3장 언어 모델(Language Model)

LM의 정의, 통계적 LM, N-gram LM, 한국어 LM,
Perplexity에 대해 알아봅시다!

2022년도 동계 연구 인턴 2주차 스터디
박성호

3-1 언어 모델의 정의

단어 또는 단어 시퀀스(문장)에 확률을 할당하는 모델

- 어떻게 가장 자연스러운 단어 시퀀스를 찾아 낼 수 있을까?
  →**통계**(조건부 확률, 카운트 기반, N-gram) 또는 **인공 신경망**(GPT, BERT)
- 이전 단어들로 다음 단어를 예측 또는 양쪽의 문맥으로 가운데 비어 있는 단어를 예측
- 기계 번역, 오타 교정, 음성 인식에 활용 → 적절한 문장에 높은 확률(P) 할당
  - EX) P(나는 있는 상태다 버스안에) < P(나는 버스를 탔다)
    P(나는 버스를 탓다) < P(나는 버스를 탔다)
    P(나는 버스를 타따) < P(나는 버스를 탔다)

3-2 통계적 언어 모델

언어 모델의 전통적 접근 방법, SLM

- 조건부 확률

$$P(B|A) = \frac{P(A,B)}{P(A)}\ ,\ P(A,B) = P(A)P(B|A)$$

- 조건부 확률의 연쇄 법칙

$$P(x_1, x_2, \ldots, x_n) =$$
$$P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)P(x_4|x_1, x_2, x_3)\ldots P(x_n|x_1\ldots x_{n-1})$$
$$\rightarrow \prod_{i=1}^{n} P(x_i|x_1\ldots x_{i-1})$$

문장에 대한 확률

- 문장에서 단어는 **문맥이라는 관계**로 인해 **이전 단어의 영향**을 받아 등장하므로 **문장의 확률**은 문장을 이루는 각 단어들을 이전 단어가 주어졌을 때 다음 단어로 등장할 **확률의 곱**, **조건부 확률의 연쇄 법칙**으로 해석할 수 있다.

  "An adorable little boy is spreading smiles" 라는 문장의 확률은

- $$P(Anadorablelittleboyisspreadingsmiles)$$
- $$= P(An) \times P(adorable|An) \times P(little|Anadorable)\ldots$$
  $$\times P(smiles|Anadorablelittleboyisspreading)$$

문장에 대한 확률 - 희소(Sparsity) 문제

- $P(is|Anadorablelittleboy) = \frac{count(Anadorablelittleboyis)}{count(Anadorablelittleboy)}$

만약 훈련 데이터 셋에서 ‘An adorable little boy’라는 시퀀스가 없다면?
→ Division by zero로 정의 할 수 없다!

이렇게 데이터 셋이 **충분하지 못하면** 언어를 정확히 모델링하지 못하는 것을 희소 문제(Sparsity Problem)라고 한다.

Smoothing, Backoff와 같은 일반화 기법이 있지만
희소 문제에 대한 근본적인 해결책이 아니므로 한계를 갖는다.

3-3 N-gram 언어 모델

카운트 기반한 통계적 접근의 SLM, n개의 **일부 단어**만 **고려**하는 것이 특징

- SLM의 한계는 훈련 코퍼스에 확률을 계산하고 싶은 문장이나 단어가 없을 수도 있다..
  → 참고할 단어를 줄인다면? *"An adorable little boy"* 대신
  근사적으로 *"boy"*나 *"little boy"*를 쓴다면 count가 늘어날 수 있다!
  → $P(is|Anadborablelittleboy) \approx P(is|littleboy)$
  - 한계: ① 그래도 count가 0이 될 가능성..
    ② n의 값을 선택하는 것에는 trade-off가 따른다.
    일반적으로 n=2 이면, n=1 보다 성능이 좋지만
    n이 커질 수록 **희소 문제 위험성도 증가**
    그렇다고 작게하면 count는 늘어나겠지만 현실의 확률분포와 멀어진다..
    일반적으로 $n \leqq 5$ 권장

3-4 한국어 언어 모델

한국어는 영어보다 다음 단어를 예측하기가 훨씬 까다롭다.

