MYONGJI UNIVERSITY

Artificial Intelligence

Lecture 6. Text Mining Part 1

Spring 2022

Prof. Jonghoon Chun, Ph.D.

E-mail : jchun@mju.ac.kr
Lecture Note: https://lms.mju.ac.kr

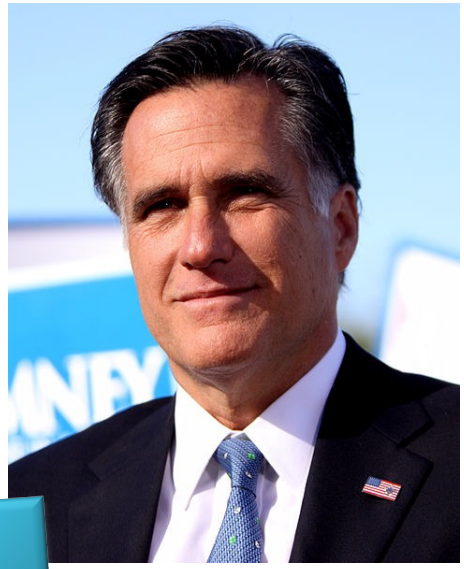
Agenda

- Text Mining 개요
- 영문 텍스트 분석
- 한글 텍스트 처리 기법
- WordCloud

Agenda

- Text Mining 개요
- 영문 텍스트 분석
- 한글 텍스트 처리 기법
- WordCloud

Document Classification



누구의 연설문인가?

?

?

Thank you, New Hampshire! Tonight, we made [illegible]y!

This state has always been a special place for our family. Ann and I made a home here and we've filled it with great memories of our children and grandchildren. And this Granite State moment is one we will always remember.

Tonight, we celebrate. Tomorrow, we go back to work.

We remember when Barack Obama came to New Hampshire four years ago.

He promised to bring people together.

He promised to change the broken system in Washington.

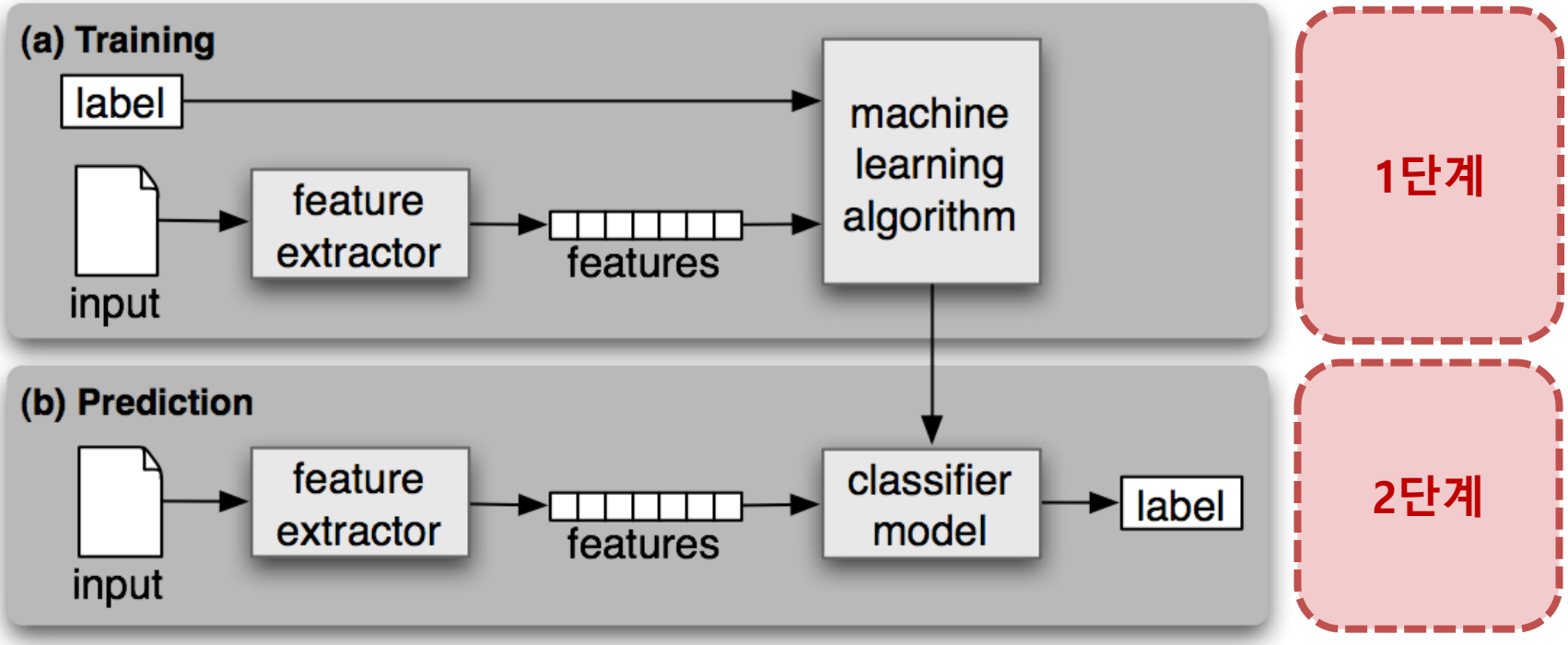
He promised to improve our nation.

Those were the days of lofty promises made by a hopeful candidate. Today, we are faced with the disappointing record of a failed President. The last three years have held a lot of change, but they haven't offered much hope.

