자연어처리 – 과제 #2

202204495 홍창희

1. 0~9 범위의 10개의 숫자를 인식하는 시스템의 경우의 perplexity를 유도하라.

$[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$

10개 중 1개 → $\frac{1}{10}$의 확률

$$\Rightarrow \boxed{PPL(x) = \left(\frac{1}{10}\right)^{10\left(-\frac{1}{10}\right)} = 10}$$

2. 다음 문장에 대해 unigram, 2-gram, 3-gram 방식에 의해 perplexity를 계산하는 수식을 유도하라.

문장: There is a major problem with the maximum likelihood estimation process

문장을 W, 하나의 단어를 w, 문장의 길이를 N이라고 하고, $P(W)$는 문장의 확률이라 하겠다.

unigram)

$$P(W) = P(w_1, w_2, \cdots, w_n) = \prod_{i=1}^{n} P(w_i)$$

$$= P(\text{There}) \cdot P(\text{is}) \cdot P(a) \cdot P(\text{major}) \cdot P(\text{problem}) \cdot P(\text{with}) \cdot P(\text{the}) \cdot P(\text{maximum}) \cdot P(\text{likelihood}) \cdot P(\text{estimation}) \cdot P(\text{process})$$

$$PPL(W) = P(w_1, w_2, \cdots, w_n)^{-\frac{1}{N}}$$

$$= \left(P(\text{There}) \cdot P(\text{is}) \cdot P(a) \cdot \cdots \cdot P(\text{process})\right)^{-\frac{1}{11}}$$

$$= \sqrt[11]{\frac{1}{P(\text{There}) \cdot P(\text{is}) \cdot P(a) \cdot \cdots \cdot P(\text{process})}}$$

2-gram) $P(W) = P(w_1, w_2, \cdots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-1})$

$= P(\text{is} \mid \text{There}) \cdot P(\text{a} \mid \text{is}) \cdot P(\text{major} \mid \text{a}) \cdot P(\text{problem} \mid \text{major})$
$\cdot P(\text{with} \mid \text{problem}) \cdot P(\text{the} \mid \text{with}) \cdot P(\text{maximum} \mid \text{the})$
$\cdot P(\text{likelihood} \mid \text{maximum}) \cdot P(\text{estimation} \mid \text{likelihood}) \cdot P(\text{process} \mid \text{estimation})$

$PPL(W) = P(w_1, w_2, \cdots, w_n)^{-\frac{1}{N}}$

$= \left(P(\text{is} \mid \text{There}) \cdot P(\text{a} \mid \text{is}) \cdot \cdots \cdot P(\text{process} \mid \text{estimation}) \right)^{-\frac{1}{11}}$

$= \sqrt[11]{\frac{1}{P(\text{is} \mid \text{There}) \cdot P(\text{a} \mid \text{is}) \cdot \cdots \cdot P(\text{process} \mid \text{estimation})}}$

3-gram) $P(W) = P(w_1, w_2, \cdots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1})$

$= P(\text{a} \mid \text{There is}) \cdot P(\text{major} \mid \text{is a}) \cdot P(\text{problem} \mid \text{a major}) \cdot P(\text{with} \mid \text{major problem})$
$\cdot P(\text{the} \mid \text{problem with}) \cdot P(\text{maximum} \mid \text{with the}) \cdot P(\text{likelihood} \mid \text{the maximum})$
$\cdot P(\text{estimation} \mid \text{maximum likelihood}) \cdot P(\text{process} \mid \text{likelihood estimation})$

$PPL(W) = P(w_1, w_2, \cdots, w_n)^{-\frac{1}{N}}$

$= \left(P(\text{a} \mid \text{There is}) \cdot P(\text{major} \mid \text{is a}) \cdot \cdots \cdot P(\text{process} \mid \text{likelihood estimation}) \right)^{-\frac{1}{11}}$

$= \sqrt[11]{\frac{1}{P(\text{a} \mid \text{There is}) \cdot P(\text{major} \mid \text{is a}) \cdot \cdots \cdot P(\text{process} \mid \text{likelihood estimation})}}$

3. 일정한 text data를 이용하여 n-gram 시스템을 훈련시켰을 때의 perplexity를 고려한다.

a. n이 증가할수록 perplexity가 감소하는 이유를 설명하라.

이유를 설명하기 전에 perplexity와 n-gram에 대해 먼저 설명하겠다.

Perplexity란 언어 모델을 평가하기 위한 평가 지표로, "헷갈리는 정도"로 이해하면 좋다.
Perplexity가 낮을수록 언어 모델의 성능이 좋은 것이고, 높을수록 성능이 낮은 것이다.
그리고, N-gram 언어 모델에선 등장하는 단어 중의 일부 단어만 고려하는데, 여기서 n은 그 고려하는 단어의 수를 의미한다.

N-gram 언어 모델은 전체 문장을 고려한 언어 모델 보다는 정확도가 떨어질 수 밖에 없는데, n-gram은 앞의 단어 몇 개만 고려하다 보니 문장이 의도하는 대로 다음 단어를 예측하지 못하는 경우가 생기기 때문이다. 아래의 예시를 통해 설명하겠다.

문장 'An adorable little boy is spreading' 뒤에 올 단어를 4-gram 언어 모델을 통해 예측하면,
작고 사랑스러운(An adorable little) 이라는 수식어를 제거하고, 반영하지 않게 된다.

