

Assignment 1 - Part A

융합대학원 SDS
김다영 (202429번)

1. Foundations

1.1 ① 언어 모델링에서 엔트로피를 정의.

엔트로피는 언어 모델에서 텍스트 데이터를 예측할 때 모델이 느끼는 "불확실성"의 정도를 수학적으로 나타낸 척도.

정보 이론에서 도입된 개념으로, 확률 분포에서 발생할 수 있는 사건의 평균 정보량을 측정한다.

즉, 언어 모델에서 단어 사전의 불확실성을 측정할 때 사용되고 언어 모델이 특정 단어 사전에서 다음 단어를 얼마나 정확히 예측할 수 있는지를 측정하는 척도.

② 언어 모델의 불확실성을 어떻게 측정하는가?

$$\text{엔트로피 } H = - \sum_{w \in V} P(w) \log P(w)$$

$\Rightarrow V$: 어휘 (vocabulary) / 모델이 예측할 수 있는 모든 단어의 집합

$P(w)$: 단어 w 가 등장할 확률

$\log P(w)$: 로그학수 / 비트 단위로 정보량을 측정

i) 높은 엔트로피는 $H(P)$ 가 크다는 뜻으로 모델의 불확실성이 크다는 것을 의미.

이는 다양한 단어가 고르게 등장할 확률이 있을 때, 단어들의 확률이 거의 동일하면 불확실성이 높아져서

모델이 다음에 어느 단어를 예측해야 할지 예측하기 어려운 상황을 나타낸다.

$\Rightarrow H(P)$ 가 크면 불확실성이 크다

ii) 낮은 엔트로피는 모델의 불확실성이 적다는 것을 의미.

이는 특정 단어가 매우 높은 확률을 가지거나 항상 등장한다면 그 단어의 확률 $P(w)$ 는 1에 가까워지고,

$\log P(w)$ 는 0에 가까워져 결국 $H(P)$ 는 0에 가까워짐.

그래서 낮은 엔트로피는 모델의 불확실성이 적다는 것을 의미하고, 다음 단어를 예측하기 쉬운 상황을 나타낸다.

$\Rightarrow H(P)$ 가 작으면 불확실성이 적다.

1.2 "You may say I'm a dreamer, but I'm not the only one"

① Phonetics / Orthography

: 각 단어가 어떻게 발음되고 쓰여지는지 분석

i) 음운론 (Phonetics)

You may say I'm a dreamer, but I'm not the only one

/ju:/ /meɪ/ /seɪ/ /aɪm/ /eɪ/ /'drɪzmər/ /bʌt/ /aɪm/ /nɒt/ /əʊ/ /'oʊnli/ /uən/

ii) 정체법 (Orthography)

\Rightarrow 음성을 문자로 변환한 것

② Phonology (음운론)

: 음장의 소리 패턴을 다룬다. 각 단어가 개별 발음뿐만 아니라, 음장 내에서 어떻게 발음이 변화하는지를 주목.

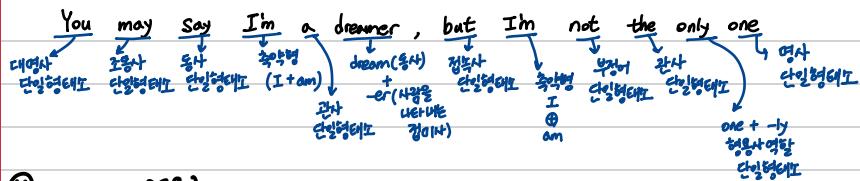
영어와 같은 현상을 다룬다.

I'm a \Rightarrow /əɪmə/

the only \Rightarrow /ðə: 'oʊnlɪ/

③ Morphology (형태론)

: 각 단어의 내부 구조와 그 단어가 어떻게 구성되어 있는지를 분석하고 접미사나 축약형 등을 분석



④ Syntax (구문론)

