자연어처리 중간고사 대체 과제

소프트웨어학과 20171487 김태민

실습 5-1)

```
In [14]: # konlpy 관련 패키지 import
         from konlpy.tag import Okt
         from konlpy.tag import Kkma
         from konlpy.tag import Hannanum
         from konlpy.tag import Komoran
         from konlpy.tag import Twitter

In [15]: kkma = Kkma()
         okt = Okt()
         komoran = Komoran()
         hannanum = Hannanum()
         twitter = Twitter()

In [16]: # konlpy 중 Kkma는 문장 분리가 가능 (다른 라이브러리는 되지 않음)
         print ("kkma 문장 분리 : ", kkma.sentences('네 안녕하세요 반갑습니다.'))

         kkma 문장 분리 :  ['네 안녕하세요', '반갑습니다.']

In [17]: # konlpy 의 라이브러리 형태소 분석 비교
         print("okt 형태소 분석 :", okt.morphs(u"집에 가면 감자 좀 쪄줄래?"))
         print("kkma 형태소 분석 : ", kkma.morphs(u"집에 가면 감자 좀 쪄줄래?"))
         print("hannanum 형태소 분석 : ", hannanum.morphs(u"집에 가면 감자 좀 쪄줄래?"))
         print("komoran 형태소 분석 : ", komoran.morphs(u"집에 가면 감자 좀 쪄줄래?"))
         print("twitter 형태소 분석 : ", twitter.morphs(u"집에 가면 감자 좀 쪄줄래?"))

         okt 형태소 분석 : ['집', '에', '가면', '감자', '좀', '쪄줄래', '?']
         kkma 형태소 분석 :  ['집', '에', '가', '면', '감자', '좀', '찌', '어', '주', 'ㄹ래', '?']
         hannanum 형태소 분석 :  ['집', '에', '가', '면', '감', '자', '좀', '찌', '어', '줄', '래', '?']
         komoran 형태소 분석 :  ['집', '에', '가', '면', '감자', '좀', '찌', '어', '주', 'ㄹ래', '?']
         twitter 형태소 분석 :  ['집', '에', '가면', '감자', '좀', '쪄줄래', '?']

In [ ]:
```

Konlpy에서 Kkma는 문장 분리가 가능하다. Kkma.sentences()에 '네 안녕하세요 반갑습니다.' 라는 문장을 넣어 실행해본 결과, '네 안녕하세요' 와 '반갑습니다' 로 분리된 것을 알 수 있다. '집에 가면 감자 좀 쪄줄래?'라는 문장을 넣어 비교해본 결과 '감자'와 '쪄줄래'를 분석하는 부분에서 차이가 있었다. hannanum에서는 명사인 '감자'를 음절로 나누어 '감'과 '자'로 분리했다. Okt, twitter에서는 '쪄줄래'를 분리하지 않고 한 어절로 취급하였고 kkma, komoran에서는 '쪄줄래'의 원형을 살려 '찌', '어', '주', 'ㄹ래'로 구분했다. Hannanum에서는 '찌', '어', '줄', '래'로 구분했다.

문장 수정)

```
In [24]: # 다른 문장으로 비교
         print("okt 형태소 분석 :", okt.morphs(u"이번 여름 휴가엔 바다를 갈 것이다."))
         print("kkma 형태소 분석 : ", kkma.morphs(u"이번 여름 휴가엔 바다를 갈 것이다."))
         print("hannanum 형태소 분석 : ", hannanum.morphs(u"이번 여름 휴가엔 바다를 갈 것이다."))
         print("komoran 형태소 분석 : ", komoran.morphs(u"이번 여름 휴가엔 바다를 갈 것이다."))
         print("twitter 형태소 분석 : ", twitter.morphs(u"이번 여름 휴가엔 바다를 갈 것이다."))

         okt 형태소 분석 : ['이번', '여름', '휴가', '엔', '바다', '를', '갈', '것', '이다', '.']
         kkma 형태소 분석 :  ['이번', '여름', '휴가', '에', '는', '바다', '를', '갈', 'ㄹ', '것', '이', '다', '.']
         hannanum 형태소 분석 :  ['이번', '여름', '휴가', '엔', '바다', '를', '가', 'ㄹ', '것', '이', '다', '.']
         komoran 형태소 분석 :  ['이번', '여름', '휴가', '에', 'ㄴ', '바다', '를', '가', 'ㄹ', '것', '이', '다', '.']
         twitter 형태소 분석 :  ['이번', '여름', '휴가', '엔', '바다', '를', '갈', '것', '이다', '.']
```

