Chapter. 03

자연어 분석 기초

# 정규표현식과 텍스트 전처리

FASTCAMPUS
ONLINE
금융공학/퀀트 I

강사. 장순용

# 키포인트

- 정규표현식.
- 메타 문자.
- 텍스트 데이터 전처리.

# I 정규표현식 (Regular Expression)이란?

- 문자열 패턴을 만드는데 사용.
- 복잡한 문자열을 처리할 때 필요함.
- Python, R 등 많은 프로그래밍 언어에서 지원.
- 일반 문자열 함수의 조합 보다 함축적이며 강력한 기능.

# I 메타 문자 (Meta Character)



- 메타문자는 원래 그 문자가 가진 뜻이 아닌 특별한 용도로 사용되는 문자이다.

. ^ $ * + ? { } [ ] \ | ( )

- 정규 표현식에서 메타 문자들은 특별한 의미를 갖게 된다.

# I 메타 문자 (Meta Character)



- 문자 클래스 [ ]

→ [와 ] 사이에는 어떤 문자도 들어갈 수 있다.

→ “[와 ] 사이의 문자들과 매치”라는 의미를 갖는다.

| 정규표현식 | 문자열 | Match 여부 | 설명 |
|---|---|---|---|
| [abc] | a | Yes | 정규식과 일치하는 문자인 “a”가 있으므로 매치. |
| [abc] | before | Yes | 정규식과 일치하는 문자인 “b”가 있으므로 매치. |
| [abc] | dude | No | “a”, “b”, “c” 중 어느 하나도 포함하고 있지 않음. |

# I 메타 문자 (Meta Character)



- 문자 클래스 [ ]

→ [ ] 안의 두 문자 사이에 하이픈 “-”을 사용하게 되면 범위를 의미한다.

예). [a-c]라는 정규 표현식은 [abc]와 같다.

예). [0-5]는 [012345]와 같다.

예). [a-zA-Z] : 알파벳 대소문자 전체와 같다.

예). [0-9] : 숫자.

# I 메타 문자 (Meta Character)



- 문자 클래스 [ ]

→ 또한 다음과 같은 **단축 표현**이 있다.

예). **w** = [a-zA-Z0-9_]

예). **W** = [^a-zA-Z0-9_]

예). **d** = [0-9]

예). **D** = [^0-9]

# I 메타 문자 (Meta Character)



- NOT 문자 클래스 [^ ]

→ “[^와 ] 사이의 문자들과 매치하지 않음”을 의미한다.

| 정규표현식 | 문자열 | Match 여부 | 설명 |
|---|---|---|---|
| [^abc] | a | No | “a”가 “abc”에 포함됨. |
| [^abc] | before | Yes | “b”를 제외한 나머지는 “abc”에 포함되지 않음. |
| [^abc] | dude | Yes | “a”, “b”, “c” 중 어느 하나도 포함하고 있지 않음. |

# I 메타 문자 (Meta Character)



- 점 .

→ 정규 표현식의 점 . 메타문자는 모든 문자와 매치됨.

→ 그런데 [ ] 안의 점 . 은 패턴이 아니라 문자 그대로를 의미한다.

| 정규표현식 | 문자열 | Match 여부 | 설명 |
|---|---|---|---|
| a.b | aab | Yes | 가운데 문자 “a”가 모든 문자를 의미하는 .과 일치. |
| a.b | a0b | Yes | 가운데 문자 “0”가 모든 문자를 의미하는 .과 일치. |
| a.b | abc | No | “a”문자와 “b”문자 사이에 어떤 문자도 없음. |

# I 메타 문자 (Meta Character)



- 반복 *

→ *는 바로 앞의 문자가 0회 부터 무한대의 회수로 반복되는 패턴을 의미한다.

| 정규표현식 | 문자열 | Match 여부 | 설명 |
|---|---|---|---|
| ca*t | ct | Yes | “a”가 0번 반복되어 매치. |
| ca*t | cat | Yes | “a”가 0번 이상 반복되어 매치 (한번 반복). |
| ca*t | caaat | Yes | “a”가 0번 이상 반복되어 매치 (세번 반복). |

# I 메타 문자 (Meta Character)



- 반복 +

→ +는 바로 앞의 문자가 1회 부터 무한대의 회수로 반복되는 패턴을 의미한다.

| 정규표현식 | 문자열 | Match 여부 | 설명 |
|---|---|---|---|
| ca+t | ct | No | “a”가 0번 반복되어 매치 아님. |
| ca+t | cat | Yes | “a”가 한번 이상 반복되어 매치 (한번 반복). |
| ca+t | caaat | Yes | “a”가 한번 이상 반복되어 매치 (세번 반복). |

# I 메타 문자 (Meta Character)



- 발생여부 ?