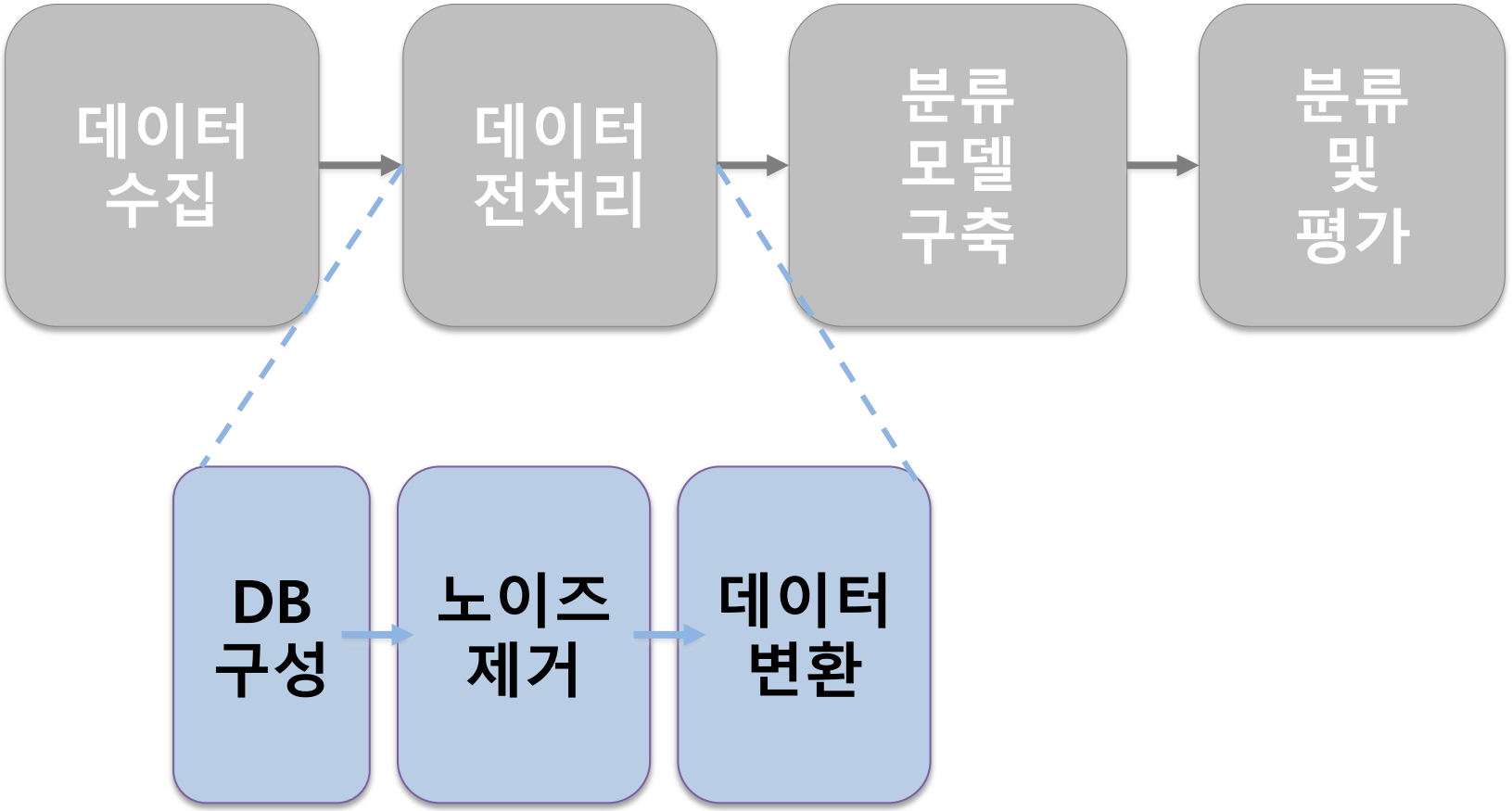
Document Classification

자동 분류 (Automatic Classification)

- Document (automatic) classification
  - 전통적인 machine learning 문제
- Training (훈련) vs. Prediction (예측)
- Learning (학습) vs. Classification (분류)



Automatic Classification System Process



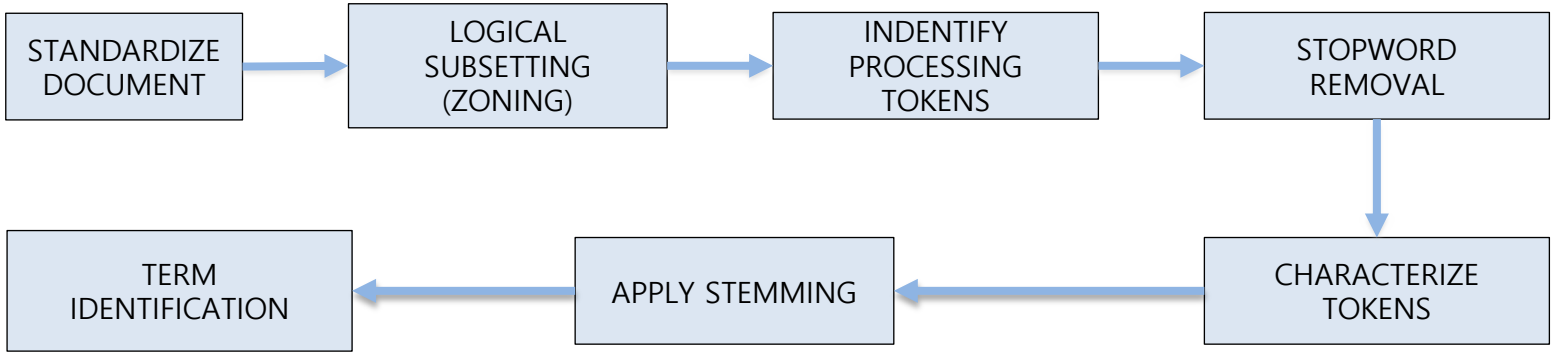
Document Data Model

- Vector Space Model
  - The Vector Space Model (VSM) is a way of representing documents through the words that they contain
  - Each document is broken down into a term(word) frequency table
  - Each document is represented as a vector based against the vocabulary
  - 텍스트 문서를 단어 색인 등의 Processing Token으로 구성된 벡터로 표현하는 모델
- E.g.,

	team	coach	play	ball	score	game	win	lost	timeout	season
Document 1	3	0	5	0	2	6	0	2	0	2
Document 2	0	7	0	2	1	0	0	3	0	0
Document 3	0	1	0	0	1	2	2	0	3	0

Term Frequency Table

- Term frequency table generation process (conceptual)



