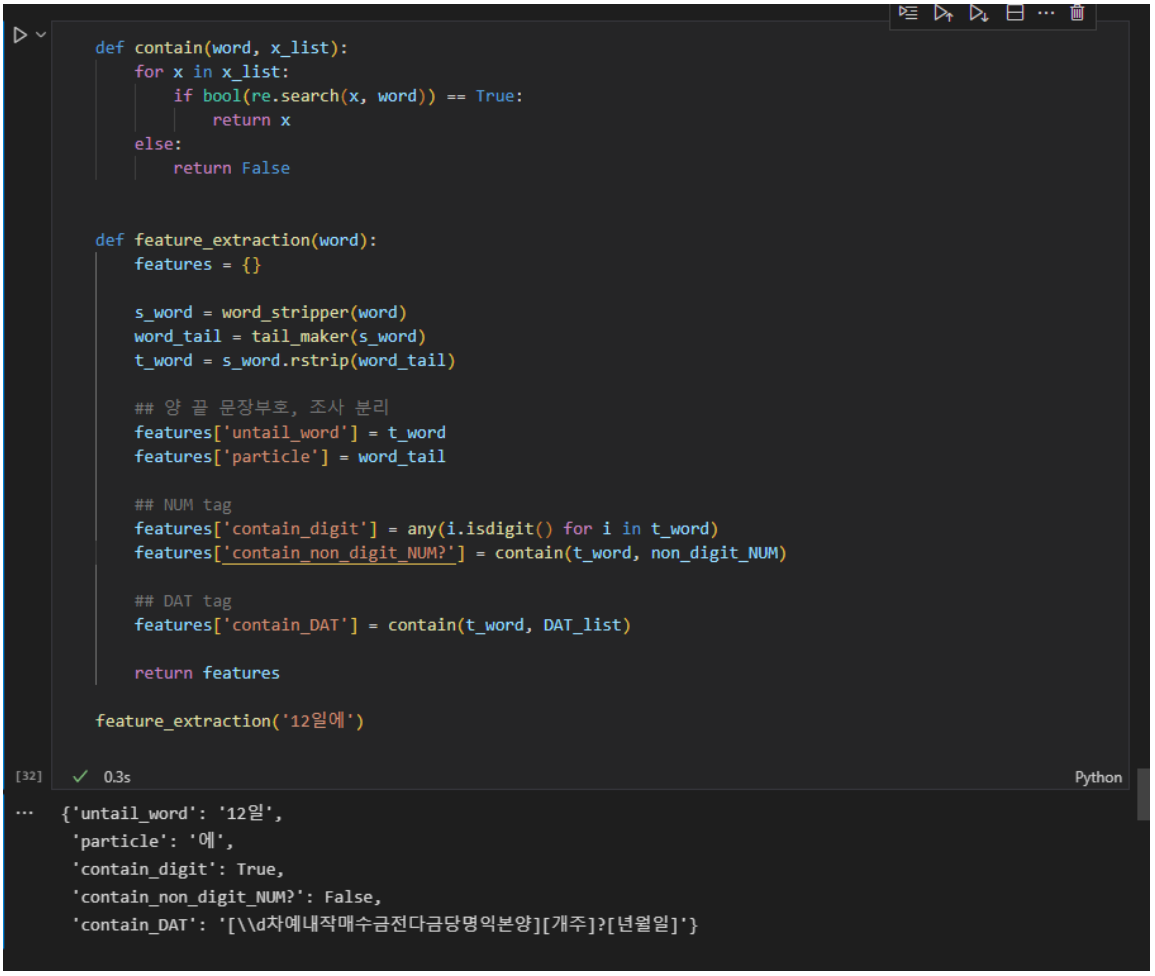
따라서 DAT태그에 대해서 정확도를 향상시키기로 한다.