: 문장 내에서 단어들이 어떻게 문법적으로 배열되는지 분석

· You may say I'm a dreamer, but I'm not the only one

→ 두 개의 절로 나뉘며, "but"이라는 접속사를 연결된 부합문

· You may say

→ 주어 + 조동사 + 동사로 구성된 절

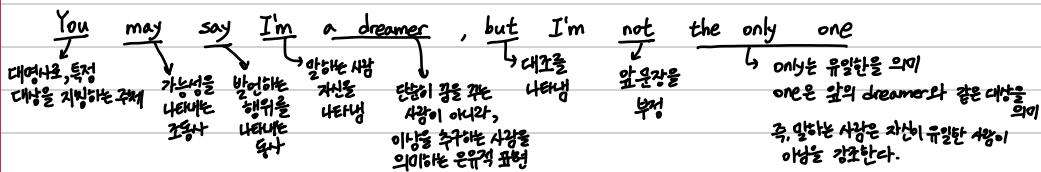
· I'm a dreamer
↳ 주어 + 동사 ↳ 영사구로 "dreamer"가 주어로 사용됨

· but → 등위 접속사

· I'm not the only one : 이 문장의 두 번째 절을 형성.
↳ 주어 + 부정사 ↳ 영사구

⑤ Semantics (의미론)

: 문장의 단어들이 어떤 의미를 가리인지 분석함. 은유적 표현과 문장 전체의 의미 분석.



⑥ Discourse / pragmatics (담화 / 화용론)

: 문맥 속에서 문장이 어떻게 해석되는지, 상황에 따른 의미 변화 분석

· You may say I'm a dreamer, but I'm not the only one

→ 문맥상 이상주의적이고 평화로운 세상을 꿈꾸는 사람들을 표현.

여기서 but은 자신의 그런 꿈을 주는 유일한 사람이 아니라는 점을 강조하는 역할을 한다.

2. Regular Expressions and Edit Distance

2.1

- $[a-zA-Z]^+$
- $[a-zA-Z]^*$
- $(ba(ab)^*)^b$
- $(0(10)^*)^1 \mid (1(01)^*0)$
- $\backslash b(\backslash w+\backslash s+\backslash 1\backslash b$
- ${}^e \backslash d+\backslash s.^*\backslash s[a-zA-Z]^+ \$$

2.2 a. Statistical (10글자) → Deep (4글자)

· Table for first word

	D	E	E	P
O	1	2	3	4
S	1	2	3	4
T	2	2	2	4
A	3	3	3	4
T	4	4	4	4
I	5	5	5	5
S	6	6	6	6
T	7	7	7	7
I	8	8	8	8
C	9	9	9	9
A	10	10	10	10
1	11	11	11	11

- i) S → d : 대체비율 1
- ii) t → e : 대체비율 1
- iii) a → e : 대체비율 1
- iv) t → p : 대체비율 1
- v) i, s, t, f, c, a, l : 삭제
 $\therefore 4(\text{대체}) + 7(\text{삭제}) = 11$

답 11

b. interesting → difficult

interesting : 11글자

difficult : 9글자

i) interest → difficult 대체

$\Rightarrow 8$

ii) ing 4번 $\Rightarrow 3$

$\therefore 8+3=11$

c. Statistical → Deep

interesting → difficult

i) a 답대로 11

ii) b 답대로 11

답 22

$\therefore 11+11=22$

d. Statistical → Deep

i) 대체 4번 $\Rightarrow 5 \times 4 = 20$

ii) 삭제 7번 $\Rightarrow 1 \times 7 = 7$ 답 27

$\therefore 20+7=27$

3. N-gram Language Models

3.1 a. P(hat | the) : "the"가 등장한 후에 "hat"이 등장하는 확률

i) 각 단어의 등장 빈도 계산

· the : 6회	· was : 2회	· a : 1회	· alas : 1회	· cat's : 1회
· hat : 4회	· on : 1회	· blue : 1회	· red : 1회	
· cat : 3회	· i : 1회	· said : 1회	· <s> : 4회	
· wore : 1회	· want : 1회	· but : 1회	· </s> : 4회	

}

V = 17

ii) the 다음에 hat이 나온 횟수 : 2회

$$\therefore P(\text{hat} | \text{the}) = \frac{\text{the 다음 hat 나온 횟수}}{\text{the 등장 횟수}} = \frac{2}{6} = \frac{1}{3} \approx 0.333$$

- b. "Laplace Smoothing" → 단어가 등장하지 않은 경우에도 확률이 0이 되지 않도록 조정하는 방법
이를 위해 모든 단어의 등장 빈도에 1을 더해준다.

