Sequence labeling: POS Tagger
(+ NLTK Practice)

Knowledge & Language Engineering Lab.

POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Contents

- Sequence Labeling
  - Introduction
  - Method
- Practice
  - Simple POS tagger using HMM Algorithm
  - Natural Language ToolKit (Open source Platform)

[Relation Extraction]

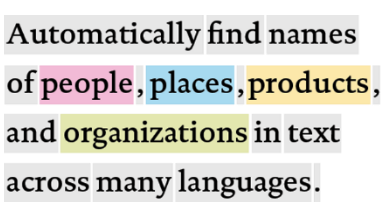
SEQUENCE LABELING

Introduction

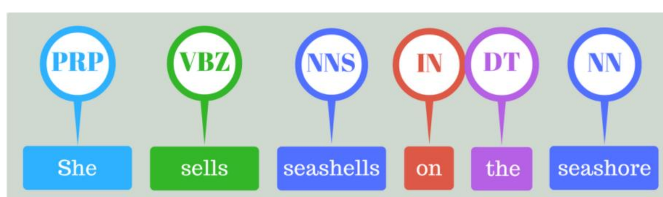
- Sequence labeling
  - A pattern recognition task that classifies a categorical label to each member of a sequence elements.
  - In NLP, which deals with sequential data, sequence labeling is one of the major task.

- Tasks or subtasks

Named entity recognition



Part of speech tagging



Spacing problem

아버지가방에들어가신다.

↓

아버지가 방에 들어가신다.

Introduction

- Sequential Data
  - Data stored in chronological order.
  - Generally, each element is related to each other.
  - E.g.)
    - Video: a sequence of frames
    - Text: a sequence of words
    - Voice: a sequence of signals

Methods

- Sequence labeling methods
  - Vector space model
    - Neural network model
    - Structured SVM
  - Probabilistic model
    - Hidden Markov Model (HMM)
    - Conditional Random Field (CRF)

Methods

- Sequence labeling methods
  - Vector space model
    - Neural network model
    - Structured SVM
  - Probabilistic model
    - **Hidden Markov Model (HMM)**
    - Conditional Random Field (CRF)

Hidden Markov Model

- $y^*_{1:N} = argmax_{y_{1:N}} P(y_{1:N}|x_{1:N})$

  $(Bayes\ rule)$

  $= argmax_{y_{1:N}} P(x_{1:N}|y_{1:N})P(y_{1:N})$

  $= argmax_{y_{1:N}} \prod_{k=1}^{N} P(x_k|x_{1:k-1}, y_{1:K}) \prod_{k=1}^{N} P(y_k|y_{1:k-1})$