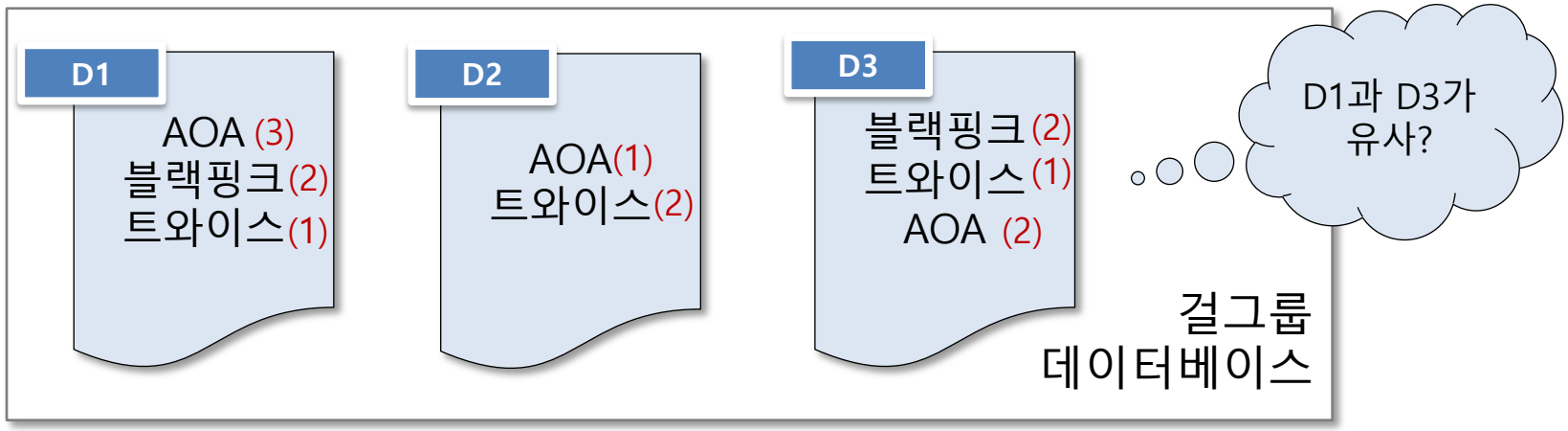
- Term Frequency Table

Document	*team*	*coach*	*hockey*	*baseball*	*soccer*	*penalty*	*score*	*win*	*loss*	*season*
Document1	5	0	3	0	2	0	0	2	0	0
Document2	3	0	2	0	1	1	0	1	0	1
Document3	0	7	0	2	1	0	0	3	0	0
Document4	0	1	0	0	1	2	2	0	3	0

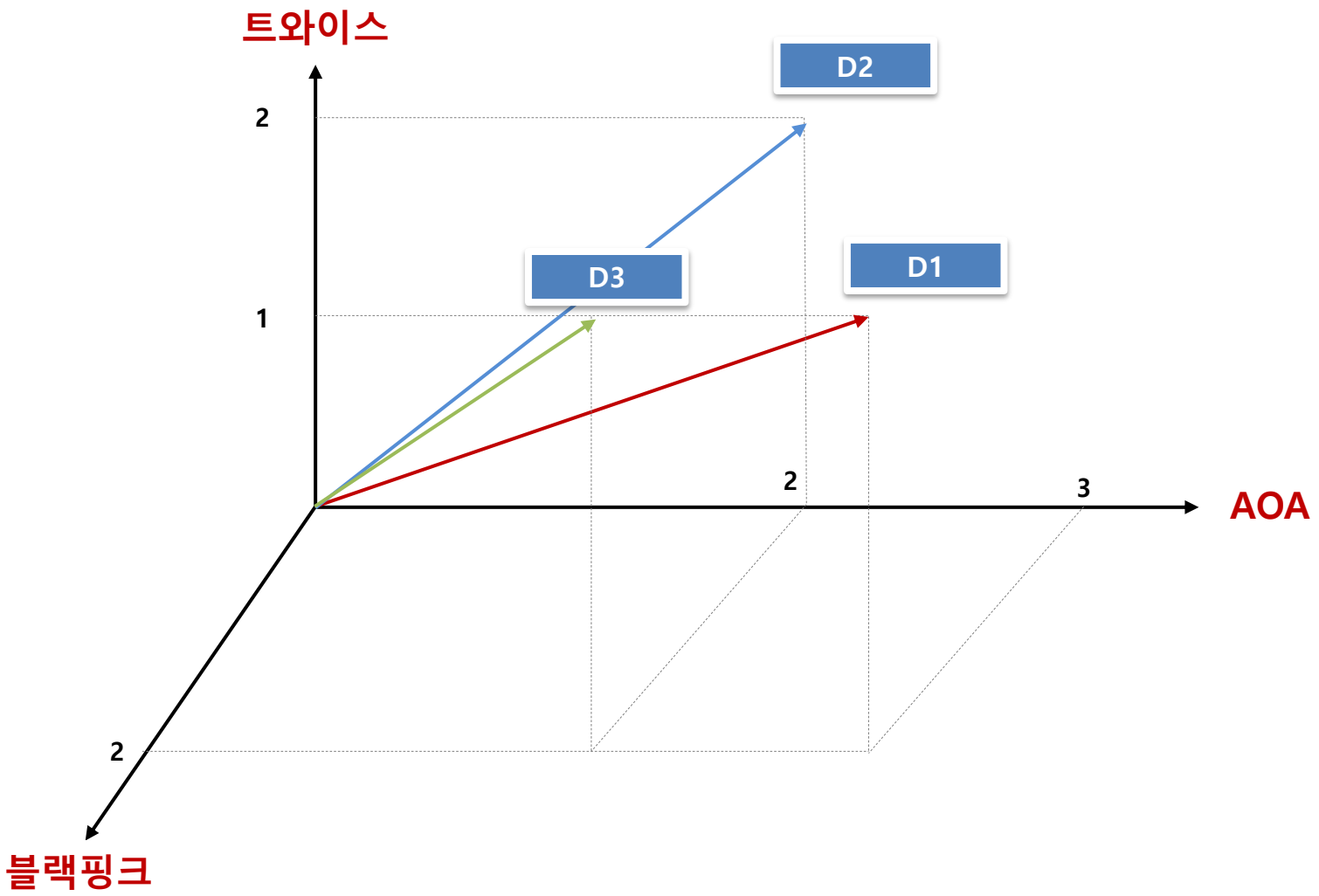
Vector Space Model – an example

- 문서(document)가 벡터
  - Term 하나 하나가 벡터의 차원이 됨!

