확률론적 언어 모형

확률론적 언어 모형(Probabilistic Language Model)은 m개의 단어 $w_1, w_2, \dots, w_m$ 열(word sequence)이 주어졌을 때 문장으로써 성립될 확률 $P(w_1, w_2, \dots, w_m)$ 을 출력함으로써 이 단어 열이 실제로 현실에서 사용될 수 있는 문장(sentence)인지를 판별하는 모형이다.

이 확률은 각 단어의 확률과 단어들의 조건부 확률을 이용하여 다음과 같이 계산할 수 있다.

$$\begin{aligned} P(w_1, w_2, \dots, w_m) &= P(w_1, w_2, \dots, w_{m-1}) \cdot P(w_m \mid w_1, w_2, \dots, w_{m-1}) \\ &= P(w_1, w_2, \dots, w_{m-2}) \cdot P(w_{m-1} \mid w_1, w_2, \dots, w_{m-2}) \cdot P(w_m \mid w_1, w_2, \dots, w_{m-1}) \\ &= P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_1, w_2) P(w_4 \mid w_1, w_2, w_3) \cdots P(w_m \mid w_1, w_2, \dots, \end{aligned}$$

여기에서 $P(w_m \mid w_1, w_2, \dots, w_{m-1})$ 은 지금까지 $w_1, w_2, \dots, w_{m-1}$라는 단어 열이 나왔을 때, 그 다음 단어로 w_m이 나올 조건부 확률을 말한다. 여기에서 지금까지 나온 단어를 **문맥(context)** 정보라고 한다.

이 때 조건부 확률을 어떻게 모형화하는냐에 따라

- 유니그램 모형(Unigram Model)
- 바이그램 모형(Bigram Model)
- N그램 모형(N-gram Model)

등으로 나뉘어 진다.

유니그램 모형

만약 모든 단어의 활용이 완전히 서로 독립이라면 단어 열의 확률은 다음과 같이 각 단어의 확률의 곱이 된다. 이러한 모형을 유니그램 모형이라고 한다.

$$P(w_1, w_2, \dots, w_m) = \prod_{i=1}^{m} P(w_i)$$

바이그램 모형

만약 단어의 활용이 바로 전 단어에만 의존한다면 단어 열의 확률은 다음과 같다. 이러한 모형을 바이그램 모형 또는 마코프 모형(Markov Model)이라고 한다.

$$P(w_1, w_2, \dots, w_m) = P(w_1) \prod_{i=2}^{m} P(w_i \mid w_{i-1})$$

N그램 모형

만약 단어의 활용이 바로 전 $n-1$개의 단어에만 의존한다면 단어 열의 확률은 다음과 같다. 이러한 모형을 N그램 모형이라고 한다.

$$P(w_1, w_2, \ldots, w_m) = P(w_1) \prod_{i=n}^{m} P(w_i \mid w_{i-1}, \ldots, w_{i-n})$$

NLTK의 N그램 기능

NLTK 패키지에는 바이그램과 N-그램을 생성하는 `bigrams` , `ngrams` 명령이 있다.

In [1]:

```
from nltk import bigrams, word_tokenize
from nltk.util import ngrams

sentence = "I am a boy."
tokens = word_tokenize(sentence)

bigram = bigrams(tokens)
trigram = ngrams(tokens, 3)

print("₩nbigram:")
for t in bigram:
    print(t)

print("₩ntrigram:")
for t in trigram:
    print(t)
```

```
bigram:
('I', 'am')
('am', 'a')
('a', 'boy')
('boy', '.')

trigram:
('I', 'am', 'a')
('am', 'a', 'boy')
('a', 'boy', '.')
```

조건부 확률을 추정할 때는 문장의 시작과 끝이라는 조건을 표시하기 위해 모든 문장에 문장의 시작과 끝을 나타내는 특별 토큰을 추가한다. 예를 들어 문장의 시작은 `SS` , 문장의 끝은 `SE` 이라는 토큰을 사용할 수 있다.