'이번 여름 휴가엔 바다를 갈 것이다'라는 문장을 넣어봤을 때 이번에는 kkma와 komoran 분석기 사이에 차이점이 존재했다. Kkma는 '엔'과 '갈'을 각각 '에', '는', '갈', 'ㄹ'로 구분했다. Komoran은 '엔'과 '갈'을 각각 '에', 'ㄴ', '가', 'ㄹ' 로 구분했다. 해당 문장에서는 komoran이 형태소를 잘 살려서 구

분한 것으로 보인다.

실습 5-2)

```
In [1]: # konlpy 관련 패키지 import
        from konlpy.tag import Okt
        from konlpy.tag import Kkma
        from konlpy.tag import Hannanum
        from konlpy.tag import Komoran
        from konlpy.tag import Twitter

In [3]: kkma = Kkma()
        okt = Okt()
        komoran = Komoran()
        hannanum = Hannanum()
        twitter = Twitter()

In [4]: # konlpy 의 라이브러리 품사태깅 비교
        print("okt 품사태깅 :", okt.pos(u"집에 가면 감자 좀 쪄줄래?"))
        print("kkma 품사태깅 : ", kkma.pos(u"집에 가면 감자 좀 쪄줄래?"))
        print("hannanum 품사태깅 : ", hannanum.pos(u"집에 가면 감자 좀 쪄줄래?"))
        print("komoran 품사태깅 : ", komoran.pos(u"집에 가면 감자 좀 쪄줄래?"))
        print("twitter 품사태깅 : ", twitter.pos(u"집에 가면 감자 좀 쪄줄래?"))

        okt 품사태깅 : [('집', 'Noun'), ('에', 'Josa'), ('가면', 'Noun'), ('감자', 'Noun'), ('좀', 'Noun'), ('쪄줄래', 'Verb'), ('?', 'Punctuatio
        n')]
        kkma 품사태깅 :  [('집', 'NNG'), ('에', 'JKM'), ('가', 'VV'), ('면', 'ECE'), ('감자', 'NNG'), ('좀', 'MAG'), ('찌', 'VV'), ('어', 'ECS'),
        ('주', 'VXV'), ('ㄹ래', 'EFQ'), ('?', 'SF')]
        hannanum 품사태깅 :  [('집', 'N'), ('에', 'J'), ('가', 'P'), ('면', 'E'), ('감', 'P'), ('자', 'E'), ('좀', 'M'), ('찌', 'P'), ('어', 'E'),
        ('줄', 'P'), ('래', 'E'), ('?', 'S')]
        komoran 품사태깅 :  [('집', 'NNG'), ('에', 'JKB'), ('가', 'VV'), ('면', 'EC'), ('감자', 'NNP'), ('좀', 'MAG'), ('찌', 'VV'), ('어', 'EC'),
        ('주', 'VX'), ('ㄹ래', 'EF'), ('?', 'SF')]
        twitter 품사태깅 :  [('집', 'Noun'), ('에', 'Josa'), ('가면', 'Noun'), ('감자', 'Noun'), ('좀', 'Noun'), ('쪄줄래', 'Verb'), ('?', 'Punctu
        ation')]

In [ ]:
```

각 형태소 분석기는 품사 태그명에 차이가 있다. Hannanum 분석기는 5언을 기준으로 분석하고 hannanum을 제외한 다른 분석기들은 9품사를 세분화한 기준으로 분석했다. 또한 각 분석기들 마다 품사 태그명이 다르다. 예를 들어, '집'이라는 단어에 태그를 했을 때, okt, twitter에서는 명사를 칭하는 'Noun' 이라는 태그로 분석했고 kkma와 komoran에서는 'NNG'로 분석했다. Hannanum에서는 체언이라는 뜻으로 'N'으로 구분했다. Kkma와 komoran은 '감자'라는 명사를 세분화했다는 점은 같지만 kkma에서는 보통 명사로, komoran에서는 고유 명사로 분석했다는 차이점이 있다. 문맥을 파악했을 때 kkma에서 보통 명사로 분석한 것이 적절하다고 볼 수 있다.