2. 우선 DAT 태그가 붙는 단어들에서 자주 포함되는 단어 목록을 다음과 같이 작성하였다.

```
DAT_list = [
    '[\d차예내작매수금전다금당명익본양][개주]?[년월일]', '\d[/]', '\d{4}', '\d세기', '\d분기',
    '연[간내말초]',
    ##특수한 단어
    '^지난', '다음', '이번', '예전', '옛날', '전[날에]',
    '오늘', '내일', '날짜', '월경기', '^동안', '유행기', '방학', '일제강점기', '하계',
    '호시기', '휴가철',
    ## 한글표현
    '하루', '이틀','사흘','나흘', '열흘', '며칠', '나흗날', '오뉴월', '한달',
    '이듬해', '올해', '^해$', '^달$', '^주$',
    ## 계절
    '봄', '(여름|서머)', '가을', '겨울', '(계절|시즌)',
    ## 요일
    '[월화수목금토일]요일', '공휴일', '일주간', '일주일', '주말', '주중','일일',
]
```

[22] ✓ 0.3s Python

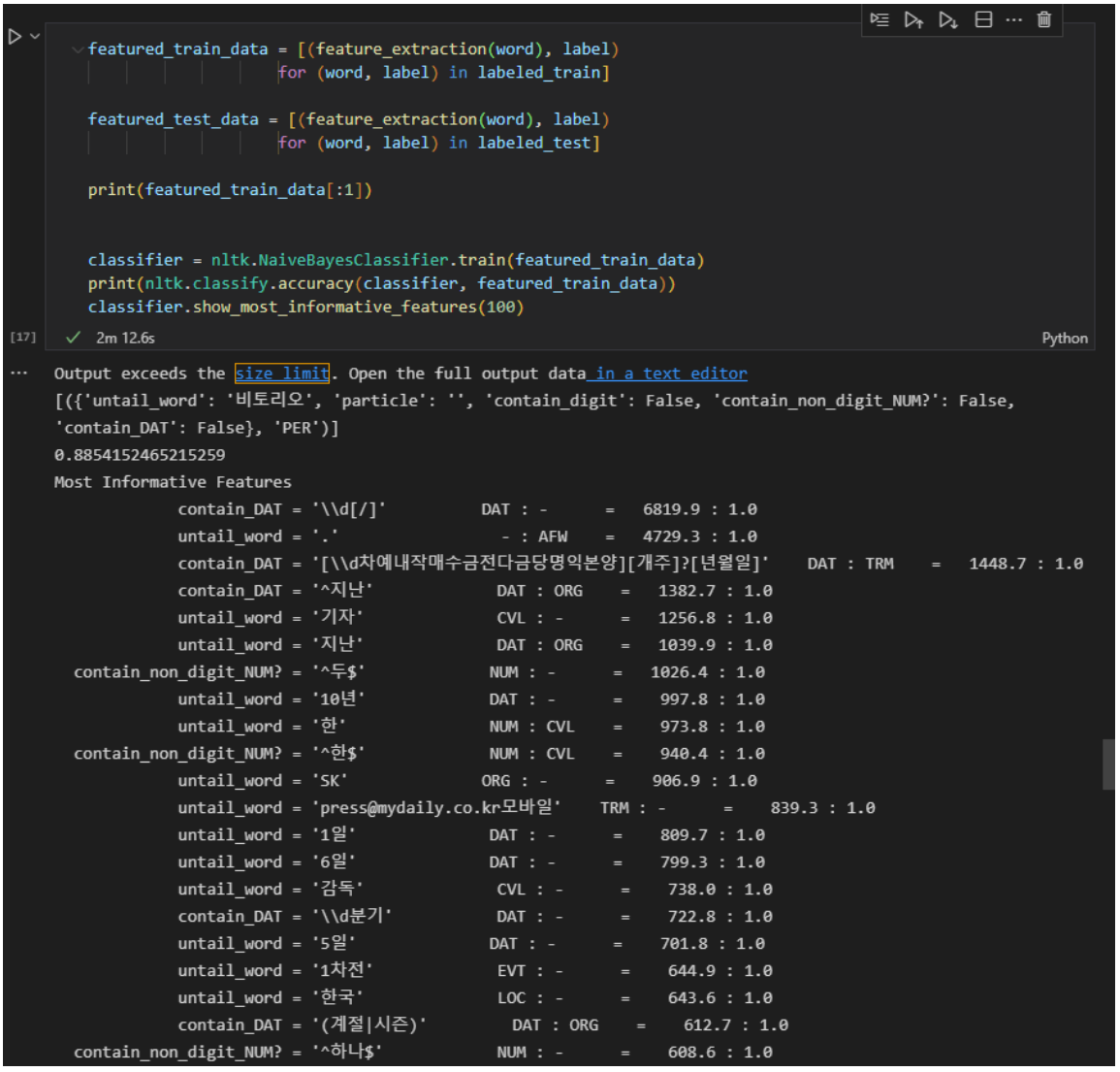
3. 그리고 다음과 같은 특징추출 함수를 통해서, 해당 단어가 포함되는 경우 해당단어를 특징으로 삼도록 모델을 수정하였다.



```python
def contain(word, x_list):
    for x in x_list:
        if bool(re.search(x, word)) == True:
            return x
    else:
        return False


def feature_extraction(word):
    features = {}

    s_word = word_stripper(word)
    word_tail = tail_maker(s_word)
    t_word = s_word.rstrip(word_tail)

    ## 양 끝 문장부호, 조사 분리
    features['untail_word'] = t_word
    features['particle'] = word_tail

    ## NUM tag
    features['contain_digit'] = any(i.isdigit() for i in t_word)
    features['contain_non_digit_NUM?'] = contain(t_word, non_digit_NUM)

    ## DAT tag
    features['contain_DAT'] = contain(t_word, DAT_list)

    return features

feature_extraction('12일에')
```

```
[32]  ✓ 0.3s                                                    Python

...  {'untail_word': '12일',
      'particle': '에',
      'contain_digit': True,
      'contain_non_digit_NUM?': False,
      'contain_DAT': '[\\d차예내작매수금전다금당명익본양][개주]?[년월일]'}
```