  $(Markov\ assumption)$

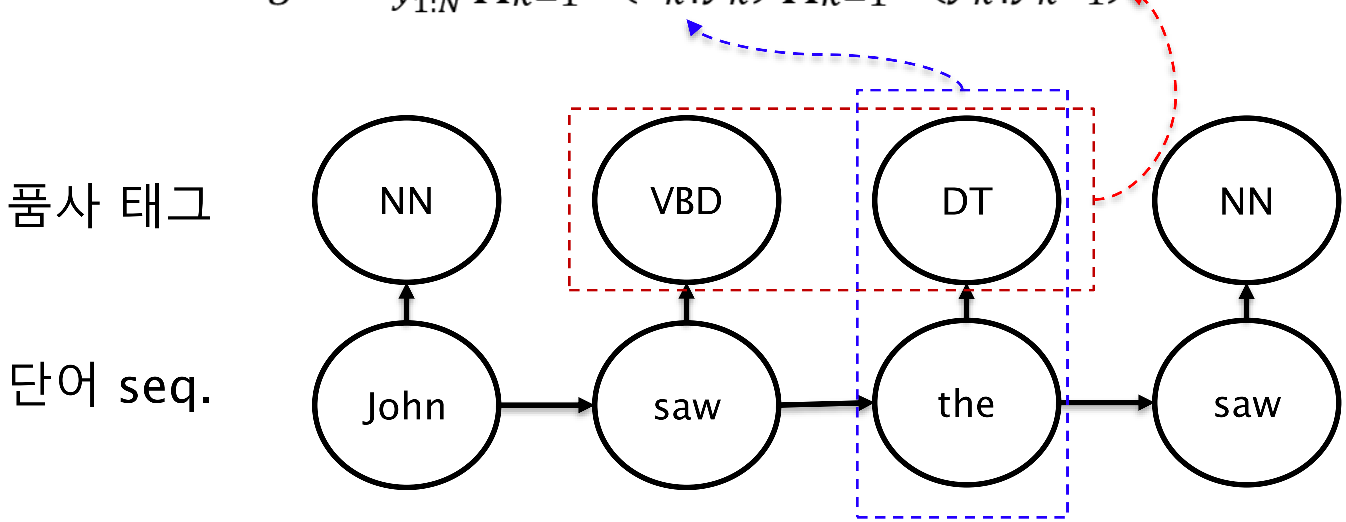
  $\approx argmax_{y_{1:N}} \prod_{k=1}^{N} P(x_k|y_k) \prod_{k=1}^{N} P(y_k|y_{k-1})$

Hidden Markov Model

- $y^*_{1:N} = argmax_{y_{1:N}} P(y_{1:N}|x_{1:N})$

  $(Bayes\ rule)$
  $= argmax_{y_{1:N}} P(x_{1:N}|y_{1:N})P(y_{1:N})$
  $= argmax_{y_{1:N}} \prod_{k=1}^{N} P(x_k|x_{1:k-1}, y_{1:N}) \prod_{k=1}^{N} P(y_k|y_{1:k-1})$

  $(Markov\ assumption)$
  $\approx argmax_{y_{1:N}} \prod_{k=1}^{N} P(x_k|y_k) \prod_{k=1}^{N} P(y_k|y_{k-1})$



Hidden Markov Model

- $argmax_{y_{1:N}} \prod_{k=1}^{N} P(x_k|y_k) \prod_{k=1}^{N} P(y_k|y_{k-1})$
  - $P(x_k|y_k)$: emission probability
    - 각 state(y) 에서 관측 가능한 값(x)의 확률
    - E.g.) 명사(NN) 인 'saw' 가 등장할 확률
    - $P(x_k|y_k) = \frac{P(x_k, y_k)}{P(y_k)}$
  - $P(y_k|y_{k-1})$: transition probability
    - State(y) 간의 변화 확률
    - E.g.) 동사(VB) 이후에 명사(NN)가 등장할 확률
    - $P(y_k|y_{k-1}) = \frac{P(y_k, y_{k-1})}{P(y_{k-1})}$

Hidden Markov Model

- $\log(P(\text{NN VBD DT NN}|\text{John saw the saw}\ )$

$$= \log P(Jone|NN) + \log P(NN|<BOS>)$$

$$+ \log P(saw|VBD) + \log P(VBD|NN)$$

$$+ \log P(the|DT) + \log P(DT|VBD)$$

$$+ \log P(saw|NN) + \log P(NN|DT)$$

$$+ \log P(<EOS>|NN)$$

PRACTICE

KLE tagset

- 부가자료: KLE_Tagset.pdf 파일 참고

preprocessing

- Preprocess each line with a list of tuples.
  - $[[(word_1, tag_1), (word_2, tag_2), ..., (word_n, tag_n),$
    $[(word_1, tag_1), (word_2, tag_2), ..., (word_n, tag_n)$
    $\vdots$
    $[(word_1, tag_1), (word_2, tag_2), ..., (word_n, tag_n)]]$