- Assumption
  - 가정 1: 단어는 단 3개만 존재한다고 가정(AOA, 블랙핑크, 트와이스)
  - 가정 2: Database에 단 3건의 문서만 저장



Vector Space Model – an example



A simpler example

- Document 1: “A dog and a cat.”
- Document 2: “A Frog”

- 전체 term들의 집합
  - { a, dog, and, cat, frog }

- Stopword(불용어) 제거
  - {dog, cat, frog}

- Document 1 vector
  - (1, 1, 0)

Dog	Cat	Frog
1	1	0

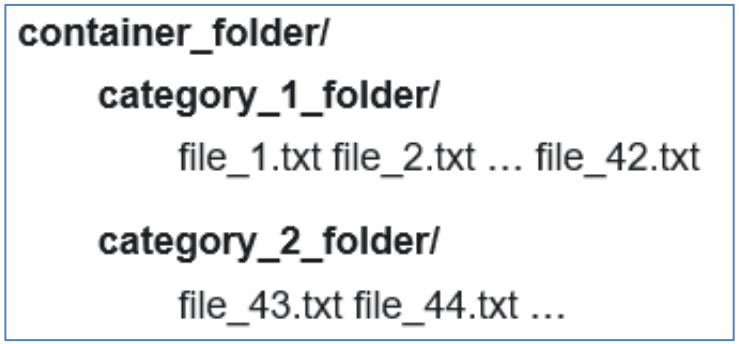
- Document 2 vector
  - (0, 0, 1)

Dog	Cat	Frog
0	0	1

Text Data Loading

- Scikit-learn의 load_files 기능 활용
  - Load text files with categories as subfolder names
  - text와 label을 자동적으로 붙여준 특별 bunch타입 객체를 반환
  - 레이블은 subdirectrory의 알파벳 순서에 따라 0부터 숫자 부여
  - subdirectrory 에서만 데이터를 읽음. (바로 아래 파일들은 무시)

- Bunch
  - 리스트 타입의 data와 target 속성이 부여되는데, data는 text 파일 하나당 원소 하나가 입력되며, target은 subdirectrory의 알파벳 순서에 따라 0, 1, 2, ….와 같은 label이 부여됨



data: [file_1.txt, file_2.txt, …, file_43.txt, file_44.txt,…]

target: [0, 0, 0, …, 1, 1, 1,…]
→ 실제로는 random하게 shuffle됨

Text Data Loading – an example

- 영어 영화평 예제 링크: http://ai.stanford.edu/~amaas/data/sentiment/
  - ./data/aclimdb/test
  - ./data/aclimdb/test/neg
  - ./data/aclimdb/test/pos
  - ./data/aclimdb/train
  - ./data/aclimdb/train/neg
  - ./data/aclimdb/train/pos

- reviews_train = load_files("./data/aclImdb_v1/aclImdb/train")
  - Data list: reviews_train.data
  - ./data/aclimdb/train/neg/와 ./data/aclimdb/train/pos/의 text파일들을 읽어, 문자열 형태의 리스트 구성
  - Target list: reviews_train.target
  - ./data/aclimdb/train/neg와 ./data/aclimdb/train/pos에는 각각 0과 1을 부여한 리스트 구성

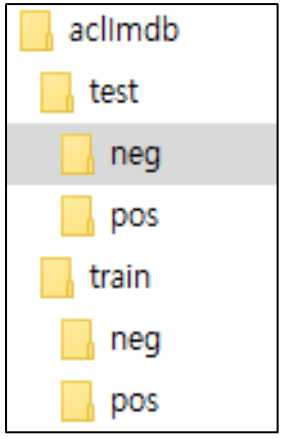
Text Data Loading

```
from sklearn.datasets import load_files

reviews_train = load_files("/Users/jonghoonchun/Downloads/aclImdb/train")
# 텍스트와 레이블을 포함하고 있는 Bunch object를 반환함

text_train, y_train = reviews_train.data, reviews_train.target

print(type(reviews_train))

print("text_train의 타입: {}".format(type(text_train)))
print("text_train의 길이: {}".format(len(text_train)))
print("text_train[6]:\n{}".format(text_train[6]))
print("y_train[6]:\n{}".format(y_train[6]))
```

```
In [10]: print(y_train[1:10])

[0 1 0 0 1 1 0 0 1]
```

Shuffling

```
<class 'sklearn.utils.Bunch'>
text_train의 타입: <class 'list'>
text_train의 길이: 25000
text_train[6]:
b"This movie has a special way of telling the story, at first i found it
rather odd as it jumped through time and I had no idea whats happening.<b
r /><br />Anyway the story line was although simple, but still very real
and touching. You met someone the first time, you fell in love completel
y, but broke up at last and promoted a deadly agony. Who hasn't go throug
h this? but we will never forget this kind of pain in our life. <br /><br
/>I would say i am rather touched as two actor has shown great performanc
e in showing the love between the characters. I just wish that the story
could be a happy ending."
y_train[6]:
1
```



디렉토리 구조

Text Data Loading

- Text data중에서
 삭제

In [7]:
```
print(text_train[5], "\n")
print(type(text_train[5]), "\n")

# Python3 에서는 text_train 데이터가 문자열이 아닌 문자열 데이터의
# binary encoding인 bytes 타입이므로 문자열 앞에 b를 삽입해야 함
text_train = [doc.replace(b"<br />", b" ") for doc in text_train]

print(text_train[5])
```

```
b"The Movie was sub-par, but this Television Pilot delivers a great springboard into what
eal program. The Actors deliver and the special effects (for a television series) are spec
ent interesting script doesn't hurt either.<br /><br />Stargate SG1 is currently one of my

<class 'bytes'>

b"The Movie was sub-par, but this Television Pilot delivers a great springboard into what
eal program. The Actors deliver and the special effects (for a television series) are spec
ent interesting script doesn't hurt either.  Stargate SG1 is currently one of my favorite
```

Text Data Loading

- 클래스 별 샘플 수 확인

```
In [6]: import numpy as np
        print("클래스별 샘플 수 (훈련 데이터): {}".format(np.bincount(y_train)))

        클래스별 샘플 수 (훈련 데이터): [12500 12500]
```

  - np.bincount
    - Count number of occurrences of each value in array of non-negative integers