4. 이상의 모델을 통해서 정확도를 다시 측정한 결과, 88.54%로 정확도는 큰 차이가 없었으나, 유의미한 특징 목록에도 새로 설정한 특징이 포함된 것을 확인할 수 있었다.



```
featured_train_data = [(feature_extraction(word), label)
                       for (word, label) in labeled_train]

featured_test_data = [(feature_extraction(word), label)
                      for (word, label) in labeled_test]

print(featured_train_data[:1])


classifier = nltk.NaiveBayesClassifier.train(featured_train_data)
print(nltk.classify.accuracy(classifier, featured_train_data))
classifier.show_most_informative_features(100)
```

[17] ✓ 2m 12.6s Python

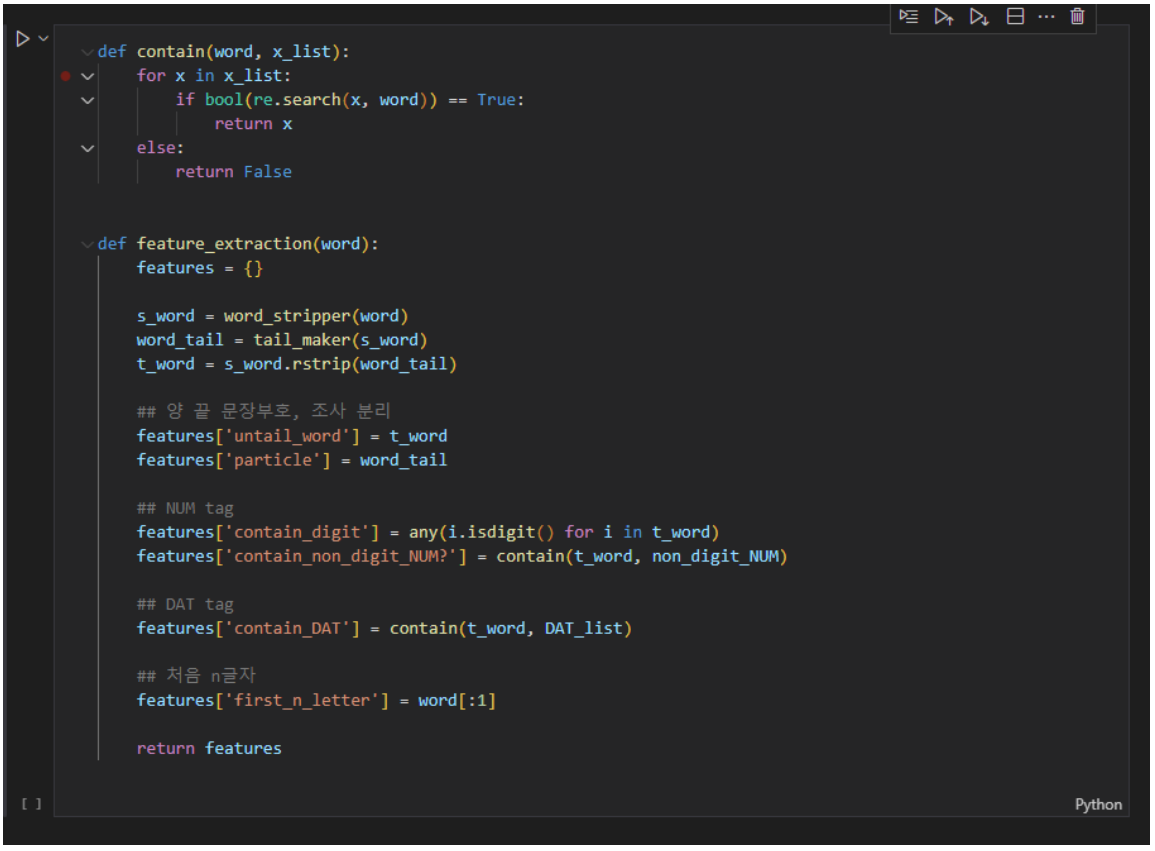
```
Output exceeds the size limit. Open the full output data in a text editor
[({'untail_word': '비토리오', 'particle': '', 'contain_digit': False, 'contain_non_digit_NUM?': False,
'contain_DAT': False}, 'PER')]
0.8854152465215259
Most Informative Features
              contain_DAT = '\\d[/]'          DAT : -      =   6819.9 : 1.0
              untail_word = '.'                 - : AFW    =   4729.3 : 1.0
              contain_DAT = '[\\d차예내작매수금전다금당명익본양][개주]?[년월일]'    DAT : TRM    =   1448.7 : 1.0
              contain_DAT = '^지난'             DAT : ORG    =   1382.7 : 1.0
              untail_word = '기자'              CVL : -      =   1256.8 : 1.0
              untail_word = '지난'              DAT : ORG    =   1039.9 : 1.0
  contain_non_digit_NUM? = '^두$'             NUM : -      =   1026.4 : 1.0
              untail_word = '10년'              DAT : -      =    997.8 : 1.0
              untail_word = '한'                NUM : CVL    =    973.8 : 1.0
  contain_non_digit_NUM? = '^한$'             NUM : CVL    =    940.4 : 1.0
              untail_word = 'SK'               ORG : -      =    906.9 : 1.0
              untail_word = 'press@mydaily.co.kr모바일'    TRM : -      =    839.3 : 1.0
              untail_word = '1일'               DAT : -      =    809.7 : 1.0
              untail_word = '6일'               DAT : -      =    799.3 : 1.0
              untail_word = '감독'              CVL : -      =    738.0 : 1.0
              contain_DAT = '\\d분기'           DAT : -      =    722.8 : 1.0
              untail_word = '5일'               DAT : -      =    701.8 : 1.0
              untail_word = '1차전'             EVT : -      =    644.9 : 1.0
              untail_word = '한국'              LOC : -      =    643.6 : 1.0
              contain_DAT = '(계절|시즌)'         DAT : ORG    =    612.7 : 1.0
  contain_non_digit_NUM? = '^하나$'           NUM : -      =    608.6 : 1.0
```

4. 처음 n글자

마지막으로, 단어의 처음에서부터 n번째 글자를 특징으로 하여, 가장 정확도가 높게 나오는 경우를 측정해본다.