문장 수정)

```
In [5]: # 다른 문장으로 비교
        print("okt 품사태깅 :", okt.pos(u"오늘 점심 뭐 먹을까?"))
        print("kkma 품사태깅 : ", kkma.pos(u"오늘 점심 뭐 먹을까?"))
        print("hannanum 품사태깅 : ", hannanum.pos(u"오늘 점심 뭐 먹을까?"))
        print("komoran 품사태깅 : ", komoran.pos(u"오늘 점심 뭐 먹을까?"))
        print("twitter 품사태깅 : ", twitter.pos(u"오늘 점심 뭐 먹을까?"))

        okt 품사태깅 : [('오늘', 'Noun'), ('점심', 'Noun'), ('뭐', 'Noun'), ('먹을까', 'Verb'), ('?', 'Punctuation')]
        kkma 품사태깅 :  [('오늘', 'NNG'), ('점심', 'NNG'), ('뭐', 'NP'), ('먹', 'VV'), ('을까', 'EFQ'), ('?', 'SF')]
        hannanum 품사태깅 :  [('오늘', 'N'), ('점심', 'N'), ('뭐', 'N'), ('먹', 'P'), ('을까', 'E'), ('?', 'S')]
        komoran 품사태깅 :  [('오늘', 'NNG'), ('점심', 'NNP'), ('뭐', 'NP'), ('먹', 'VV'), ('을까', 'EF'), ('?', 'SF')]
        twitter 품사태깅 :  [('오늘', 'Noun'), ('점심', 'Noun'), ('뭐', 'Noun'), ('먹을까', 'Verb'), ('?', 'Punctuation')]
```

다른 문장으로 테스트해봤을 때 kkma에서는 점심을 보통 명사로 분석했고 komoran에서는 고유 명사로 분석했다. 앞선 2개의 문장에서 결과를 봤을 때 명사를 보통 명사와 고유 명사로 구분하는 것은 kkma가 더 적절한 것을 알 수 있다.

실습 6-1)

NLTK 패키지 다운로드

```
In [61]: import nltk
```

구구조 구문 분석 규칙 작성

```
In [62]: grammar = nltk.CFG.fromstring("""
S -> NP VP
NP -> NN XSN JK | NN JK
VP -> NP VP | VV EP EF | VV EF
NN -> '배' | '케이크' | '아이'
XSN -> '들'
JK -> '이' | '를' | '가'
VV -> '먹' | '고프'
EP -> '었'
EF -> '다'
""")
```

규칙 기반 구문 분석기 생성 및 구구조 구문 분석 수행

제시된 ChartParser 외에도 ShiftReduceParser, RecursiveDescentParser 등

다양한 구문 분석 알고리즘이 제공된다.

```
In [63]: parser = nltk.ChartParser(grammar)

In [64]: sent = ['아이', '들', '이', '케이크', '를', '먹', '었', '다']

In [65]: for tree in parser.parse(sent):
           print(tree)

(S
  (NP (NN 아이) (XSN 들) (JK 이))
  (VP (NP (NN 케이크) (JK 를)) (VP (VV 먹) (EP 었) (EF 다))))

In [66]: sent2 = ['배', '가', '고프', '다']

In [67]: for tree in parser.parse(sent2):
           print(tree)

(S (NP (NN 배) (JK 가)) (VP (VV 고프) (EF 다)))
```

'아이들이 케이크를 먹었다'라는 문장을 이용하여 규칙 기반 구문 분석기 생성 및 구구조 구문 분석을 수행했다. 문장을 기준으로 구문 분석기를 생성하려면 몇 가지 규칙을 선언해주는 것이 필요하기 때문에 해당 문장을 형태소로 단위로 분류하여 선언해준다. 예를 들어, '아이들이'의 경우 '아이', '들', '이'를 각각 명사'NN', 명사 파생 접미사'XSN', 주격 조사'JK'로 구분하여 선언해주어야 한다. 결과를 살펴보면 주어절('아이들이')+서술절(케이크를+서술절(먹었다))의 구조를 가지고 있는 것을 알 수 있다.