Text Data Loading

- Full code

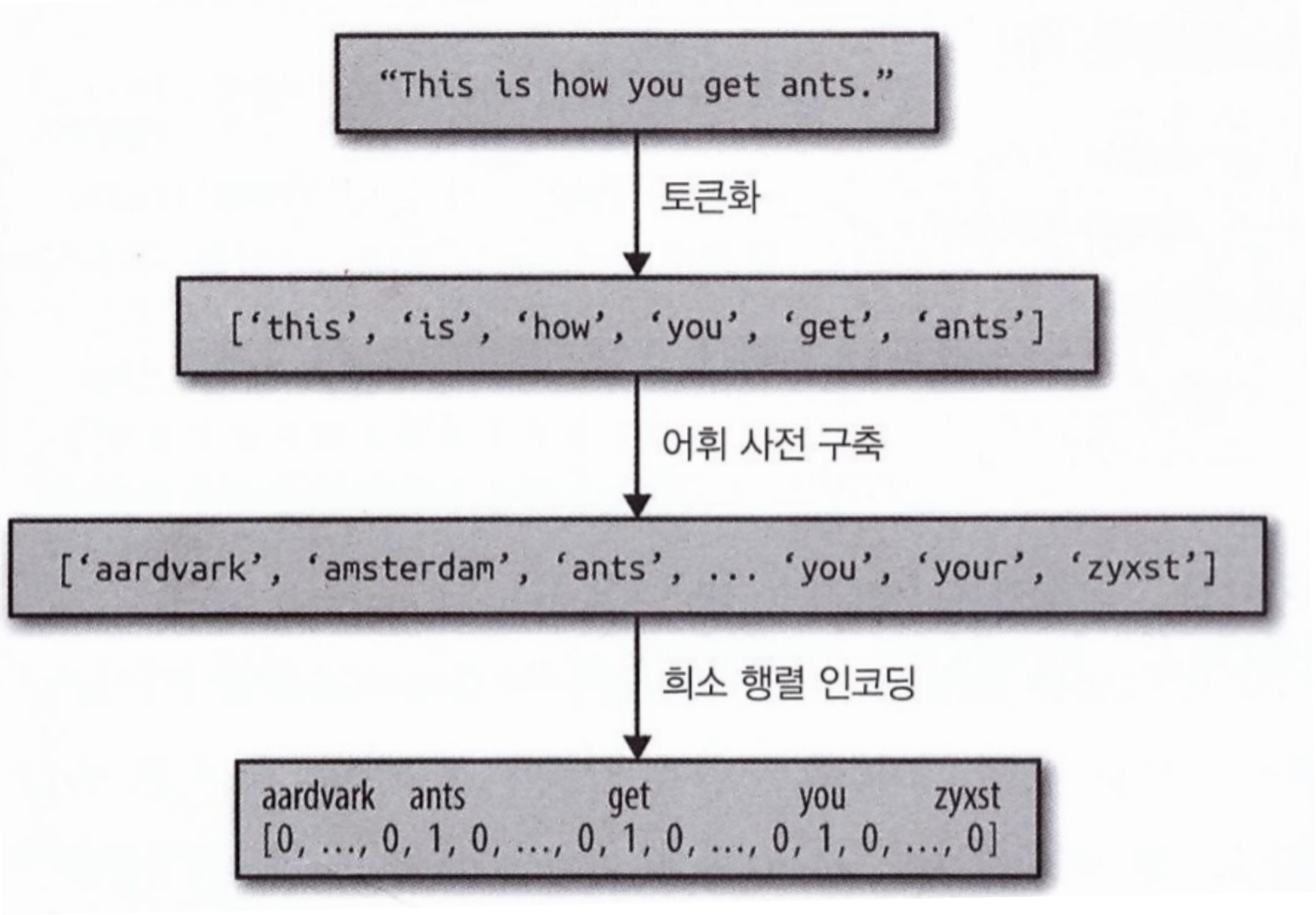
In [8]:
```
from sklearn.datasets import load_files
import numpy as np

reviews_train = load_files("/Users/jonghoonchun/Downloads/aclImdb/train")
reviews_test = load_files("/Users/jonghoonchun/Downloads/aclImdb/test")

text_train, y_train = reviews_train.data, reviews_train.target
text_test, y_test = reviews_test.data, reviews_test.target

text_train = [doc.replace(b"<br />", b" ") for doc in text_train]
text_test = [doc.replace(b"<br />", b" ") for doc in text_test]


print("학습 데이터의 문서 수: {}".format(len(text_train)))
print("테스트 데이터의 문서 수: {}".format(len(text_test)))

print("클래스별 샘플 수 (학습 데이터): {}".format(np.bincount(y_train)))
print("클래스별 샘플 수 (테스트 데이터): {}".format(np.bincount(y_test)))
```

```
학습 데이터의 문서 수: 25000
테스트 데이터의 문서 수: 25000
클래스별 샘플 수 (학습 데이터): [12500 12500]
클래스별 샘플 수 (테스트 데이터): [12500 12500]
```

Bag Of Words(BOW)

- Vector Space Model로 Document를 표현
- Term들의 출현 빈도수로 표현
- BOW 생성과정
  - 토큰화(tokenization)
    - 문서를 공백이나 구두점을 기준으로 term(token)들로 분리
  - 어휘사전 구축
    - 모든 문서에 나타나는 term들을 수집하여 사전구축
  - 인코딩
    - 어휘사전의 각 term이 document別로 몇 번 출현하는 지를 계산하여 벡터로 표현

Bag Of Words


"This is how you get ants."
토큰화
['this', 'is', 'how', 'you', 'get', 'ants']
어휘 사전 구축
['aardvark', 'amsterdam', 'ants', ... 'you', 'your', 'zyxst']
희소 행렬 인코딩
aardvark ants get you zyxst
[0, ..., 0, 1, 0, ..., 0, 1, 0, ..., 0, 1, 0, ..., 0]

BOW로 표현하기

- Scikit-learn에서 제공하는 BOW 표현 라이브러리 활용
  - CountVectorizer
    - Term Frequency (TF)
    - 문서 집합으로부터 단어의 수를 세어 카운트 행렬을 생성
  - TfidfVectorizer
    - Term Frequency - Inverse Document Frequency (TF-IDF)
    - 문서 집합으로부터 단어의 수를 세고 TF-IDF 방식으로 단어의 가중치를 조정한 카운트 matrix를 생성