그/CT 도/fjb 강하/YBH ㄴ/fmotg 카리스마/CMC 를/fjco 필요/CMC 하/fph ㅂ니다/fmof ./g
애플/CMC 이/fjcs 80/CS %/g 로/fjcao 그/SG 뒤/CMC 를/fjco 쫓/YBD 았/fmb 습니다/fmof ./g
이제/SBO 참가자들/CMC 이/fjcs 기념촬영/CMC 을/fjco 하/YBD 고/fmoc 있/YA 다/fmof ./g

↓

[[(그, CT), (도, fjb), (강하, YBH), ..., (ㅂ니다, fmof), (., g)],
[(애플, CMC), (이, fjcs), (80, CS), …., (습니다, fmof), (., g)],
[(이제, SBO), (참가자들, CMC), (이, fjcs) ,…., (다, fmof), (., g)]]

Train function

- Count the number of (word, tag)
  - Nested dictionary type
    - pos2words_freq = defaultdict(lambda: defaultdict(int))
    - Pos2words[pos][word] _freq:
      - stores the number (frequency) of (word, tag)

- Count the number of bigram tags (tag_{i-1}, tag_i)
  - Dictionary type
    - Define trans_freq = defaultdict(int) for bigrams counts
    - Define bos_freq = defaultdict(int) for the bigrams counts containing "BOS"
      - Trans[(tag_{i-1}, tag_i)] stores the number of bigrams
      - Bos[tag_i] stores the number of BOS bigrams

Train function

- Example
  - pos2words_freq

    {CMC: {아버지: 10, 올림픽: 15, ..},
    CMP: {구글: 20, 애플: 15, ..}
    YBD: {마시: 10, 듣: 20, ...}}

  - trans_freq

    {(CMC, fjb): 20, (CMP, fjb): 31, (fjco, fd): 55, ..}

  - bos_freq

    {CMP: 100, CMC: 200, CT: 55, ...}

Train function

- Frequency -> probability
  - pos2words_prob

sum = 1.0

```
{CMC: {아버지: 0.1, 올림픽: 0.2, ..},
 CMP: {구글: 0.05, 애플: 0.03, ..}
 YBD: {마시: 0.1, 듣: 0.2, ...}}
```

  - trans_prob

```
{(CMC, fjb): 0.2, (CMP, fjb): 0.1, (fjco, fd): 0.48, ..}
```

  - bos_prob

```
{CMP: 0.3, CMC: 0.1, CT: 0.24, ...}
```

Train function

- Frequency -> probability
  - pos2words_prob

```
{CMC: {아버지: 0.1, 올림픽: 0.2, ..},
 CMP: {구글: 0.05, 애플: 0.03, ..}
 YBD: {마시: 0.1, 듣: 0.2, ...}}
```

sum = 1.0

$P(x_k = \text{애플} \mid y_k = \text{CMP}) = 0.03$

  - trans_prob

```
{(CMC, fjb): 0.2, (CMP, fjb): 0.1, (fjco, fd): 0.48, ..}
```

  - bos_prob

```
{CMP: 0.3, CMC: 0.1, CT: 0.24, ...}
```

Train function

- Frequency -> probability
  - pos2words_prob

```
{CMC: {아버지: 0.1, 올림픽: 0.2, ..},
 CMP: {구글: 0.05, 애플: 0.03, ..}
 YBD: {마시: 0.1, 듣: 0.2, ...}}
```

sum = 1.0

$P(x_k = \text{애플} \mid y_k = \text{CMP}) = 0.03$

  - trans_prob

```
{(CMC, fjb): 0.2, (CMP, fjb): 0.1, (fjco, fd): 0.48, ..}
```

$P(y_k = \text{fd} \mid y_{k-1} = \text{fjco}) = 0.48$

  - bos_prob

```
{CMP: 0.3, CMC: 0.1, CT: 0.24, ...}
```

Train function

- Emission probability
  - $P(x_k|y_k) = \frac{P(x_k,y_k)}{P(y_k)} = \frac{\#\ of\ (word_k,tag_k)}{\#\ of\ tag_k}$