예를 들어 ["I", "am", "a", "boy", "."]라는 토큰열(문장)은 ["SS", "I", "am", "a", "boy", ".", "SE"]라는 토큰열이 된다. ngrams 명령은 padding 기능을 사용하여 이런 특별 토큰을 추가할 수 있다.

In [2]:

```
bigram = ngrams(tokens, 2, pad_left=True, pad_right=True, left_pad_symbol="SS", right_pad_symbol="
for t in bigram:
    print(t)
```

```
('SS', 'I')
('I', 'am')
('am', 'a')
('a', 'boy')
('boy', '.')
('.', 'SE')
```

조건부 확률 추정 방법

NLTK패키지를 사용하면 바이그램 형태의 조건부 확률을 쉽게 추정할 수 있다. 우선 ConditionalFreqDist 클래스로 각 문맥별 단어 빈도를 측정한 후에 ConditionalProbDist 클래스를 사용하면 조건부 확률을 추정한다.

In [3]:

```
from nltk import ConditionalFreqDist

sentence = "I am a boy."
tokens = word_tokenize(sentence)
bigram = ngrams(tokens, 2, pad_left=True, pad_right=True, left_pad_symbol="SS", right_pad_symbol="
cfd = ConditionalFreqDist([(t[0], t[1]) for t in bigram])
```

ConditionalFreqDist 클래스는 문맥을 조건으로 가지는 사전 자료형과 비슷하다.

In [4]:

```
cfd.conditions()
```

Out[4]:

```
['SS', 'I', 'am', 'a', 'boy', '.']
```

In [5]:

```
cfd["SS"]
```

Out[5]:

```
FreqDist({'I': 1})
```

다음은 nltk 패키지의 샘플 코퍼스인 movie_reviews의 텍스트로부터 바이그램 확률을 추정하는 예제이다.

In [6]:

```
import nltk
nltk.download('movie_reviews')
nltk.download('punkt')
from nltk.corpus import movie_reviews

sentences = []
for tokens in movie_reviews.sents():
    bigram = ngrams(tokens, 2, pad_left=True, pad_right=True, left_pad_symbol="SS", right_pad_symb
    sentences += [t for t in bigram]

sentences[:20]
```

```
[nltk_data] Downloading package movie_reviews to
[nltk_data]     /home/dockeruser/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
[nltk_data] Downloading package punkt to /home/dockeruser/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[6]:

```
[('SS', 'plot'),
 ('plot', ':'),
 (':', 'two'),
 ('two', 'teen'),
 ('teen', 'couples'),
 ('couples', 'go'),
 ('go', 'to'),
 ('to', 'a'),
 ('a', 'church'),
 ('church', 'party'),
 ('party', ','),
 (',', 'drink'),
 ('drink', 'and'),
 ('and', 'then'),
 ('then', 'drive'),
 ('drive', '.'),
 ('.', 'SE'),
 ('SS', 'they'),
 ('they', 'get'),
 ('get', 'into')]
```

문장의 처음(SS 문맥), i라는 단어 다음, 마침표 다음에 나오는 단어의 빈도는 다음과 같다.

In [7]:

```
cfd = ConditionalFreqDist(sentences)
```

- 문장의 처음에 올 수 있는 단어들

In [8]:

```
cfd["SS"].most_common(5)
```

Out[8]:

```
[('the', 8071), ('.', 3173), ('it', 3136), ('i', 2471), ('but', 1814)]
```

- i 다음에 올 수 있는 단어들

In [9]:

```
cfd["i"].most_common(5)
```

Out[9]:

```
[("'", 1357), ('was', 506), ('can', 351), ('have', 330), ('don', 276)]
```

- 마침표 다음에 올 수 있는 단어들

In [10]:

```
cfd["."].most_common(5)
```

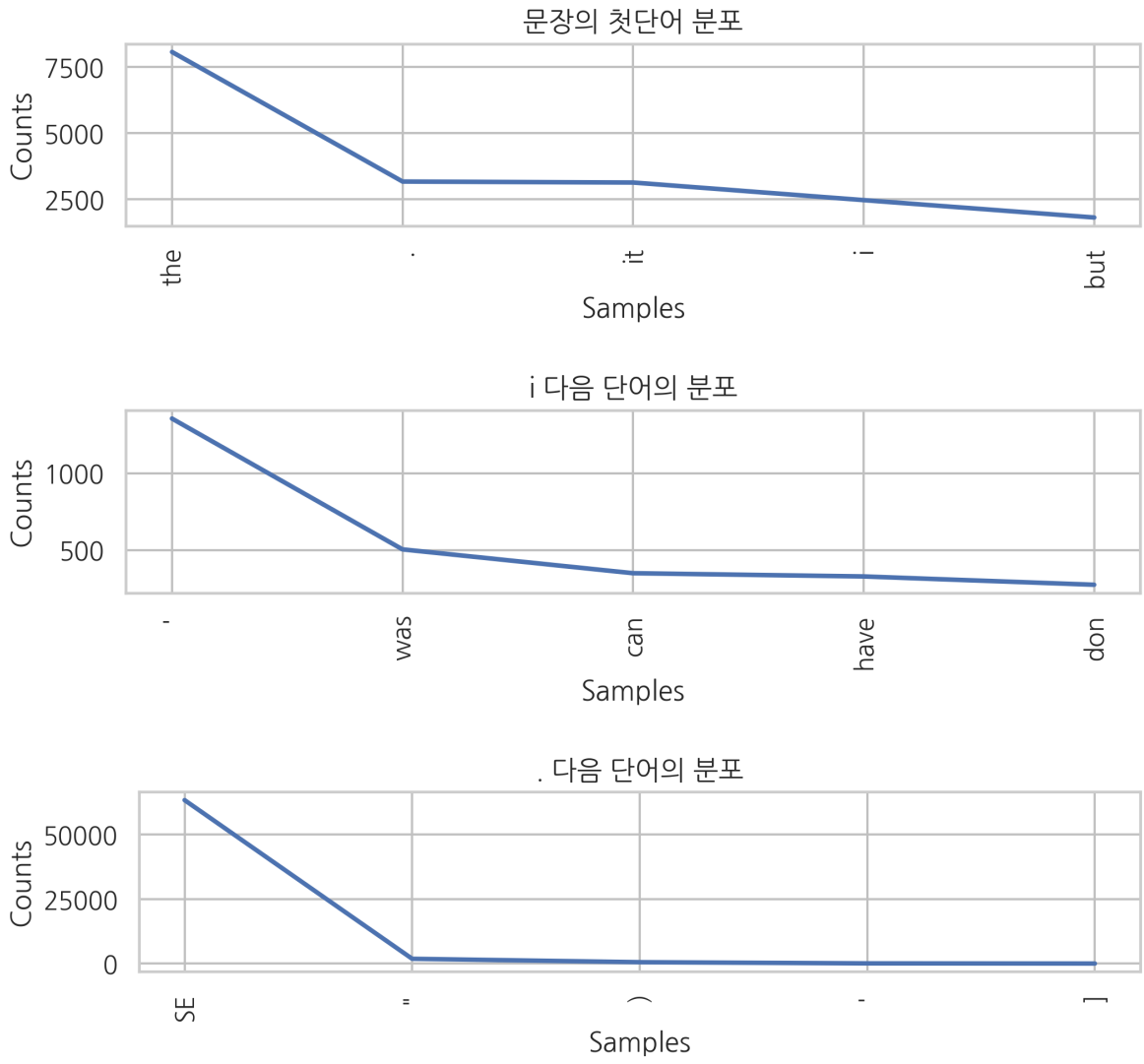
Out[10]:

```
[('SE', 63404), ('"', 1854), (')', 535), ("'", 70), (']', 10)]
```

그림으로 그리면 다음과 같다.

In [11]:

```
plt.subplot(311)
cfd["SS"].plot(5, title="문장의 첫단어 분포")
plt.subplot(312)
cfd["i"].plot(5, title="i 다음 단어의 분포")
plt.subplot(313)
cfd["."].plot(5, title=". 다음 단어의 분포")
```



빈도를 추정하면 각각의 조건부 확률은 기본적으로 다음과 같이 추정할 수 있다.

$$P(w \mid w_c) = \frac{C((w_c, w))}{C((w_c))}$$

위 식에서 $C(w_c, w)$은 전체 말뭉치에서 (w_c, w)라는 바이그램이 나타나는 횟수이고 $C(w_c)$은 전체 말뭉치에서 (w_c)라는 유니그램(단어)이 나타나는 횟수이다.

NLTK의 `ConditionalProbDist` 클래스에 `MLEProbDist` 클래스 팩토리를 인수로 넣어 위와 같이 빈도를 추정할 수 있다.

In [12]:

```
from nltk.probability import ConditionalProbDist, MLEProbDist
cpd = ConditionalProbDist(cfd, MLEProbDist)
```

트레이닝이 끝나면 조건부 확률의 값을 보거나 샘플 문장을 입력해서 문장의 로그 확률을 구할 수 있다.

In [13]:

```
cpd["i"].prob("am")
```

Out[13]:

0.018562267971650354

In [14]:

```
cpd["i"].prob("is")
```

Out[14]:

0.0002249971875351558

In [15]:

```
cpd["we"].prob("are")
```

Out[15]:

0.08504504504504505

In [16]:

```
cpd["we"].prob("is")
```

Out[16]:

0.0

바이그램 언어 모형

조건부 확률을 알게 되면 각 문장의 확률을 구할 수 있다.

다음으로 이 토큰열을 N-그램형태로 분해한다. 바이그램 모형에서는 전체 문장의 확률은 다음과 같이 조건부 확률의 곱으로 나타난다.

$$P(\text{SS I am a boy SE}) = P(\text{I} \mid \text{SS}) \cdot P(\text{am} \mid \text{I}) \cdot P(\text{a} \mid \text{am}) \cdot P(\text{boy} \mid \text{a}) \cdot P(\text{.} \mid \text{boy}) \cdot P(\text{SE} \mid \text{.})$$

우선 다음과 같이 문장(단어 리스트)의 리스트를 만든다.

In [17]:

```
def sentence_score(s):
    p = 0.0
    for i in range(len(s) - 1):
        c = s[i]
        w = s[i + 1]
        p += np.log(cpd[c].prob(w) + np.finfo(float).eps)
    return np.exp(p)
```

In [18]:

```
test_sentence = ["i", "like", "the", "movie", "."]
sentence_score(test_sentence)
```

Out[18]:

2.7407641340715 61e-06

In [19]:

```
test_sentence = ["like", "i", "the", ".", "movie"]
sentence_score(test_sentence)
```

Out[19]:

1.5015040140827832e-38

문장의 생성

이 모형을 기반으로 임의의 랜덤한 문장을 생성할 수 있다.

In [20]:

```
def generate_sentence(seed=None):
    if seed is not None:
        import random
        random.seed(seed)
    c = "SS"
    sentence = []
    while True:
        if c not in cpd:
            break
        w = cpd[c].generate()

        if w == "SE":
            break
        elif w in ["i", "ii", "iii"]:
            w2 = w.upper()
        elif w in ["mr", "luc", "i", "robin", "williams", "cindy", "crawford"]:
            w2 = w.title()
        else:
            w2 = w

        if c == "SS":
            sentence.append(w2.title())
        elif c in ["`", "W"", "'", "("]:
            sentence.append(w2)
        elif w in ["'", ".", ",", ")", ":", ";", "?"]:
            sentence.append(w2)
        else:
            sentence.append(" " + w2)

        c = w
    return "".join(sentence)
```

In [21]:

```
generate_sentence(6)
```

Out[21]:

```
"Writers in one of the prison, yet, works as agent ray liotta, one of watching, we s
upposed to doubt that the holocaust - or the fact, the amc and don's wonderful exper
ience, do what could either."
```

이번에는 한글 자료를 이용해보자 코퍼스로는 아래의 웹사이트에 공개된 Naver sentiment movie corpus 자료를 사용한다.

- https://github.com/e9t/nsmc (https://github.com/e9t/nsmc)

In [22]:

```
%%time
!wget -nc -q https://raw.githubusercontent.com/e9t/nsmc/master/ratings_train.txt
```