A. 처음 1글자 → 90.45%

1. 우선 처음 한글자에 대해서 특징을 추출하는 함수를 만들었다.



```python
def contain(word, x_list):
    for x in x_list:
        if bool(re.search(x, word)) == True:
            return x
    else:
        return False


def feature_extraction(word):
    features = {}

    s_word = word_stripper(word)
    word_tail = tail_maker(s_word)
    t_word = s_word.rstrip(word_tail)

    ## 양 끝 문장부호, 조사 분리
    features['untail_word'] = t_word
    features['particle'] = word_tail

    ## NUM tag
    features['contain_digit'] = any(i.isdigit() for i in t_word)
    features['contain_non_digit_NUM?'] = contain(t_word, non_digit_NUM)

    ## DAT tag
    features['contain_DAT'] = contain(t_word, DAT_list)

    ## 처음 n글자
    features['first_n_letter'] = word[:1]

    return features
```

[ ]

Python

2. 그리고 그에 따른 정확도를 측정한 결과, 정확도는 90.45%였다.

```python
def feature_extraction(word):
    features = {}

    s_word = word_stripper(word)
    word_tail = tail_maker(s_word)
    t_word = s_word.rstrip(word_tail)

    ## 양 끝 문장부호, 조사 분리
    features['untail_word'] = t_word
    features['particle'] = word_tail

    ## NUM tag
    features['contain_digit'] = any(i.isdigit() for i in t_word)