그러므로 원래 전체 문장의 의도 대로 예측되지 않는 확률이 증가할 수 밖에 없다.
원래 문장의 의도대로라면 뒤에 "smile"(웃음)이 나와야 자연스러울 테지만, 뒤에 "insults"(모욕)이 나올 확률이 증가한다는 뜻이다.

이 문제에서 'n이 증가할수록' 이란 n-gram 언어 모델에서 고려하는 단어의 수가 전체 문장의 단어의 수와 가까워진다는 의미이기에, n이 증가할수록 perplexity가 감소하여 예측 성능이 좋아질 수 밖에 없다.

b. Perplexity를 계산할 때 사용 단어군을 고정시켜야 하는 이유는 무엇인가?

1, 모델에서 계산된 퍼플렉시티는 단어들 간의 확률적 관계를 반영한 성능 평가 지표이다.
만약 사용하는 단어군이 동적으로 변화하면, 어떤 문장에서 새로 등장하는 단어들이 모델에서 전혀 학습되지 않은 단어일 수 있고, 이는 해당 단어들의 확률을 적절하게 계산하지 못하게 만든다.
고정된 어휘 내에서만 확률을 계산함으로써 이러한 문제를 방지할 수 있다.

2, 모델의 perplexity는 동일한 단어군을 사용해야만 서로 다른 n-gram 모델 간의 성능 비교가 공정하게 이루어질 수 있다 예를 들어, 한 모델이 어휘가 고정된 상태에서 평가된 퍼플렉시티와, 다른 모델이 어휘를 확장하거나 변경하면서 평가된 퍼플렉시티를 비교하는 것은 의미가 없다.
이를 일관성 있게 하기 위해 어휘를 고정하는 것이 중요하다.

4. 다음 프로그램은 nltk 패키지를 이용하여 n-gram을 구하는 프로그램이다.

```
from nltk import ngrams

sentence = 'this is a foo bar sentences and i want to ngramize it'

n = 6
sixgrams = ngrams(sentence.split(), n)

for grams in sixgrams:
    print(grams)
```

또한 brown corpus 데이터를 읽어오는 프로그램은 다음과 같다.

```
brown = nltk.corpus.brown
corpus = [word.lower() for word in brown.words()]
```

brown corpus 에서 모든 3-gram을 구한 다음, 어떤 문장의 perplexity를 계산하는 프로그램은 어떻게 구현해야 하는지 설명하라. 단, 최종 단계로 구현하지 않아도 된다.

---

perplexity를 계산하려면 3-gram의 빈도를 count 해야하기 때문에, 수업에서 배운 Counter를 사용하겠다.

```
from nltk import ngrams
from collections import Counter
import nltk
```

주어지는 문장으로는 문제에서 주어진 sentence를 사용하겠다.

```
sentence = 'He went to the store for groceries.'
```

(1) brown corpus에서 3-gram과 2-gram 생성, 등장 횟수 계산

brown corpus에서 모든 3-gram과 2-gram을 추출하고, 각 3-gram과 2-gram의 등장 횟수를 계산한다.

```
trigrams_in_corpus = list(ngrams(corpus, 3))
bigrams_in_corpus = list(ngrams(corpus, 2))

trigram_counts = Counter(trigrams_in_corpus)
bigram_counts = Counter(bigrams_in_corpus)
```

(2) 주어진 문장에서 3-gram과 2-gram 생성

주어진 문장에서도 동일하게 3-gram과 2-gram을 생성한다. 문장의 각 3-gram에 대한 조건부 확률을 계산하기 위해 2-gram도 함께 필요하다.

```
sentence_words = sentence.lower().split()
trigrams_in_sentence = list(ngrams(sentence_words, 3))
bigrams_in_sentence = list(ngrams(sentence_words[:-1], 2))
```

(3) 조건부 확률을 통한 Perplexity 계산

Perplexity를 계산하기위해 우선 문장의 확률 변수와 perplexity 변수를 선언하고, 문장의 3-gram의 개수를 N으로 저장했다.

```
sentence_probability = 1
perplexity = 1
N = len(sentence_words)  # 문장의 길이
```

3-gram과 그에 해당하는 2-gram을 동시에 처리하기 위해서는 리스트에서 항목과 그 항목의 인덱스 i가 함께 반환되어야 했는데, 이를 위해 enumerate 함수를 사용했다.

```
for i, trigram in enumerate(trigrams_in_sentence):
    bigram = bigrams_in_sentence[i]  # 대응하는 2-gram

    trigram_count = trigram_counts[trigram]  # 해당 3-gram의 등장 횟수
    bigram_count = bigram_counts[bigram]  # 해당 2-gram의 등장 횟수

    # 2-gram의 등장 횟수가 0이면, 조건부 확률이 정의되지 않으므로 예외를 발생시켜 프로그램 종료
    if bigram_count == 0:
        raise ValueError("문장의 확률이 정의되지 않음")

    # 조건부 확률 계산: P(w3 | w1, w2)
    conditional_probability = trigram_count / bigram_count

    # 조건부 확률이 0일 경우 예외를 발생시켜 프로그램 종료
    if conditional_probability == 0:
        raise ValueError("문장의 확률이 0이 됨")

    # 조건부 확률들을 곱해나가 문장의 확률 P(W)를 만들어감
    sentence_probability *= conditional_probability
```

perplexity는 문장의 확률의 역수에 N-제곱근으로 정의되므로 아래와 같이 계산한다.

```
perplexity = (1/sentence_probability) ** (1 / N)
```

 # 결과 출력

```
print(perplexity)
```