구문 분석기에 문장 추가)

```
In [66]: sent2 = ['배', '가', '고프', '다']

In [67]: for tree in parser.parse(sent2):
           print(tree)

(S (NP (NN 배) (JK 가)) (VP (VV 고프) (EF 다)))
```

'배가 고프다'라는 문장을 형태소 별로 나누어 sent2에 저장하고 이를 분석했다.

배(명사), 가(주격 조사), 고프(동사), 다(종결 어미)

위와 같이 해당하는 규칙에 형태소를 넣어주고 출력을 해본 결과, 올바르게 분석이 된 것을 확인할 수 있다.

실습 6-2)

```
In [17]: import spacy
```

영어 문장의 의존 구문 분석 수행

Spacy 모델은 문장을 token들로 구성된 document로 처리한다.

각 token에는 품사, 의존 관계, 개체명 정보 등이 태깅된다.

이 예제에서 출력하는 정보들은 다음과 같다.

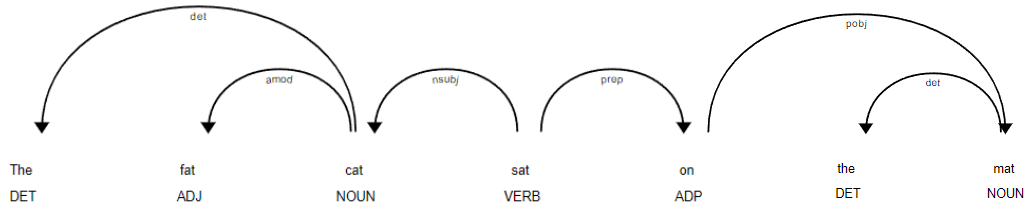
- token.text: token 문자열
- token.dep_: token과 token의 지배소 간의 의존 관계 유형
- token.head: 지배소 token

```
In [18]: # 영어 multi-task 통계 모델
nlp = spacy.load('en_core_web_sm')

In [19]: doc = nlp('The fat cat sat on the mat')

In [20]: for token in doc:
  print(token.text, token.dep_, token.head.text)
```

```
The det cat
fat amod cat
cat nsubj sat
sat ROOT sat
on prep sat
the det mat
mat pobj on
```

의존 구문 분석 결과를 시각화한다.

```
In [21]: from spacy import displacy

In [23]: # Jupyter, Colab 등에서 동작
displacy.render(doc, style='dep', jupyter=True)
```

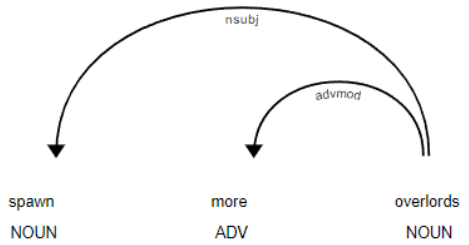


의존 구문 분석을 수행하였고 결과에서는 결과를 시각화했다. 결과를 분석하면 CAT은 명사로써 관사 The에 종속적이며 관형사 fat에 종속적이다. sat은 전치사 on에 종속적이며 on은 명사 mat의 목적격이므로 mat에 종속적이고 mat 또한 관사 the에 종속적이다.

문장 수정)

```
In [40]: doc = nlp('spawn more overlords')
```

```
In [41]: displacy.render(doc, style='dep', jupyter=True)
```



Overload는 명사로써 관형사 more에 종속적이고 spawn은 동사로써 명사 overlord에 종속적이다.