어휘사전 생성 및 확인

In [9]:
```
from sklearn.feature_extraction.text import CountVectorizer
bards_words = ["The fool doth think he is wise,",
               "but the wise man knows himself to be a fool"]

# CountVectorizer 객체 생성
vect = CountVectorizer()

# 문자열 리스트를 token으로 분리하고, 어휘사전을 구축
vect.fit(bards_words)

# 어휘사전은 vocabulary_ 속성에 저장됨
print("어휘사전의 크기: {}".format(len(vect.vocabulary_)))
print("어휘사전의 내용:\n{}".format(vect.vocabulary_))
```

```
어휘사전의 크기: 13
어휘사전의 내용:
{'the': 9, 'fool': 3, 'doth': 2, 'think': 10, 'he': 4, 'is': 6, 'wise':
12, 'but': 1, 'man': 8, 'knows': 7, 'himself': 5, 'to': 11, 'be': 0}
```

BOW 생성

- 입력 데이터에 대한 BOW는 transform method를 이용

```
bag_of_words = vect.transform(bards_words)

# 대부분 0이 되므로 sparse matrix로 표현됨
print("BOW: {}".format(repr(bag_of_words)), "\n")
print(bag_of_words, "\n")

# numpy array로 변환해야 display가 가능함
# 다만, 데이터 사이즈가 큰 경우 메모리 overflow 이슈가 발생할 수 있음
# Sparse matrix의 경우, 압축된 표현 방식을 사용하므로 메모리 이슈 생길 가능성이 적어짐
print("BOW의 밀집 표현: \n{}".format(bag_of_words.toarray()))
```

```
BOW: <2x13 sparse matrix of type '<class 'numpy.int64'>'
        with 16 stored elements in Compressed Sparse Row format>

  (0, 2)        1
  (0, 3)        1
  (0, 4)        1
  (0, 6)        1
  (0, 9)        1
  (0, 10)       1
  (0, 12)       1
  (1, 0)        1
  (1, 1)        1
  (1, 3)        1
  (1, 5)        1
  (1, 7)        1
  (1, 8)        1
  (1, 9)        1
  (1, 11)       1
  (1, 12)       1

BOW의 밀집 표현:
[[0 0 1 1 1 0 1 0 0 1 1 0 1]
 [1 1 0 1 0 1 0 1 1 1 0 1 1]]
```

"The fool doth think he is wise,"

"but the wise man knows himself to be a fool"

영화 리뷰에 대한 BOW

- 학습 데이터의 BOW 생성 및 특성 출력

In [11]:
```
# 영화 리뷰 학습 데이터데 대한 BOW 생성
vect = CountVectorizer().fit(text_train)
X_train = vect.transform(text_train)
print("X_train:\n{}".format(repr(X_train)))
```

```
X_train:
<25000x74849 sparse matrix of type '<class 'numpy.int64'>'
        with 3431196 stored elements in Compressed Sparse Row format>
```

In [12]:
```
#  get_feature_names: 특징(term)을 추출하여 리스트를 구성하는 method
feature_names = vect.get_feature_names()
print("특성 개수:{}".format(len(feature_names)))
print("첫 20개의 특성:\n{}".format(feature_names[:20]))
```

```
특성 개수:74849
첫 20개의 특성:
['00', '000', '0000000000001', '00001', '00015', '000s', '001', '00383
0', '006', '007', '0079', '0080', '0083', '0093638', '00am', '00pm', '0
0s', '01', '01pm', '02']
```

영화 리뷰에 대한 BOW

- 학습 데이터의 BOW 생성 및 특성 출력

In [13]:
```
print("20010에서 20030까지의 특성:\n{}".format(feature_names[20010:20030]))
print("매 2000번째 특성:\n{}".format(feature_names[::2000]))
```

```
20010에서 20030까지의 특성:
['dratted', 'draub', 'draught', 'draughts', 'draughtswoman', 'draw', 'd
rawback', 'drawbacks', 'drawer', 'drawers', 'drawing', 'drawings', 'dra
wl', 'drawled', 'drawling', 'drawn', 'draws', 'draza', 'dre', 'drea']
매 2000번째 특성:
['00', 'aesir', 'aquarian', 'barking', 'blustering', 'bête', 'chicaner
y', 'condensing', 'cunning', 'detox', 'draper', 'enshrined', 'favorit',
'freezer', 'goldman', 'hasan', 'huitieme', 'intelligible', 'kantrowit
z', 'lawful', 'maars', 'megalunged', 'mostey', 'norrland', 'padilla',
'pincher', 'promisingly', 'receptionist', 'rivals', 'schnaas', 'shunnin
g', 'sparse', 'subset', 'temptations', 'treatises', 'unproven', 'walkma
n', 'xylophonist']
```

불용어 제거

- 불용어(stopwords)
  - 의미없는 단어(eg. “the”, “of”, “all” 등)
  - 불용어로 인해 분류가 정확히 되지 않을 수 있으므로 제거
  - Scikit-learn에서는 feature_extraction.text에 불용어 목록을 갖고 있음

```
In [16]: from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
         print("불용어 개수: {}".format(len(ENGLISH_STOP_WORDS)))

         # ENGLISH_STOP_WORDS는 frozenset이라는 클래스임. 그러므로 리스트로 변환
         print("불용어 일부:\n{}".format(list(ENGLISH_STOP_WORDS)[:20]))
```

```
불용어 개수: 318
불용어 일부:
['them', 'therein', 'system', 'still', 'well', 'around', 'from', 'onl
y', 'third', 'than', 'the', 'less', 'found', 'are', 'becoming', 'detai
l', 'via', 'towards', 'between', 'eg']
```

불용어 제거

In [20]:
```
# stop_word = "english": builtin 불용어를 사용함
# built-in 불용어에 추가할 수도 있고, 자신만의 목록을 사용할 수도 있음
vect = CountVectorizer(min_df = 5, stop_words = "english").fit(text_train
X_train = vect.transform(text_train)
print("불용어가 제거된 X_train: \n{}".format(repr(X_train)))
```

```
불용어가 제거된 X_train:
<25000x26966 sparse matrix of type '<class 'numpy.int64'>'
        with 2149958 stored elements in Compressed Sparse Row format>
```

TF-IDF(Term Frequency - Inverse Document Frequency)

- 단어를 갯수 그대로 카운트하지 않고 모든 문서에 공통적으로 들어있는 단어의 경우 문서 구별 능력이 떨어진다고 보아 가중치를 축소하는 방법

For a term i in document j:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Zipf's Law

- Zipf's law
  - The Rank-Frequency Law of Zipf
    Frequency * Rank = Constant

- Frequency(빈도수) Rank (중요도)는 반비례?

