Tokenize

1) 토큰화

a. 문장으로 토큰화

`nltk.tokenize.sent_tokenize` : 주어진 텍스트를 개별 문장으로 토큰화.

- 예시

```
import nltk
from nltk.tokenize import sent_tokenize

text = "a! bc. d. e? f~ g)"
text2 = "hi! my name is soyoung. and you? um~ ex)"
print(sent_tokenize(text))
print(sent_tokenize(text2))
```

- 결과

```
['a!', 'bc.', 'd. e?', 'f~ g)']
['hi!', 'my name is soyoung.', 'and you?', 'um~ ex)']
```

- a. 와 같은 건 말머리라고 판단한다.

2) 정규 표현식

a. 정규표현식?

- 복잡한 문자열을 처리할 때 사용하는 기법
- 왜 필요한가?
  - "James 990505-1012345\nTony 940105-1922111"에서 주민등록번호 뒷7자리를 *******로 수정해야할 때
  - 무식하게 처리할 경우
    - ```
      text= "James 990505-1012345\nTony 940105-1922111"

      res = []
      for t in text.split("\n"):
          res.append(t[:-7]+"*******")
      print("\n".join(res))
      ```
  - 정규화를 사용할 경우
      ```

        - ```
          import re

          text= "James 990505-1012345\nTony 940105-1922111"
          pat = re.compile("(\d{6})[-]\d{7}")
          print(pat.sub("\g<1>-*******", text))
          ```

- 사용법
    - `[ ? ]` : 에 속하는 것 중 하나라도 매치
        - `[0-9]` : 0~9
        - `[a-zA-Z]` : 모든 알파벳
        - `[^0-9]` : 숫자가 아닌 모든 것
    - 자주 사용되는 것
        - `\d` - 숫자와 매치, [0-9]와 동일한 표현식이다.
        - `\D` - 숫자가 아닌 것과 매치, `[^0-9]`와 동일한 표현식이다.
        - `\s` - whitespace 문자와 매치, `[ \t\n\r\f\v]`와 동일한 표현식이다. 맨 앞의 빈 칸은 공백문자(space)를 의미한다.
        - `\S` - whitespace 문자가 아닌 것과 매치, `[^ \t\n\r\f\v]`와 동일한 표현식이다.
        - `\w` - 문자+숫자(alphanumeric)와 매치, `[a-zA-Z0-9_]`와 동일한 표현식이다.
        - `\W` - 문자+숫자(alphanumeric)가 아닌 문자와 매치, `[^a-zA-Z0-9_]`와 동일한 표현식이다.
    - dot
        - `a.b` : a + 모든문자(최소 하나) + b
        - `a[.]b` : a + . + b
    - *
        - `a*` : a를 n번(n>=0) 반복
    - +
        - `a+` : a를 n번(n>0) 반복
    - {}
        - `{n,m}` : n~m회 제한
        - `{1,}` : +와 동일
        - `{0,}` : *와 동일
        - `a{2}` : a를 2번 반복
    - ?
        - `a?` : a가 있어도, 없어도 된다.
    - 상세 : https://wikidocs.net/4309

b. 불용어 처리

불용어는 문장의 전체적인 의미에 크게 기여하지 않음.

검색 공간을 줄이기 위해 불용어를 삭제하면 좋다.
          ```

3) 토큰의 대체 및 수정

오류를 제거하기 위해 단어 대체 필요. ex) doesn't -> does not

```
text = "Don't hesitate to ask questions"

print(word_tokenize(text)) //['Do', "n't", 'hesitate', 'to', 'ask',
'questions']
print(word_tokenize(replacer.replace(text))) //['Do', 'not', 'hesitate',
'to', 'ask', 'questions']
```

- 먼저 대체를 하고 tokenize하는 것이 더 효율적

```
# 단어 풀어쓰기
class RegexpReplacer(object):
    def __init__(self, patterns=replacement_patterns):
        self.patterns = [(re.compile(regex), repl) for (regex, repl) in
patterns]

    def replace(self, text):
        s = text
        for (pattern, repl) in self.patterns:
            (s, count) = re.subn(pattern, repl, s)
        return s

# 중복되는 단어 수정 ex) lottttt -> lot
class RepeatReplacer(object):
    def __init__(self):
        self.repeat_regexp = re.compile(r"(\w*)(\w)\2(\w*)")
        self.repl = r"\1\2\3"

    def replace(self, word):
        if wordnet.synsets(word): return word
        repl_word = self.repeat_regexp.sub(self.repl, word)
        if repl_word != word:
            return self.replace(repl_word)
        else:
            return repl_word

# 대체 가능한 단어 대체
class WordReplacer(object):
    def __init__(self, word_map):
        self.word_map = word_map
    def replace(self, word):
        return self.word_map.get(word, word)
```