```
CPU times: user 0 ns, sys: 20 ms, total: 20 ms
Wall time: 321 ms
```

In [23]:

```
import codecs
with codecs.open("ratings_train.txt", encoding='utf-8') as f:
    data = [line.split('₩t') for line in f.read().splitlines()]
    data = data[1:]   # header 제외

docs = [row[1] for row in data]
len(docs)
```

Out[23]:

```
150000
```

In [24]:

```
import warnings
warnings.simplefilter("ignore")

from konlpy.tag import Okt

tagger = Okt()

def tokenize(doc):
    tokens = ['/'.join(t) for t in tagger.pos(doc)]
    return tokens

tokenize("그 영화는 아주 재미있었어요.")
```

Out[24]:

```
['그/Noun', '영화/Noun', '는/Josa', '아주/Noun', '재미있었어요/Adjective', './Punctu
ation']
```

In [25]:

```
from tqdm import tqdm
sentences = []
for d in tqdm(docs):
    tokens = tokenize(d)
    bigram = ngrams(tokens, 2, pad_left=True, pad_right=True, left_pad_symbol="SS", right_pad_symb
    sentences += [t for t in bigram]
```

100%|██████████| 150000/150000 [03:59<00:00, 625.85it/s]

In [26]:

```
sentences[:30]
```

Out[26]:

```
[('SS', '아/Exclamation'),
 ('아/Exclamation', '더빙/Noun'),
 ('더빙/Noun', '../Punctuation'),
 ('../Punctuation', '진짜/Noun'),
 ('진짜/Noun', '짜증나네요/Adjective'),
 ('짜증나네요/Adjective', '목소리/Noun'),
 ('목소리/Noun', 'SE'),
 ('SS', '흠/Noun'),
 ('흠/Noun', '.../Punctuation'),
 ('.../Punctuation', '포스터/Noun'),
 ('포스터/Noun', '보고/Noun'),
 ('보고/Noun', '초딩/Noun'),
 ('초딩/Noun', '영화/Noun'),
 ('영화/Noun', '줄/Noun'),
 ('줄/Noun', '..../Punctuation'),
 ('..../Punctuation', '오버/Noun'),
 ('오버/Noun', '연기/Noun'),
 ('연기/Noun', '조차/Josa'),
 ('조차/Josa', '가볍지/Adjective'),
 ('가볍지/Adjective', '않구나/Verb'),
 ('않구나/Verb', 'SE'),
 ('SS', '너/Modifier'),
 ('너/Modifier', '무재/Noun'),
 ('무재/Noun', '밋었/Noun'),
 ('밋었/Noun', '다그/Noun'),
 ('다그/Noun', '래서/Noun'),
 ('래서/Noun', '보는것을/Verb'),
 ('보는것을/Verb', '추천/Noun'),
 ('추천/Noun', '한/Josa'),
 ('한/Josa', '다/Adverb')]
```

In [27]:

```
cfd = ConditionalFreqDist(sentences)
cpd = ConditionalProbDist(cfd, MLEProbDist)

def korean_most_common(c, n, pos=None):
    if pos is None:
        return cfd[tokenize(c)[0]].most_common(n)
    else:
        return cfd["/".join([c, pos])].most_common(n)
```

In [28]:

```
korean_most_common("나", 10)
```

Out[28]:

```
[('는/Josa', 831),
 ('의/Josa', 339),
 ('만/Josa', 213),
 ('에게/Josa', 148),
 ('에겐/Josa', 84),
 ('랑/Josa', 81),
 ('한테/Josa', 50),
 ('참/Verb', 45),
 ('이/Determiner', 44),
 ('와도/Josa', 43)]
```

In [29]:

```
korean_most_common("의", 10)
```

Out[29]:

```
[('영화/Noun', 19),
 ('연기/Noun', 14),
 ('구심/Noun', 12),
 ('모습/Noun', 9),
 ('감독/Noun', 8),
 ('매력/Noun', 7),
 ('감동/Noun', 7),
 ('흐름/Noun', 6),
 ('그/Noun', 6),
 ('이야기/Noun', 6)]
```

In [30]:

```
korean_most_common(".", 10, "Punctuation")
```

Out[30]:

```
[('SE', 26503),
 ('영화/Noun', 667),
 ('이/Noun', 565),
 ('정말/Noun', 480),
 ('그리고/Conjunction', 455),
 ('./Punctuation', 445),
 ('하지만/Conjunction', 369),
 ('이/Determiner', 352),
 ('그/Noun', 325),
 ('스토리/Noun', 317)]
```

In [31]:

```
def korean_bigram_prob(c, w):
    context = tokenize(c)[0]
    word = tokenize(w)[0]
    return cpd[context].prob(word)
```

In [32]:

```
korean_bigram_prob("이", "영화")
```

Out[32]:

0.4010748656417948

In [33]:

```
korean_bigram_prob("영화", "이")
```

Out[33]:

0.00015767585785521414

In [34]:

```
def korean_generate_sentence(seed=None, debug=False):
    if seed is not None:
        import random
        random.seed(seed)
    c = "SS"
    sentence = []
    while True:
        if c not in cpd:
            break

        w = cpd[c].generate()

        if w == "SE":
            break

        w2 = w.split("/")[0]
        pos = w.split("/")[1]

        if c == "SS":
            sentence.append(w2.title())
        elif c in ["`", "₩", "'", "("]:
            sentence.append(w2)
        elif w2 in ["'", ".", ",", ")", ":", ";", "?"]:
            sentence.append(w2)
        elif pos in ["Josa", "Punctuation", "Suffix"]:
            sentence.append(w2)
        elif w in ["임/Noun", "것/Noun", "는걸/Noun", "릴때/Noun",
                   "되다/Verb", "이다/Verb", "하다/Verb", "이다/Adjective"]:
            sentence.append(w2)
        else:
            sentence.append(" " + w2)
        c = w

        if debug:
            print(w)

    return "".join(sentence)
```

In [35]:

```
korean_generate_sentence(0)
```

Out[35]:

'미키짱과 말도 전혀 빗나가지 않던 전개로 꽥꽥대는거 보니까 요^^'

In [36]:

```
korean_generate_sentence(1)
```

Out[36]:

'내용 일테인데 이 영화 최고의 암살 할려고 한 데 선배랑 김선아 연기도 크다. 배슬기 여배우도 있는 척 하는거지?'

In [37]:

```
korean_generate_sentence(2)
```

Out[37]:

'도리까지 본 영화 너무... 뭔가.. 최고네요. 하지만.. 눈물 낫다는건 또 영화에 들지 않는다. 근데 뭐야 어떻게 그렇게 착했던 윤재랑은 에바 그린 드레스 소리 듣는거임""" 에리욧의 미모로 합성 한 가수 노래와 흥행 놓친 영화다. 사투리 연기 하나 없는 ‘ 스피드 감 넘치는 스릴 넘치는 연기를 이해 되지 못 하시는 분보다 훨 재밌구만 평점을 망쳐 놓은 듯하다. 영화 보는이로 하여금 불편함을 느꼇을듯'

In [38]:

```
korean_generate_sentence(3)
```

Out[38]:

'내 인생을 반헬싱이 너무 무섭고 재밌고, 칼 세이건으로 연탄가스 맡아서 죽을 같이 작업하는구나 ㅋㅋㅋㅋㅋ 진짜'

In [39]:

```
korean_generate_sentence(5)
```

Out[39]:

'좋았어요... ㅎㄷㄷㄷ 시나리오나 그래픽이 대단한 심리전이 미라 파스틱 함.. 너무 무섭고 나쁜세 끼는 듯 진짜 꼭 필요가 있는지도 모르겠지만 나름 그의 복수 후!!!!!!!!!!!'

확률론적 언어 모형의 활용

확률론적 언어 모형은 다음과 같은 분야에 광범위하게 활용할 수 있다.

- 철자 및 문법 교정(Spell Correction)
- 음성 인식(Speech Recognition)
- 자동 번역(Machine Translation)
- 자동 요약(Summarization)
- 챗봇(Question-Answering)