$$\text{Laplace Smoothing 적용된 } P(\text{hat} | \text{the}) = \frac{\text{the 다음에 hat 나온 횟수} + 1}{\text{the 총 등장 횟수} + V}$$

↳ 이는 크기 (전체 고유 단어의 수)

$$\therefore P(\text{hat} | \text{the}) = \frac{2 + 1}{6 + 17} = \frac{3}{23} \approx 0.1304$$

C. Good-Turing Smoothing

→ 빈도수가 낮거나 등장하지 않은 단어들의 확률을 조정하여, 낮은 빈도의 단어에 더 높은 확률을 부여하는 방법

i) 단어의 등장 횟수 C

$$C = \frac{(C+1) \cdot N_{C+1}}{N_C}$$

- C : 단어의 원래 등장 횟수
- N_C : 등장 횟수가 C 인 단어의 개수
- N_{C+1} : 등장 횟수가 $C+1$ 인 단어의 개수

ii) $P(\text{hat} | \text{the})$ 계산

- 1) the 다음에 hat이 등장한 횟수 : 2

$$\Rightarrow C = 2$$

- 2) N_1 : 등장 횟수가 1인 단어의 개수 : 11 (wore, one, i, want, a, blue, said, but, alas, red, cat)

$$N_2 : 1 (\text{was})$$

$$N_3 : 1 (\text{cat})$$

iii) Good-Turing 스무딩 적용

$$C^* = \frac{(2+1) \cdot N_3}{N_2} = \frac{3}{1} = 3$$

- .. 적용 후 the 다음에 hat이 나오는 새로운 빈도 = 3

iv) $P(\text{hat} | \text{the})$ 최종 계산

$$P(\text{hat} | \text{the}) = \frac{C^*}{\text{the 총 횟수}} = \frac{3}{6} = \frac{1}{2} = 0.5$$

d. i) 작은 규모의 코퍼스 상황

: Laplace Smoothing 이 Good-Turing Smoothing 보다 더 적합.

그리풀러스 스무딩은 작은 데이터셋에서 단어가 등장하지 않았을 때 확률이 0이 되지 않도록 조정하는데 적합.
데이터가 제한적인 경우 단어들이 한번도 등장하지 않는 상황이 자주 발생하여 그리풀러스 스무딩을 적용하여 모든 단어에 일정한 확률을 부여가 가능함.

Ex) 학습 데이터가 적고 다양한 단어가 충분히 등장하지 않은 경우. 예를 들어 작은 규모의 문서에서 다음과 같이 단어를 예측할 때

⇒ 이에 반해, Good-Turing 스무딩은 작은 데이터셋에서는 정확한 확률 추정이 어려울 수 있으며, 등장하지 않은 단어의 확률을 고의로 높이는 문제가 발생할 수 있음. 또한, 수학적으로 계산이 복잡하여, 노트북 데이터셋에서 사용하면 오히려 성능 저하될 수 있음.

Ex) 데이터가 매우 적고 단어 빈도 분포가 영향하지 않은 상황. 예를 들어 몇개의 문장만 있는 작은 데이터셋에서는 Good-Turing이 잘 작동하지 않을 수 있음.

ii) 대규모 코퍼스 상황

: Good-Turing Smoothing이 Laplace Smoothing 보다 더 적합함.

- ⇒ Good-Turing 스무딩은 주로 낮은 인도수 등장한 단어나 한번도 등장하지 않은 학률을 더 적절하게 조정하는데 사용됨.
 자주 등장하지 않은 단어가 많은 상황에서 유리하여 대규모 데이터셋에서 드물게 등장하는 단어들의 학률을 추정하는데 적합.
 특히, 등장하지 않은 단어에 대해 적절한 학률을 할당할 수 있으므로, 회도성 문제를 해결하는데 유리!
- Ex) 뉴스 기사와 같은 대규모 코퍼스에서, 자주 등장하지 않는 고유명이나 특수한 단어들에 학률을 부여하는 데 Good-Turing 스무딩이 적합. 단어의 실제 빈도에 더 잘 맞춰 학률을 조정해줌.
- ⇒ 이에 반해, 라플라스 스무딩은 모든 단어에 동일한 학률을 더해주므로, 큰 데이터셋에서는 자주 등장하는 단어의 학률을 과도하게 낮출 수 있음. 이로 인해 데이터의 불포화를 더해줄 수 있으며, 대규모 데이터셋에서는 덜 적합함.
- Ex) 두백만 개의 풍경으로 구성된 대규모 코퍼스에서 라플라스 스무딩을 적용하면 자주 등장하는 단어들조차 불필요하게 낮은 학률을 갖게 되어 모델 성능이 저하될 수 있음.