실습 7-1)

wordnet 관련 패키지 nltk import

```
In [1]: import nltk
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import wordnet
from nltk import word_tokenize
from nltk.corpus import stopwords
import sys
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\USER\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\USER\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\USER\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

단어와 문장에 나타난 단어에 대해 Best Sense 추출

```
In [2]: def disambiguate(word, sentence, stopwords):
        # Best sense 를 얻기위한 Lesk 알고리즘을 작성해보세요.

        word_senses = wordnet.synsets(word)
        best_sense = word_senses[0]  # Assume that first sense is most freq.
        max_overlap = 0
        context = set(word_tokenize(sentence))
        for sense in word_senses:
            signature = tokenized_gloss(sense)
            overlap = compute_overlap(signature, context, stopwords)
            if overlap > max_overlap:
                max_overlap = overlap
                best_sense = sense

        return best_sense
```

sense의 definition에 대한 모든 token 추출

In [2]:
```
def disambiguate(word, sentence, stopwords):
        # Best sense 를 얻기위한 Lesk 알고리즘을 작성해보세요.

        word_senses = wordnet.synsets(word)
        best_sense = word_senses[0]  # Assume that first sense is most freq.
        max_overlap = 0
        context = set(word_tokenize(sentence))
        for sense in word_senses:
            signature = tokenized_gloss(sense)
            overlap = compute_overlap(signature, context, stopwords)
            if overlap > max_overlap:
                max_overlap = overlap
                best_sense = sense

        return best_sense
```

sense의 definition에 대한 모든 token 추출

In [3]:
```
def tokenized_gloss(sense):
        tokens = set(word_tokenize(sense.definition()))
        for example in sense.examples():
            tokens.union(set(word_tokenize(example)))
        return tokens
```

겹치는 단어 비교

In [4]:
```
def compute_overlap(signature, context, stopwords):
        gloss = signature.difference(stopwords)
        return len(gloss.intersection(context))
```

Main

In [5]:
```
stopwords = set(stopwords.words('english'))# NLTK에서 지정한 영어 불용어 처리 ex) i, my, they...
sentence = ("They eat a meal")
context = set(word_tokenize(sentence))
word = 'eat'

print("Word :", word)
syn = wordnet.synsets('eat')[1]
print("Sense :", syn.name())
print("Definition :", syn.definition())
print("Sentence :", sentence)

signature = tokenized_gloss(syn)
print(signature)
print(compute_overlap(signature, context, stopwords))
print("Best sense: ", disambiguate(word, sentence, stopwords))
```

```
Word : eat
Sense : eat.v.02
Definition : eat a meal; take a meal
Sentence : They eat a meal
{'take', 'meal', 'eat', 'a', ';'}
2
Best sense:  Synset('eat.v.02')
```

중의성을 해결하기 위한 알고리즘인 lesk 알고리즘에 대한 실습을 진행해보았다.

결과를 살펴보면 다음과 같다

Word : 'eat' – 중의성을 파악할 단어

Sense : 기존에 저장돼 있던 'eat'의 사전적 정의에서 의미가 겹치는 개수

Definition : 가장 적절하다 판단되는 사전적 정의

Eat은 먹다, 파괴하다, 낭비하다 등 여러 의미가 많지만 해당 문장에서는 '먹다'로 올바르게 분석한 것을 알 수 있다.

문장 변경)

```
In [17]: stopwords = set(stopwords.words('english'))# NLTK에서 지정한 영어 불용어 처리 ex) i, my, they...
sentence = ("tom is really mean")
context = set(word_tokenize(sentence))
word = 'mean'

print("Word :", word)
syn = wordnet.synsets('mean')[1]
print("Sense :", syn.name())
print("Definition :", syn.definition())
print("Sentence :", sentence)

signature = tokenized_gloss(syn)
print(signature)
print(compute_overlap(signature, context, stopwords))
print("Best sense: ", disambiguate(word, sentence, stopwords))
```

```
Word : mean
Sense : mean.v.01
Definition : mean or intend to express or convey
Sentence : tom is really mean
{'convey', 'intend', 'to', 'express', 'mean', 'or'}
1
Best sense:  Synset('mean.v.01')
```

이번에는 문장을 ' tom is really mean '으로 바꿔서 실행해보았다. 문맥상 mean은 '못된', '심술궂은' 등으로 해석이 되면 올바르게 해석이 되었다고 볼 수 있다. Word를 'mean'으로 지정해주고 실행해본 결과 Definition에서는 ' mean or intend to express or convey ' 즉 '의미' 또는 '의미하다' 라는 뜻으로 해석하여 잘못 정의 된 것을 알 수 있다. 겹치는 의미도 1개 밖에 없어 기존에 저장된 정의에 의존하는 lesk 알고리즘의 문제점을 알 수 있었다.