    features['contain_non_digit_NUM?'] = contain(t_word, non_digit_NUM)

    ## DAT tag
    features['contain_DAT'] = contain(t_word, DAT_list)

    ## 처음 n글자
    features['first_n_letter'] = word[:1]

    return features

featured_train_data = [(feature_extraction(word), label)
                       for (word, label) in labeled_train]

featured_test_data = [(feature_extraction(word), label)
                      for (word, label) in labeled_test]

print(featured_train_data[:1])


classifier = nltk.NaiveBayesClassifier.train(featured_train_data)
print(nltk.classify.accuracy(classifier, featured_train_data))
classifier.show_most_informative_features(500)
```

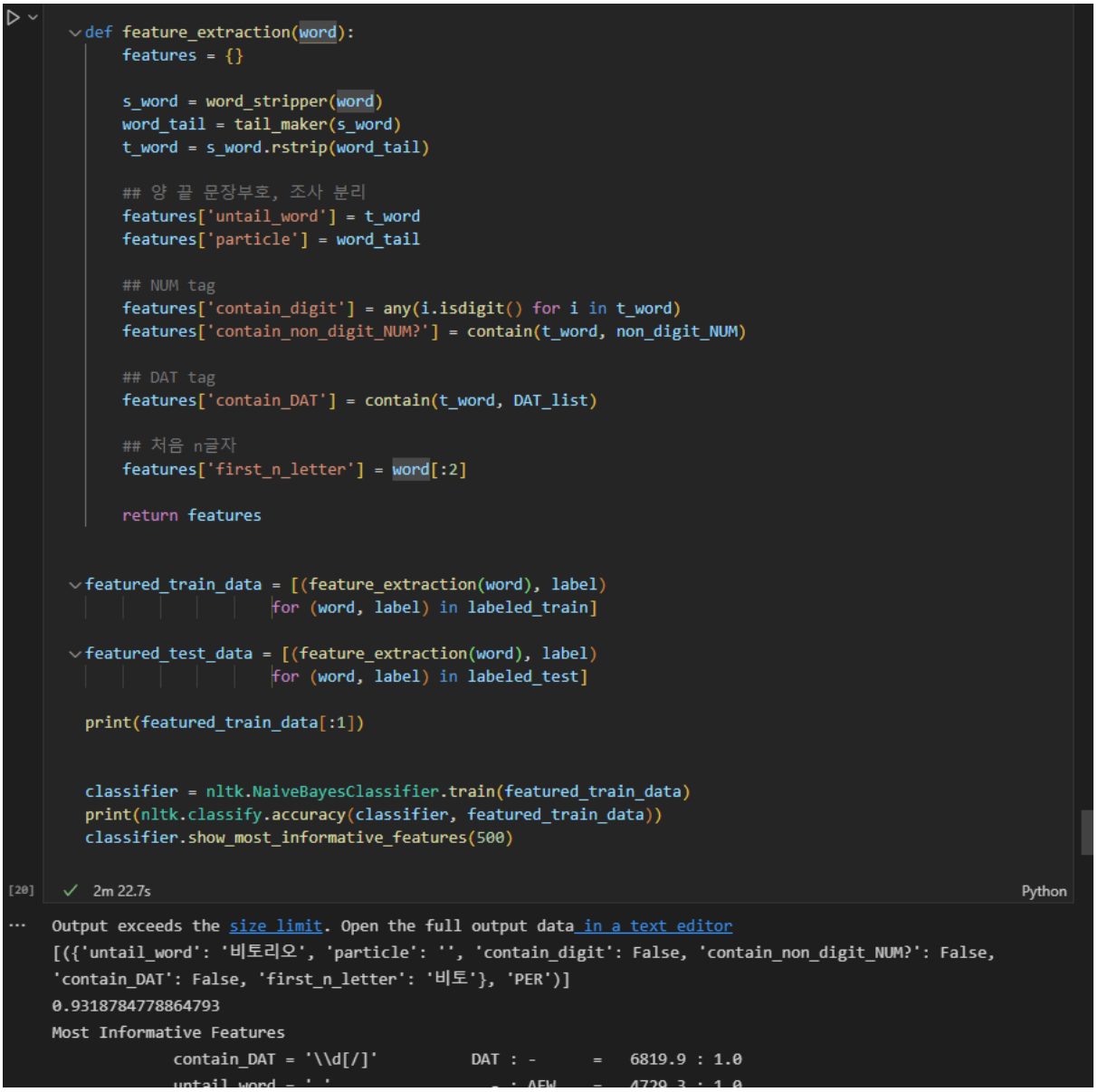
[18] ✓ 2m 22.5s Python

```
Output exceeds the size limit. Open the full output data in a text editor
[({'untail_word': '비토리오', 'particle': '', 'contain_digit': False, 'contain_non_digit_NUM?': False,
'contain_DAT': False, 'first_n_letter': '비'}, 'PER')]
0.9045411736314501
Most Informative Features
             contain_DAT = '\\d[/]'          DAT : -      =   6819.9 : 1.0
          first_n_letter = '1'               NUM : PER    =   4848.0 : 1.0
```