3.2 i) Trigram language model

: 세 개의 연속된 단어를 기준으로, 앞의 두 단어가 주어졌을 때 다음 단어가 나올 학률을 계산하는 모델

$$P(W_3|W_1, W_2) = \frac{C(W_1, W_2, W_3)}{C(W_1, W_2)} \rightarrow \text{trigram 빈도수}$$

$$\qquad\qquad\qquad C(W_1, W_2) \rightarrow \text{bigram 빈도수}$$

ii) Laplace smoothing 적용

$$P(W_3|W_1, W_2) = \frac{C(W_1, W_2, W_3) + 1}{C(W_1, W_2) + V}$$

- $C(W_1, W_2, W_3) + 1$
⇒ 어휘의 크기, 즉 고유한 단어의 총 개수
- $C(W_1, W_2) + V$
⇒ 실제 trigram 빈도수에 1을 더하여, 해당 trigram이 한번도 등장하지 않은 경우에도 최소한의 학률을 부여.

∴ formula for estimating $P(w_3|w_1, w_2)$ with Laplace Smoothing

$$\Rightarrow P(W_3|W_1, W_2) = \frac{C(W_1, W_2, W_3) + 1}{C(W_1, W_2) + V}$$

4. Text Classifications

4.1 ① Naive binarized Bayes Classifier

i) Bayes Rule

$$P(C|D) = \frac{P(D|C) \cdot P(C)}{P(D)} \xrightarrow{\text{dropping the denominator}} P(C|D) \propto P(C) \cdot P(D|C)$$

ii) Naive Bayes Assumptions

→ assume the probabilities $P(D|C)$ independent

$$P(D|C) = P(w_1|C), P(w_2|C), \dots, P(w_n|C) = \prod_{i=1}^n P(w_i|C)$$

$$\therefore P(C|D) \propto P(C) \cdot \prod_{i=1}^n P(w_i|C)$$

각 단어 w_i 가 클래스 C 에서 나타날 확률

② 사건 확률 (prior Probability)

· Class(Comedy)

- "fun, couple, love, love"
- "couple, fly, fast, fun, fun"

· Class(Action)

- "fast, furious, shoot"
- "furious, shoot, shoot, fun"
- "fly, fast, shoot, love"

$$P(\text{Comedy}) = \frac{2}{5}$$

$$P(\text{Action}) = \frac{3}{5}$$

③ Likelihood 계산 with add-1 smoothing

$$P(W_i | C) = \frac{C(W_i, C) + 1}{C(C) + V}$$

- $C(W_i, C)$ = 클래스 C에서 단어 W_i 가 등장한 횟수
- $C(C)$ = 클래스 C에 등장한 모든 단어의 총 횟수
- V : 어휘 크기 (고유 단어의 수)

i) Comedy 클래스

- 고유단어 : {"fun", "couple", "love", "fly", "fast"}

단어빈도

- "fun": 3
- "couple": 2
- "love": 2
- "fly": 1
- "fast": 1

$$\Rightarrow \text{Comedy 클래스의 총 단어 수} : 3+2+2+1+1 = 9$$

ii) Action 클래스

- 고유단어 : {"fast", "furious", "shoot", "fun", "fly", "love"}

$$\text{단어빈도: } 2 \quad 2 \quad 4 \quad 1 \quad 1 \quad 1 \Rightarrow \text{Action 클래스 총 단어 수} : 11$$

iii) 전체 V 크기 : 7개 (fun, couple, love, fly, fast, furious, shoot)