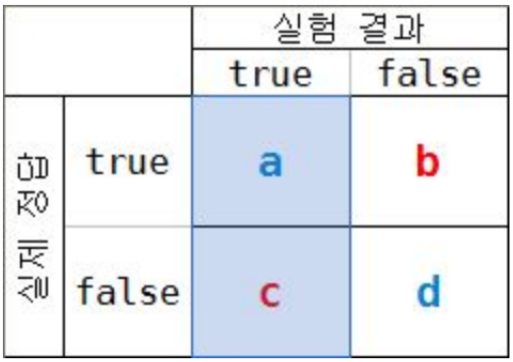
4) 측정

a. precision, recall, accuracy

- 전제

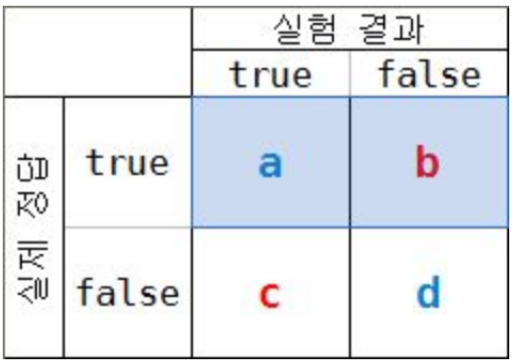
  - TT : 실제 정답 T, 실험 결과 T (a)
  - TF : 실제 정답 T, 실험 결과 F (b)
  - FT : 실제 정답 F, 실험 결과 T (c)
  - FF : 실제 정답 F, 실험 결과 F (d)

- precision



  - a/(a+c)
  - 실험 결과 True라고 판단된 것 중에 실제 정답이 True인 것

- recall



  - a/(a+b)
  - 실제 정답이 True인 것 중에 실험 결과가 True인 것

- accuracy

		실험 결과	
		true	false
실제 정답	true	a	b
	false	c	d

- (a+d)/(a+b+c+d)
- 전체 결과 중에서 정답인 것

b. F-measure

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

- precision과 recall에 대한 평균에 가중치를 주는 것

c. Edit distance

- 두 문자열의 유사도를 판단
- 문자열 A를 B로 바꾸기 위해 필요한 연산의 최소 횟수
  - 비교할 두 문자가 같으면 cost(i,j) = cost(i-1, j-1)
  - 비교할 두 문자가 다르면 cost(i,j) = 1 + min(cost(i-1,j),cost(i,j-1),cost(i-1,j-1))

```
def _edit_dist_init(len1, len2):
    A = []
    for i in range(len1):
        A.append([0] * len2)

    # (i,0), (0,j) 채우기
    for i in range(len1):
        A[i][0] = i
    for j in range(len2):
        A[0][j] = j

    return A

def _edit_dist_step(A, i, j, s1, s2, transpositions=False):
    c1 = s1[i-1]
    c2 = s2[j-1]

    a = A[i-1][j] + 1 # s1에서 skip
    b = A[i][j-1] + 1 # s2에서 skip
    c = A[i-1][j-1] + (c1!=c2) # 대체
    d = c+1 # X select
```

```
    if transpositions and i>1 and j>1:
        if s1[i-2] == c2 and s2[j-2] == c1:
            d = A[j-2][j-2] + 1

        A[i][j] = min(a,b,c,d)

def edit_distance(s1, s2, transpositions=False):
    len1 = len(s1)
    len2 = len(s2)
    lev = _edit_dist_init(len1 + 1, len2 + 1)

    for i in range(len1):
        for j in range(len2):
            _edit_dist_step(lev, i+1, j+1, s1, s2,
transpositions=transpositions)
    return lev[len1][len2]
```

d. jaccard distance

- 두 개의 객체를 집합으로 간주하여 유사성을 측정

```
def jacc_sim(query, document):
    a = set(query).intersection(set(document))
    b = set(query).union(set(document))
    return len(a)/len(b)
```

e. smith waterman distance

- 보통 DNA 서열 검출을 위해 사용

5) 최대우도추정 (MLE)

a. 컴퓨터 언어학

- 기계 번역, 음성 인식, 지능형 웹 검색, 정보 검색, 지능형 철자 검색기

b. 최대우도추정 (MLE)

- Maximum Likelihood Estimation
    - 어떤 확률변수에서 표집한 모집단으로 그 확률변수의 모수(모집값의 대표 값,parameter)를 구하는 방법.
    - 어떤 모수가 나왔을 때, 원하는 값들이 최대한 나올 수 있게 하는 모수를 선택하는 방법.
    - 도수 분포를 얻기 위해 사용. MLE는 도수 분포에서 빈도에 기초하여 모든 발생 확률을 계산한다.
    - 우도 $L(\theta|x)$는 θ가 전제되었을 때 표본 x가 등장할 확률인 $p(x|\theta)$에 비례한다.
        - 예시

동전던지기를 100번 시행했는데, 앞면이 56번 나왔다고 가정해보겠습니다. 반복적인 동전던지기는 성공확률이 p인 베르누이시행을 n번 반복시행할 때 성공횟수의 분포인 **이항분포(binomial distribution)**를 따릅니다. 이 예시에서 우리가 알고 싶은 미지의 모수 θ는 동전을 한 번 던졌을 때 앞면이 나올 확률 p가 됩니다. 이를 위해 앞면이 나올 확률이 p인 이항분포에서 뽑은 표본 x(성공횟수=앞면이 나온 횟수=56번)를 활용합니다.

이항분포의 확률함수는 다음과 같습니다.

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

우선 이 동전이 공정하다($\theta = 0.5$)고 가정하고 우도를 계산해보겠습니다.

$$p(X = 56|\theta = 0.5) = \binom{100}{56} 0.5^{56} 0.5^{44} \approx 0.0389$$

- 계산으로 구하기
  - θ에 대해 미분이 가능할 경우
    - 이항분포의 확률함수
    - θ에 대해 편미분을 해 0이 되는 지점을 구하면 우도를 최대화하는 θ를 구할 수 있다.
  - θ에 대해 미분이 불가능할 경우
    - 그래디언트 디센트 등 반복적이고 점진적인 방식으로 θ를 추정.
    - 로지스틱 회귀나 딥러닝 등 모델의 θ를 최대우도추정 기법으로 추정할 때 자주 쓰는 기