- Transition probability
  - $P(y_k|y_{k-1}) = \frac{P(y_k,y_{k-1})}{P(y_{k-1})} = \frac{\#\ of\ (tag_{k-1},tag_k)}{\#\ of\ tag_{k-1}}$

Inference

- For given input sentences
  - "감기/CMC 는/fjb 줄이/YBD 다/fmof ./g“
  - "감기/fmotg 는/fjb 줄/CMC 이다/fjj ./g“

- Calculate the log probability
  - $\log(\prod_{k=1}^{N} P(x_k|y_k) \prod_{k=1}^{N} P(y_k|y_{k-1}))$

$$= \sum \log P(x_k|y_k) + \log P(y_k|y_{k-1})$$

- Results

```
감기/CMC 는/fjb 줄이/YBD 다/fmof ./g: -5.489636
감기/fmotg 는/fjb 줄/CMC 이다/fjj ./g: -14.037157
```

NLTK (Open Source Platform)

Natural Language ToolKit (NLTK)

- NLTK
  - Python Platform for Natural Language Processing
  - Homepage
    - http://www.nltk.org/
  - 설치 방법
    - sudo pip install -U nltk

Natural Language ToolKit (NLTK)

- NLTK.tagger
  - Example

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged[0:6]
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
('Thursday', 'NNP'), ('morning', 'NN')]
```

NLTK download

- $python
- import nltk
- nltk.download()

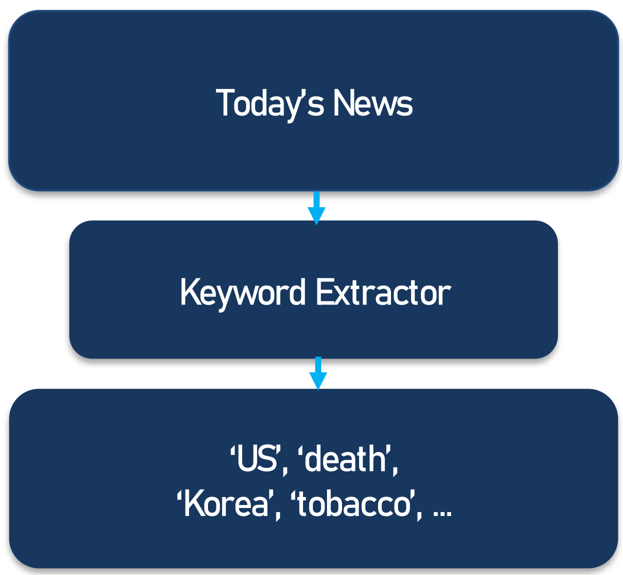
```
>>> import nltk
>>> nltk.download()
NLTK Downloader
---------------------------------------------------------------------------
    d) Download   l) List    u) Update   c) Config   h) Help   q) Quit
---------------------------------------------------------------------------
Downloader> d

Download which package (l=list; x=cancel)?
  Identifier> popular
```

Keyword Extractor

- Keyword Extractor
  - 주어진 문서(document)의 키워드 10개를 추출

- Keyword
  - def. 최대 빈도를 가지는 명사



Keyword Extractor

- Preparation Material
  - 말뭉치 다운로드
  - 말뭉치: endoc1~8.txt

Keyword Extractor

- Guideline
  - Step1. 텍스트 입력 및 형태소 분석
  - Step2. 명사 추출 및 해당 빈도수 저장
  - Step3. 빈도수로 내림차순 정렬
  - Step4. 최대 빈도수 단어 출력

Keyword Extractor

- Step1. 텍스트 입력 및 형태소 분석
  - 텍스트파일을 입력 받음
  - 입력을 tokenizing
  - 각 token의 형태소 분석

Keyword Extractor

- os.listdir(path)