- **한국어는 상대적으로 어순이 중요하지 않다..**
  나는 체육관에서 운동을 합니다. (O)
  나는 운동을 체육관에서 합니다. (O)
  나는 운동을 합니다 체육관에서. (O)
  체육관에서 운동을 합니다. (O)
  - **한국어는 교착어다. (중국어는 고립어, 영어는 굴절어+고립어)**
    교착어: 단어에 접사가 붙어 의미가 결정,
    ex) ’나’라는 어근 뒤에 ‘은/는/이/가/‘가 붙으면 주격, ‘을/를’이 붙으면 목적격
    동사에서는 어근에 접사가 붙어 시제를 결정 ‘가다’, ‘갔었다’, ‘갔다’
    →’가+시+었+겠+더+라’((어근+주체높임+과거+추측+회상)+어미)😅
  - **띄어쓰기 규칙이 제대로 지켜지지 않는다..**

영어보다 한국어 자연어 처리가 어려운 이유

- 평서문과 의문문이 같은 구조를 가지므로 물음표나 마침표가 필요한 경우가 많다

  I ate my lunch (점심 먹었어.)
  Did you have lunch? (점심 먹었어?)

- 영어에서는 특별한 경우를 제외하면 주어 정보가 포함되지만
  한국어에서는 동사가 중요시돼 주어는 자주 생략된다.
  I ate my lunch (___ 점심 먹었어.) → 맥락상 파악

- 한국어는 한자(표어 문자)와 표음 문자를 섞어 사용하므로
  맥락을 이용하는 인간에게는 효율적이지만 컴퓨터에게는 모호성의 문제를 가진다.

  Concentrate → con(=together) + centr(+center) + ate(=make)
  집중 → 集(모을 집) + 中(가운데 중)

3-5 Perplexity(PPL)

언어 모델의 평가 지표로 낮을 수록 좋은 성능을 의미한다.

- PPL은 문장의 길이로 정규화 된 문장 확률의 역수이다.
  정규화:문장이 길어지면 문장 확률이 작아지므로
  문장의 길이를 제곱근으로 기하 평균을 구하는 것

문장 W의 길이가 N일 때, $PPL(W) = \sqrt[N]{\frac{1}{P(w_1,w_2,w_3,\ldots,w_N)}}$

즉, $PPL(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(x_i|x_1 \ldots x_{i-1})}} = \prod_{i=1}^{N} P(x_i|x_1 \ldots x_{i-1})^{-\frac{1}{N}}$

분기 계수(Branching factor)

- PPL은 선택 가능한 경우의 수를 의미하는 **분기 계수**로,
  언어 모델이 특정 시점에서 평균적으로 몇 개의 선택지를 가지고
  다음 단어를 고민하고 있는지를 의미한다.
- 예를 들어 1부터 6까지 균등하게 나오는 6면 주사위를 던져
  얻은 숫자로 수열을 만드는데 주사위를 N번 던질때 PPL은

$$\left(\frac{1}{6}\right)^{\frac{-1}{N}} = 6^{N}$$

  즉, time-step 마다 6가지 경우의 수를 가지고 고민하고 있다고 해석한다

4장 카운터 기반의 단어 표현

다양한 단어의 표현 방법, BoW, DTM, TF-IDF에 대해 알아봅시다!

2022년도 동계 연구 인턴 2주차 스터디
박성호

4-1 다양한 단어의 표현

크게 국소 표현 방법과 분산 표현 방법으로 나눌 수 있다

- 국소 표현 방법(Local Representation)은 해당 단어 자체만 보고 값을 맵핑하는 것으로

  One-hot 벡터, N-gram, BoW(Bag of words), DTM(Document-Term Matrix)
  등이 있고 대표적 통계량은 TF-IDF(Term Frequency-Inverse Document Frequency) 이다.

- 분산 표현 방법(Distributed Representation)은 단어 주변을 참고해 단어의 의미와 늬앙스를 표현 하는 것으로

  Word2Vec, FastText, LSA(Latent Semantic Analysis), LDA(Latent Dirichlet Allocation), Glove
  등이 있고 대표적 통계량은 PMI(Pointwise Mutual Information)

4-2 Bag of Words(BoW)

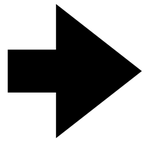
언어 모델과 반대되는 가정으로 단어의 등장 순서는 고려하지 않고 **빈도수만 고려**한다.