→ ?는 바로 앞의 문자가 0회 또는 1회 있는 패턴을 의미한다.

| 정규표현식 | 문자열 | Match 여부 | 설명 |
|---|---|---|---|
| ca?t | ct | Yes | “a”가 0번 반복됨. 매치 됨! |
| ca?t | cat | Yes | “a”가 한번 반복됨. 매치 됨! |
| ca?t | caat | No | “a”가 두번 반복됨. 매치 아님. |

# I 메타 문자 (Meta Character)



- 반복 {m}

→ {m}는 바로 앞의 문자가 m회 반복하는 패턴을 의미한다.

| 정규표현식 | 문자열 | Match 여부 | 설명 |
|---|---|---|---|
| ca{2}t | ct | No | “a”가 0번 반복됨. |
| ca{2}t | cat | No | “a”가 한번 반복됨. |
| ca{2}t | caat | Yes | “a”가 정확하게 두번 반복됨. 매치! |

# I 메타 문자 (Meta Character)



- 반복 {m,n}

→ {m,n}는 바로 앞의 문자가 m회~n회 반복하는 패턴을 의미한다.

| 정규표현식 | 문자열 | Match 여부 | 설명 |
|---|---|---|---|
| ca{2,5}t | cat | No | “a”가 1번 반복됨. 매치 아님. |
| ca{2,5}t | caaat | Yes | “a”가 3번 반복됨. 매치 됨! |
| ca{2,5}t | caaaaaat | No | “a”가 6번 반복됨. 매치 아님. |

# I 메타 문자 (Meta Character)



- 시작 일치 ^

→ ^는 문자열의 가장 처음과 일치함을 의미한다.

| 정규표현식 | 문자열 | Match 여부 | 설명 |
|---|---|---|---|
| ^Life | Life is short. | Yes | 문장의 첫 단어가 Life 이다. |
| ^Life | My Life is boring. | No | 문장의 첫 단어가 Life 아니다. |

# I 메타 문자 (Meta Character)



- 끝 일치 $

→ $는 문자열의 가장 마지막과 일치함을 의미한다.

| 정규표현식 | 문자열 | Match 여부 | 설명 |
|---|---|---|---|
| Python$ | Python is easy | No | 문장의 마지막이 Python 아니다. |
| Python$ | You need Python | Yes | 문장의 마지막이 Python 이다. |

# I 메타 문자 (Meta Character)



- OR |

→ |는 로직 OR의 패턴을 나타낼 때 사용된다.

| 정규표현식 | 문자열 | Match 여부 | 설명 |
|---|---|---|---|
| love\|hate | I love you | Yes | 문자열에 love 포함됨. |
| love\|hate | I hate him | Yes | 문자열에 hate 포함됨. |
| love\|hate | I like you | No | 문자열에 love나 hate가 없음. |

# I 정규 표현식 사용 예



- grep() 함수:

  아래에서 book은 문자열 list임.

⇒ grep("^This",book) # 'This'로 시작.

⇒ grep("^[Ii]",book) # 'I' 또는 'i'로 시작.

⇒ grep("love|hate",book) # 'love' or 'hate'.

⇒ len(grep("love|hate",book))/len(book) # 'love' or 'hate'의 비율.

# I 정규 표현식 사용 예



- 전처리:

```
corpus = []
for i in range(0, len(my_docs)):
    preproc = re.sub(r'\W', ' ', str(my_docs[i]))      # Non-alphanumeric to space.
    preproc = preproc.lower()
    preproc = re.sub(r'\s+[a-z]\s+', ' ',preproc)      # Single character to space.
    preproc = re.sub(r'\s+', ' ', preproc)             # Multiple spaces to a single space.
    corpus.append(review)
```

# 끝.



# 감사합니다.


FAST CAMPUS
ONLINE
장순용 강사.