```
pirl@pirl-Precision-Tower-7910:~/NLP/3PosTagger/Practice$ ls Data/
endoc1.txt  endoc3.txt  endoc5.txt  endoc7.txt
endoc2.txt  endoc4.txt  endoc6.txt  endoc8.txt
pirl@pirl-Precision-Tower-7910:~/NLP/3PosTagger/Practice$ python3
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 18:10:19)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from os import listdir
>>> listdir("./Data")
['endoc1.txt', 'endoc5.txt', 'endoc7.txt', 'endoc3.txt', 'endoc8.txt',
'endoc6.txt', 'endoc2.txt', 'endoc4.txt']
>>> 
```

Keyword Extractor

- Step2. 일반 명사 추출 및 해당 빈도수 저장
  - 형태소 분석 및 품사 태깅된 텍스트에서 명사를 추출
  - dictionary에 단어-빈도수 형태로 저장
  - Nouns = ['NN', 'NNS', 'NNP', 'NNPS']

Document Finder

- Non-alphabet

```
endoc1.txt: China, US, South, Sea, Beijing, systems, mi
ssile, defense, missiles, region
endoc2.txt: tobacco, San, Francisco, products, voters,
American, Reynolds, Association, sales, %
endoc3.txt: death, method, execution, injection, victim
, people, seconds, –, man, prison
endoc4.txt: Russia, North, Korea, Kim, Moscow, Lavrov,
Korean, US, talks, South
endoc5.txt: Iran, EU, sanctions, deal, US, business, co
mpanies, European, EIB, legislation
endoc6.txt: people, racist, racism, police, part, ideas
, Americans, Obama, Goff, America
endoc7.txt: Kennedy, Lewis, years, Robert, America, Cli
nton, way, hand, John, assassination
endoc8.txt: US, Afghan, Afghanistan, air, Taliban, Air,
 Force, people, casualties, children
```

Document Finder

- Free Non-alphabet words

```
endoc1.txt: China, US, South, Sea, Beijing, systems, mi
ssile, defense, missiles, region
endoc2.txt: tobacco, San, Francisco, products, voters,
American, Reynolds, Association, sales, Proposition
endoc3.txt: death, method, execution, injection, victim
, people, seconds, man, prison, chair
endoc4.txt: Russia, North, Korea, Kim, Moscow, Lavrov,
Korean, US, talks, South
endoc5.txt: Iran, EU, sanctions, deal, US, business, co
mpanies, European, EIB, legislation
endoc6.txt: people, racist, racism, police, part, ideas
, Americans, Obama, Goff, America
endoc7.txt: Kennedy, Lewis, years, Robert, America, Cli
nton, way, hand, John, assassination
endoc8.txt: US, Afghan, Afghanistan, air, Taliban, Air,
 Force, people, casualties, children
```

Document Finder

- str.isalpha()

```
str = "this";  # No space & digit in this string
print str.isalpha()

str = "this is string example....wow!!!";
print str.isalpha()
```

```
True

False
```

```
for t in tags:
        if t[1] in Nouns:

        if t[1] in Nouns and t[0].isalpha():
```

Keyword Extractor

- 형태소 분석 및 품사 태깅된 텍스트

- nltk.word_tokenize
- nltk.pos_tag

```
>>> text = word_tokenize("And now for something completely different")
>>> nltk.pos_tag(text)
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'),
('completely', 'RB'), ('different', 'JJ')]
```

Keyword Extractor

- 명사 추출
- dictionary에 단어(명사)-빈도수 형태로 저장

Keyword Extractor

- Step3. 빈도수로 내림차순 정렬
  - google “sort a Python dictionary by value”!
  - 여기서 value=빈도수, 대응하는 key값은 단어

Keyword Extractor

- sorted()
- sort a dictionary by value