B. 처음 2글자 → 93.18%

위와 같은 절차를 반복하여, 처음 2글자에 대해서도 정확도를 측정해보았다.

그 결과 정확도는 약 93.18%로, 한 글자만 특징으로 설정했던 경우보다 상승한 것을 확인할 수 있다.



```python
def feature_extraction(word):
    features = {}

    s_word = word_stripper(word)
    word_tail = tail_maker(s_word)
    t_word = s_word.rstrip(word_tail)

    ## 양 끝 문장부호, 조사 분리
    features['untail_word'] = t_word
    features['particle'] = word_tail

    ## NUM tag
    features['contain_digit'] = any(i.isdigit() for i in t_word)
    features['contain_non_digit_NUM?'] = contain(t_word, non_digit_NUM)

    ## DAT tag
    features['contain_DAT'] = contain(t_word, DAT_list)

    ## 처음 n글자
    features['first_n_letter'] = word[:2]

    return features


featured_train_data = [(feature_extraction(word), label)
                       for (word, label) in labeled_train]

featured_test_data = [(feature_extraction(word), label)
                      for (word, label) in labeled_test]

print(featured_train_data[:1])


classifier = nltk.NaiveBayesClassifier.train(featured_train_data)
print(nltk.classify.accuracy(classifier, featured_train_data))
classifier.show_most_informative_features(500)
```

[20] ✓ 2m 22.7s Python

```
Output exceeds the size limit. Open the full output data in a text editor
[({'untail_word': '비토리오', 'particle': '', 'contain_digit': False, 'contain_non_digit_NUM?': False,
'contain_DAT': False, 'first_n_letter': '비토'}, 'PER')]
0.9318784778864793
Most Informative Features
            contain_DAT = '\\d[/]'          DAT : -      =   6819.9 : 1.0
```