  Frequency ↑ * Rank ↓ = 항상 일정

  Frequency ↓ * Rank ↑ = 항상 일정

정보검색 시스템(Information Retrieval System)

- Search Engine(검색엔진)에서 어떤 document의 중요도?
  ≅ 질의어와 관련 있는 document의 상대적인 부합정도
  ≅ (Relevance) Ranking
  ≅ 질의어와 얼마나 유사한지?
  ≅ Term Frequency * Inverse Document Frequency

- IDF(Inverse Document Frequency)
  - DF(Document Frequency): "트와이스"가 나오는 document의 수
  - IDF: 전체 document 건수의 역을 취한 값
  - 역이 아니면: DF/N (N은 전체 document의 건수)
  - 역을 취하면: N/DF

- 도대체 왜?
  - Zipf's law!

TF * IDF

- (Relevance) Ranking = TF * IDF
  = TF * (N/DF)
  ≅ TF * [Log(N/DF)]
  ≅ TF * [Log(N/DF)+1]
  $\cong \alpha\, TF * \beta\, [Log_{\gamma}\left(\frac{N}{DF}\right) + 1]$

$$\boldsymbol{Rank \cong \alpha\, TF * \beta\, [Log_{\gamma} N - Log_{\gamma}\, DF + 1]}$$

- Tuning parameters
  - α : TF를 전체 중의 얼마로?
  - β : DF의 차별성은 전체 중의 얼마로?
  - γ : IDF 값에 따라서 결정?
  - $\alpha + \beta = 1,\ 0 \sim 1$ 값으로 ranking 값을 정규화?

TF-IDF(Term Frequency - Inverse Document Frequency)

In [24]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
corpus = [
    'This is the first document.',
    'This is the second second document.',
    'And the third one.',
    'Is this the first document?']

vect1 = CountVectorizer().fit(corpus)
tf = vect1.transform(corpus)

feature_names = vect1.get_feature_names()
print("Term:{}".format(feature_names[:]))
print(tf.toarray(), "\n")

vect2 = TfidfVectorizer().fit(corpus)
tfidf = vect2.transform(corpus)
print(tfidf.toarray())
```

```
Term:['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
[[0 1 1 1 0 0 1 0 1]
 [0 1 0 1 0 2 1 0 1]
 [1 0 0 0 1 0 1 1 0]
 [0 1 1 1 0 0 1 0 1]]

[[0.         0.43877674 0.54197657 0.43877674 0.         0.
  0.35872874 0.         0.43877674]
 [0.         0.27230147 0.         0.27230147 0.         0.85322574
  0.22262429 0.         0.27230147]
 [0.55280532 0.         0.         0.         0.55280532 0.
  0.28847675 0.55280532 0.        ]
 [0.         0.43877674 0.54197657 0.43877674 0.         0.
  0.35872874 0.         0.43877674]]
```

단어의 쌍으로 구성하는 n-gram 방식

- 단어 쌍으로 표현하면 단어의 순서나 결합하는 단어의 의미를 표현할 수 있음
- 하나의 단어만으로 표현한 경우

```
from sklearn.feature_extraction.text import CountVectorizer

bards_words = ["The fool doth think he is wise,",
               "but the wise man knows himself to be a fool"]


# (1, 1) min, max gram(term)수
cv = CountVectorizer(ngram_range = (1, 1)).fit(bards_words)
print("어휘사전의 크기: {}".format(len(cv.vocabulary_)))
print("어휘사전:\n{}".format(cv.get_feature_names()))
print("변환된 데이터:\n{}".format(cv.transform(bards_words).toarray()))
```

```
어휘사전의 크기: 13
어휘사전:
['be', 'but', 'doth', 'fool', 'he', 'himself', 'is', 'knows', 'man', 'the', 'think', 'to', 'w
ise']
변환된 데이터:
[[0 0 1 1 1 0 1 0 0 1 1 0 1]
 [1 1 0 1 0 1 0 1 1 1 0 1 1]]
```

단어의 쌍으로 구성하는 n-gram 방식

- Bigram

```
cv = CountVectorizer(ngram_range = (2, 2)).fit(bards_words)
print("어휘사전의 크기: {}".format(len(cv.vocabulary_)))
print("어휘사전:\n{}".format(cv.get_feature_names()))
print("변환된 데이터:\n{}".format(cv.transform(bards_words).toarray()))
```

```
어휘사전의 크기: 14
어휘사전:
['be fool', 'but the', 'doth think', 'fool doth', 'he is', 'himself to', 'is wise', 'knows hi
mself', 'man knows', 'the fool', 'the wise', 'think he', 'to be', 'wise man']
변환된 데이터:
[[0 0 1 1 1 0 1 0 0 1 0 1 0 0]
 [1 1 0 0 0 1 0 1 1 0 1 0 1 1]]
```

단어의 쌍으로 구성하는 n-gram 방식

- 1-gram부터 trigram까지 모두 포함

```
cv = CountVectorizer(ngram_range = (1, 3)).fit(bards_words)
print("어휘사전의 크기: {}".format(len(cv.vocabulary_)))
print("어휘사전:\n{}".format(cv.get_feature_names()))
print("변환된 데이터:\n{}".format(cv.transform(bards_words).toarray()))
```