```
>>> from operator import itemgetter
>>> dict = {}
>>> dict['a'] = 2
>>> dict['b'] = 1
>>> dict['c'] = 5
>>> print(sorted(dict.items(), key=itemgetter(1), reverse=True))
[('c', 5), ('a', 2), ('b', 1)]
```

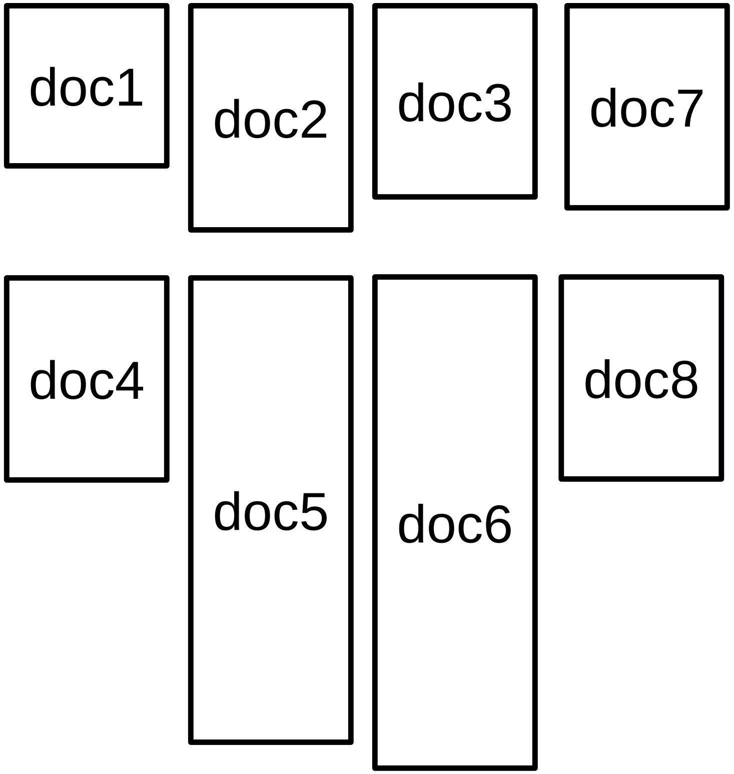
Keyword Extractor

- Step4. 최대 빈도수 단어 출력
  - 정렬한 리스트로부터 top 10개의 단어를 출력

Keyword Extractor

- 결과물

[('US', 42),
('people', 29),
('Russia', 26),
('China', 19),
('Iran', 18),
('North', 18),
('Kennedy', 17),
('years', 16),
('Korea', 16),
('Afghan', 16)]



각 뉴스 기사의 길이(# tokens) 반영

Keyword Extractor

- 각 document의 길이를 반영
- Normalized count:

  1.0 / document_length (# of words)

- word count 대신 normalized count를 dictionary에 저장

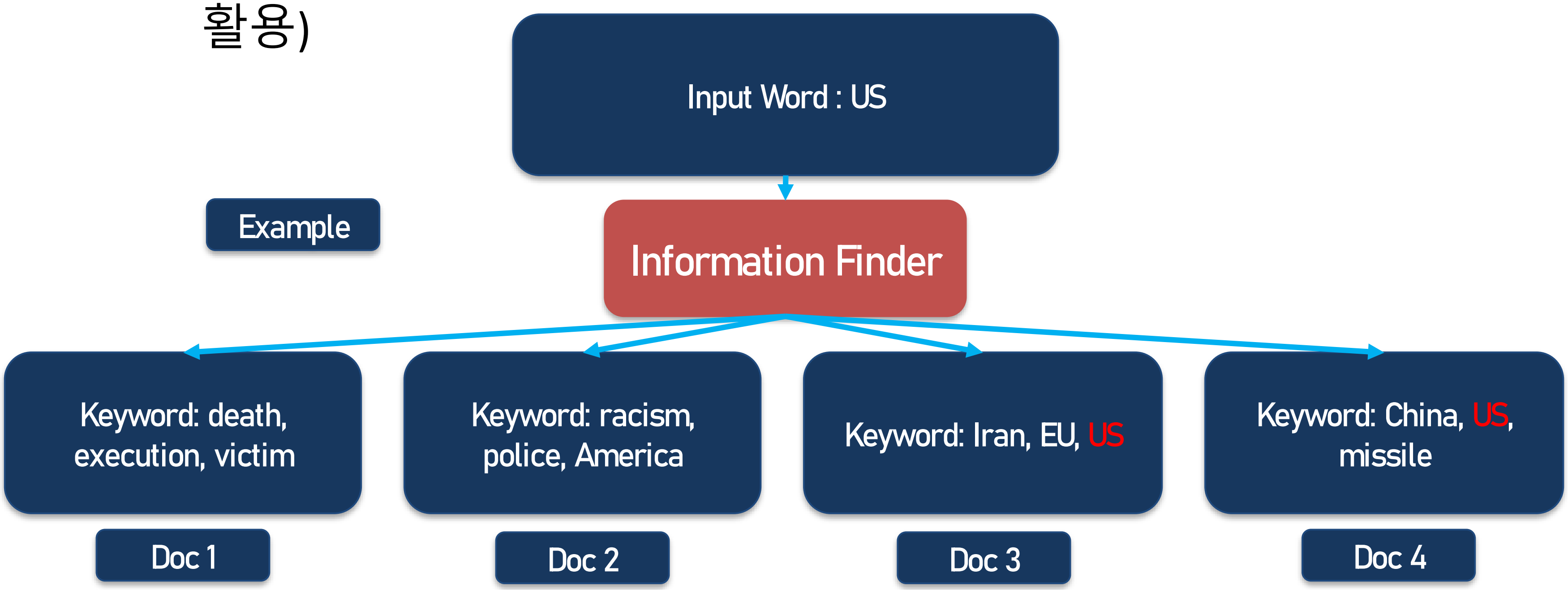
Keyword Extractor

- 결과물