- 가정: ‘문서의 주제는 단어 사용에 녹아있어
  유사한 주제라면 단어 빈도나 등장 여부가 비슷할 것이므로
  BoW 임베딩 역시 비슷할 것이다’는 것이다.
- 과정: ① 각 단어에 고유한 정수 인덱스를 부여해 단어 집합을 만든다.
  ② 각 인덱스 위치에 단어 토큰의 등장 횟수를 기록한 벡터를 만든다.

4-3 Document-Term Matrix

서로 다른 문서들의 BoW들을 결합해 **행렬로 표현**한다.

- D1 = “I like the apple”
  D2 = “I dislike apple apple”



	I	like	dislike	the	apple
D1	1	1	0	1	1
D2	1	0	1	0	2

간단한 방식이지만 임베딩 벡터 차원은 문서 전체 단어 집합만큼 커야하고
값이 0인 희소 벡터는 저장 공간과 연산에 부담이 된다

→ 불용어 제거 또는 단어의 정규화, TF-IDF등으로 정제

4-4 TF-IDF(Term Frequency-Inverse Document Frequency)

문서 내 각 단어에 대한 **중요도**를 계산할 수 있는 **가중치**

- 특정 문서에서 자주 등장하는 단어는 그 문서에서 중요한 단어 일 것이다.
  그런데 다른 모든 문서들에서도 등장한다면 중요하지 않은 단어 일 것이다.
- tf(d,t): 특정문서 d에서 특정단어 t의 등장 횟수
- df(t): 특정단어 t가 등장한 문서의 수
- idf(d,t): df(t)에 반비례하는 수로, df(t)를 총 문서의 수 n 으로
  나눈 값의 역수를 구한 뒤 Log를 사용해 스케일을 변경 후
  $log(\frac{n}{df(t)})$ 분모가 0이 되지 않도록 1을 더해
  $\rightarrow idf(d,t) = log(\frac{n}{1+df(t)})$ TF-IDF는 idf(d,t)에 df(t)를 곱해서 얻는다!

5장 벡터 유사도

코사인 유사도, 여러가지 유사도 기법에 대해 알아봅시다!

2022년도 동계 연구 인턴 2주차 스터디
박성호

5-1 코사인 유사도(Cosine Similarity)

두 벡터간의 코사인 값을 구해 유사도를 구한다

- 두 벡터의 방향이 동일하면 1, 직교(orthogonal)하면 0, 반대의 방향을 가지면 -1
  즉 코사인 유사도는 -1 이상 1 이하의 값을 가지며 1에 가까울 수록 유사하다고 본다.$similarity = cos(\Theta) = \frac{A \cdot B}{\|A\|\|B\|}$
- D1: “나는 사과가 좋아”,
  D2: “나는 바나나가 좋아”,
  D3: “나는 바나나가 좋아 나는 바나나가 좋아”
  (D1, D2)의 sim = **0.67**, (D1, D3)의 sim = **0.67**, (D2, D3)의 sim = **1**
  즉 문서의 길이의 영향을 받지 않고 **방향(패턴)만을 고려**한다.

5-2 여러 가지 유사도 기법(1)

유클리드 거리: 다차원 공간에서 점과 점 사이로 계산

- 문서 $p(p_1, p_2, \ldots p_n)$과 문서 $q(q_1, q_2, .., q_n)$의

  유클리드 거리는 $\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$ 계산 (* Python에서 math.dist()와 동일)

  유클리드 거리의 값이 **작을 수록** 두 가지 문서의 **유사도는 높다**고 해석한다.

5-2 여러 가지 유사도 기법(2)

자카드 유사도: 두 집합의 합집합에서 교집합의 비율로 계산

- 비교할 두 개의 문서 doc_1, doc_2의 Jaccard similarity $J(doc_1, doc_2)$는 $\frac{doc_1 \cap doc_2}{doc_1 \cup doc_2}$
  - Python 구현: tokenized_doc1 = doc1.split(), tokenized_doc2 = doc2.split()
    union = set(tokenized_doc1).union(set(tokenized_doc2))
    intersection = set(tokenized_doc1).intersection(set(tokenized_doc2))

    → J(doc1, doc2) = len(intersection)/len(union)

감사합니다.