```
어휘사전의 크기: 39
어휘사전:
['be', 'be fool', 'but', 'but the', 'but the wise', 'doth', 'doth thin
k', 'doth think he', 'fool', 'fool doth', 'fool doth think', 'he', 'he
is', 'he is wise', 'himself', 'himself to', 'himself to be', 'is', 'is
wise', 'knows', 'knows himself', 'knows himself to', 'man', 'man know
s', 'man knows himself', 'the', 'the fool', 'the fool doth', 'the wis
e', 'the wise man', 'think', 'think he', 'think he is', 'to', 'to be',
'to be fool', 'wise', 'wise man', 'wise man knows']
변환된 데이터:
[[0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0
0 0
  1 0 0]
 [1 1 1 1 1 0 0 0 1 0 0 0 0 0 1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 0 0 0 1
1 1
  1 1 1]]
```

어간 추출-stemming

- 어간(stem)
  - 활용어가 활용할 때 변하지 않는 부분
  - 예)
    - '보다', '보니', '보고' -> '보'
    - '먹다', '먹은', '먹고' -> '먹'
    - 'drawer', 'drawing' -> 'draw'
- 어간 추출(stemming)
  - 어미를 제외하고 어간만을 추출하는 방법

- 참고
  - 고급 토큰화의 다른 방법(표제어 추출)
    - 사전을 활용하여 표준말을 추출하는 방법(먹다, 보다 등)

어간 추출에 필요한 패키지 설치

- nltk()
  - Natural Language Toolkit
  - a platform for building Python programs to work with human language data

- 설치
  - pip install nltk
    - http://www.nltk.org/
  - 다음 코드를 수행하여 사전을 다운로드(시간이 다소 소요)

```
In [*]: import nltk
        nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

Stemming test

- 단어별 stemming

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

ps = PorterStemmer() # stemmer 객체생성

example_words = ["python", "pythoner", "pythoning", "pythoned", "pythonly"]
for stem in example_words:
    print(ps.stem(stem)) # stem method로 term별 stemming 실행
```

```
python
python
python
python
pythonli
```

Stemming test

- 문장 전체를 stemming

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

new_text = "It is important to be very pythonly while you are pythoning

words = word_tokenize(new_text)
print(words)

for w in words:
    print(ps.stem(w))
```

```
['It', 'is', 'important', 'to', 'be', 'very', 'pythonly', 'while', 'yo
u', 'are', 'pythoning', 'with', 'python']
It
is
import
to
be
veri
pythonli
while
you
are
python
with
python
```

Stemming test

```
# 화면에 출력하지 않고 term 리스트로 유지
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

new_text = "It is important to be very pythonly while you are pythoning

words = word_tokenize(new_text)
result = [ps.stem(w) for w in words]
print(result)
```

```
['It', 'is', 'import', 'to', 'be', 'veri', 'pythonli', 'while', 'you',
'are', 'python', 'with', 'python']
```

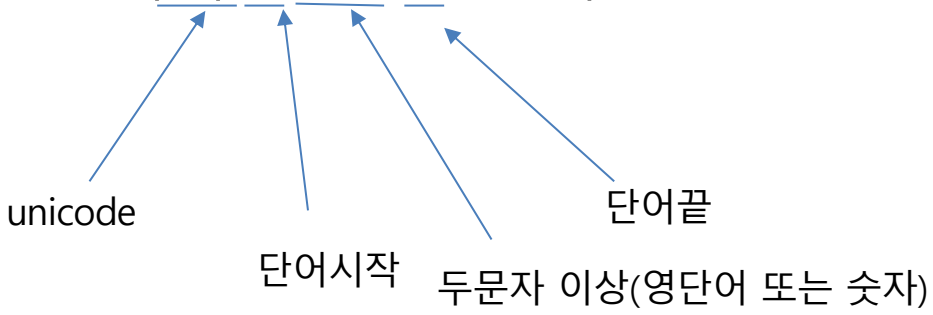
Stemming 기능을 추가한 BOW 생성

```
import nltk
from nltk import word_tokenize
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer

stemmer = PorterStemmer()

def tokenize(text):
    tokens = nltk.word_tokenize(text)
    stemmed = []
    for item in tokens:
        stemmed.append(stemmer.stem(item))
    return stemmed

# CountVectorizer에 토큰을 생성하는 별도의 함수를 지정함
# 함수 내부에서 토큰화와 stemming을 실행
vect = CountVectorizer(tokenizer = tokenize, stop_words = 'english')

vect.fit(["The swimmer likes swimming."])

sentence1 = vect.transform(['The swimmer likes swimming.'])
sentence2 = vect.transform(['The swimmer swims. .'])

print(vect.get_feature_names())
print(sentence1.toarray()) # sentence1과 sentence2는 sparse matrix
print(sentence2.toarray())
```

```
['.', 'like', 'swim', 'swimmer']
[[1 1 1 1]]
[[2 0 1 1]]
```

영문자만을 추출(숫자 및 구둣점 제거)

- 구두점 및 하나의 문자로 구성된 단어 제거
  - CountVectorizer(token_pattern =u'(?u)₩b₩w₩w+₩b')



- 구두점 제거 대안

```
import re
# Use regular expressions to do a find-and-replace
review_text = re.sub("[^a-zA-Z]", " ", review_text)
```

Full code

In [1]:
```
from sklearn.datasets import load_files
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
import nltk
from nltk import word_tokenize
from nltk.stem.porter import PorterStemmer
import numpy as np
import re

reviews_train = load_files("/Users/jonghoonchun/Downloads/aclImdb 2/train")
text_train, y_train = reviews_train.data, reviews_train.target

reviews_test = load_files("/Users/jonghoonchun/Downloads/aclImdb 2/test")
text_test, y_test = reviews_test.data, reviews_test.target
print("테스트 데이터의 문서 수: {}".format(len(text_test)))
print("클래스별 샘플 수 (테스트 데이터): {}".format(np.bincount(y_test)))

text_train = [doc.replace(b"<br /n>", b" ") for doc in text_train]
text_test = [doc.replace(b"<br /n>", b" ") for doc in text_test]

stemmer = PorterStemmer()

def tokenize(text):
    tokens = nltk.word_tokenize(text)
    stemmed = []
    for item in tokens:
        stemmed.append(stemmer.stem(item))
    return stemmed
```

실행시간이 다소 소요됨

Full code

```
vect = CountVectorizer(tokenizer = tokenize, stop_words = 'english').fit(text_train)
X_train = vect.transform(text_train)
X_test = vect.transform(text_test)

#logistic reression을 이용
clf = LogisticRegression()
clf.fit(X_train, y_train)
pre = clf.predict(X_test)

#정답률 구하기
ac_score = accuracy_score(y_test, pre)
print("정답률 = ", ac_score)
```

```
테스트 데이터의 문서 수: 25000
클래스별 샘플 수 (테스트 데이터): [12500 12500]
정답률 =  0.86032
```

END