```
[('US', 0.04148232756560206 5),
('Iran', 0.02808112324492979 8),
('China', 0.02634082488749564 5),
('Russia', 0.02146452100122221),
('tobacco', 0.01973684210526315 7),
('people', 0.01909152551826487 4),
('EU', 0.0171606864274571),
('Kennedy', 0.01480836236933797 7),
('South', 0.01420072105140088 3),
('North', 0.01393825765358536 9)]
```

Document Finder

- Document Finder
  - 입력된 단어와 가장 관련된 문서를 출력
- 가장 관련된 문서
  - Def. Keyword가 해당 단어와 일치하는 문서 (Keyword Extractor 활용)



Document Finder

- Guideline
  - Step 1. 텍스트 입력 및 형태소 분석
  - Step 2. 명사 추출 및 빈도수로 내림차순 정렬
  - Step 3. 최대 빈도를 가지는 10 개의 명사들 따로 저장
  - Step 4. 각 문서에 대해 Step 1~3 반복

==================================

  - Step 5. <키워드:해당 문서들> 의 형태로 dictionary 에 저장

==================================

  - Step 6. 사용자에게 키워드 입력 받음
  - Step 7. 입력받은 키워드가 dictionary에 있으면 해당 문서 출력

Document Finder

- Step 5. {키워드:해당 문서들} 의 형태로 dictionary 에 저장

```
dic={}
for intxt in files:
        keywords = keywords_per_doc[intxt]
```

Document Finder

- Step 6. 사용자에게 키워드 입력 받음
- input()

- 사용 예시:
- query = input().strip()

Document Finder

- Output

```
type keyword (q:to exit)
US
endoc1.txt, endoc4.txt, endoc5.txt, endoc8.txt
type keyword (q:to exit)
Korea
endoc4.txt
type keyword (q:to exit)
people
endoc3.txt, endoc6.txt, endoc8.txt
type keyword (q:to exit)
Lebanon
no such document
type keyword (q:to exit)
q
pirl@pirl-Precision-Tower-7910:~/NLP/3PosTagger/
```

END