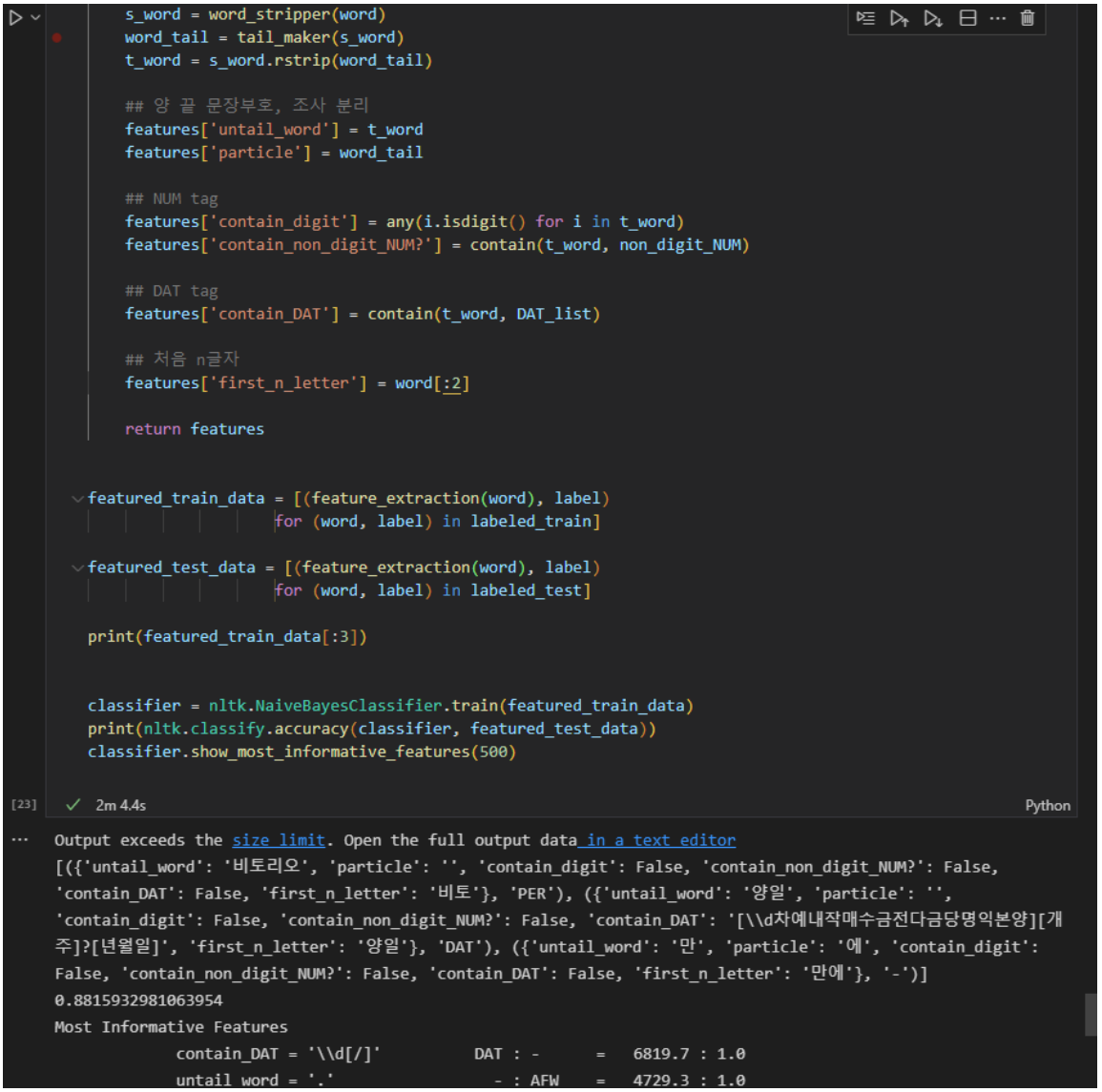
III. 최종 모델 설정 및 test set 정확도 측정 → 88.15%

1. 이상의 결과들을 모두 종합한 결과, 가장 정확도가 높게 측정되었던 첫 두 글자를 특징으로 결정하고, 다음과 같은 최종 모델을 설정하였다.

```
DAT_list = [
    '[\d차예내작매수금전다금당명익본양][개주]?[년월일]',
    '\d[/]', '\d{4}', '\d세기', '\d분기',
    '연[간내말초]',
    ##특수한 단어
    '^지난', '다음', '이번', '예전', '옛날', '전[날에]',
    '오늘', '내일', '날짜', '월경기', '^동안', '유행기',
    '방학', '일제강점기', '하계',
    '호시기', '휴가철',
    ## 한글표현
    '하루', '이틀','사흘','나흘', '열흘', '며칠', '나흗날', '오뉴월', '한달',
    '이듬해', '올해', '^해$', '^달$', '^주$',
    ## 계절
    '봄', '(여름|서머)', '가을', '겨울', '(계절|시즌)',
    ## 요일
    '[월화수목금토일]요일', '공휴일', '일주간', '일주일', '주말', '주중','일일',
    ## 수식어
    '^오는$', '^올$',
]

non_digit_NUM = [
    '^하나$', '^둘[^러레]', '^셋', '^넷',
    '^한$', '^두$', '^세$', '^네$',
    '다섯', '여섯', '일곱', '여덟', '아홉',
    '한번', '두번', '세번', '네번', '다섯번', '여섯번', '아홉번', '열번',
    '첫', '몇', '^양[팀국]$', '^양$'
    '수[십백천억]',
    ##스포츠 관련
    '버디', '보기', '이글', 'WHIP', '볼넷',
    ##기타
    '^번[씩]?$','절반'
]


def contain(word, x_list):
    for x in x_list:
        if bool(re.search(x, word)) == True:
            return x
    else:
        return False

def feature_extraction(word):
    features = {}

    s_word = word_stripper(word)
    word_tail = tail_maker(s_word)
    t_word = s_word.rstrip(word_tail)

    ## 양 끝 문장부호, 조사 분리
    features['untail_word'] = t_word
    features['particle'] = word_tail

    ## NUM tag
    features['contain_digit'] = any(i.isdigit() for i in t_word)
    features['contain_non_digit_NUM?'] = contain(t_word, non_digit_NUM)
```

```
    ## DAT tag
    features['contain_DAT'] = contain(t_word, DAT_list)

    ## 처음 n글자
    features['first_n_letter'] = word[:2]

    return features
```

2. 그리고 최종적으로 **train set(devtest set)이 아닌, test set을 대상으로** 정확도를 측정한 결과, 88.15%의 정확도를 확인할 수 있었다.



```
    s_word = word_stripper(word)
    word_tail = tail_maker(s_word)
    t_word = s_word.rstrip(word_tail)

    ## 양 끝 문장부호, 조사 분리
    features['untail_word'] = t_word
    features['particle'] = word_tail

    ## NUM tag
    features['contain_digit'] = any(i.isdigit() for i in t_word)
    features['contain_non_digit_NUM?'] = contain(t_word, non_digit_NUM)

    ## DAT tag
    features['contain_DAT'] = contain(t_word, DAT_list)

    ## 처음 n글자
    features['first_n_letter'] = word[:2]

    return features


featured_train_data = [(feature_extraction(word), label)
                       for (word, label) in labeled_train]

featured_test_data = [(feature_extraction(word), label)
                      for (word, label) in labeled_test]

print(featured_train_data[:3])


classifier = nltk.NaiveBayesClassifier.train(featured_train_data)
print(nltk.classify.accuracy(classifier, featured_test_data))
classifier.show_most_informative_features(500)
```

```
[23]  ✓ 2m 4.4s                                                    Python

...  Output exceeds the size limit. Open the full output data in a text editor
[({'untail_word': '비토리오', 'particle': '', 'contain_digit': False, 'contain_non_digit_NUM?': False,
'contain_DAT': False, 'first_n_letter': '비토'}, 'PER'), ({'untail_word': '양일', 'particle': '',
'contain_digit': False, 'contain_non_digit_NUM?': False, 'contain_DAT': '[\\d차예내작매수금전다금당명익본양][개
주]?[년월일]', 'first_n_letter': '양일'}, 'DAT'), ({'untail_word': '만', 'particle': '에', 'contain_digit':
False, 'contain_non_digit_NUM?': False, 'contain_DAT': False, 'first_n_letter': '만에'}, '-')]
0.8815932981063954
Most Informative Features
             contain_DAT = '\\d[/]'          DAT : -      =   6819.7 : 1.0
             untail_word = '.'                 - : AFW    =   4729.3 : 1.0
```