④ 계산 (add-1 smoothing 적용)

$$P(W_i | \text{Comedy}) = \frac{C(W_i, \text{Comedy}) + 1}{C(\text{Comedy}) + V}$$

$$P(W_i | \text{Action}) = \frac{C(W_i, \text{Action}) + 1}{C(\text{Action}) + V}$$

i) Comedy Class

- $P(\text{fast} | \text{Comedy}) = \frac{1+1}{9+7} = \frac{2}{16}$
- $P(\text{couple} | \text{comedy}) = \frac{2+1}{9+7} = \frac{3}{16}$
- $P(\text{shoot} | \text{comedy}) = \frac{0+1}{9+7} = \frac{1}{16}$
- $P(\text{fly} | \text{comedy}) = \frac{1+1}{9+7} = \frac{2}{16}$

ii) Action Class

- $P(\text{fast} | \text{Action}) = \frac{2+1}{11+7} = \frac{3}{18}$
- $P(\text{couple} | \text{Action}) = \frac{0+1}{11+7} = \frac{1}{18}$
- $P(\text{shoot} | \text{Action}) = \frac{4+1}{11+7} = \frac{5}{18}$
- $P(\text{fly} | \text{Action}) = \frac{1+1}{11+7} = \frac{2}{18}$

⑤ Posterior Probability 계산

i) Comedy 클래스에 대한 사후 확률

$$P(\text{Comedy} | D) \propto P(\text{comedy}) \cdot P(\text{fast} | \text{Comedy}) \cdot P(\text{couple} | \text{Comedy}) \cdot P(\text{shoot} | \text{comedy}) \cdot P(\text{fly} | \text{comedy})$$

$$P(\text{Comedy} | D) \propto \frac{2}{5} \cdot \frac{2}{16} \cdot \frac{3}{16} \cdot \frac{1}{16} \cdot \frac{2}{16}$$

$$\Rightarrow P(\text{Comedy} | D) \propto \frac{24}{327680}$$

ii) Action 클래스에 대한 사후 확률

$$P(\text{Action} | D) \propto P(\text{action}) \cdot P(\text{fast} | \text{Action}) \cdot P(\text{couple} | \text{Action}) \cdot P(\text{shoot} | \text{Action}) \cdot P(\text{fly} | \text{Action})$$

$$P(\text{Action} | D) \propto \frac{3}{5} \cdot \frac{3}{18} \cdot \frac{1}{18} \cdot \frac{5}{18} \cdot \frac{2}{18}$$

$$\Rightarrow P(\text{Action} | D) \propto \frac{90}{524880}$$

⑥ 최종 결과 비교

$$i) P(\text{Comedy} | D) \propto \frac{24}{327680} \approx 0.0000732$$

$$ii) P(\text{Action} | D) \propto \frac{90}{524880} \approx 0.0001715$$

$$\therefore P(\text{Action} | D) > P(\text{Comedy} | D)$$

class of a new movie that contains "fast, couple, shoot, fly" \Rightarrow Action 클래스

Part B

2. Attach screenshots of the code execution that displays running times (5 points)

```
● (assn1) ubuntu@DESKTOP-P9LT9AI:~/SNLP/assn1$ python pos_tagger.py
100%|██████████| 1387/1387 [00:03<00:00, 418.03it/s]
100%|██████████| 462/462 [00:01<00:00, 391.26it/s]
100%|██████████| 462/462 [00:00<00:00, 772.24it/s]

Training time: 0.67 seconds
Inference Runtime: 0.009521 minutes
Probability Estimation Runtime: 0.009043 minutes
Whole sent acc: 0.015152
Mean Probabilities: -2425.023825
Token acc: 0.942692
Unk token acc: 0.595526
Evaluation time: 1.11 seconds
Inference time: 0.01 seconds
Total execution time: 85.98 seconds
```

3. Choose one mode (unigram/bigram/trigram). Describe your chosen mode's implementation with HMM diagram (10 points)

① Trigram mode 선택

Trigram HMM 모델은 다음과 같은 주요 구성요소로 구성됨

- State : 각 단어의 POS 태그 (T_{-i-2} , T_{-i-1} , T_{-i})
- Transition Probability : 현재 태그가 이전 두 태그에 의해 결정되는 확률 $P(T_i | T_{i-1}, T_{i-2})$
- Emission Probability : 특정 태그에서 단어가 나타날 확률 $P(\text{word} | \text{tag})$

② 코드의 Trigram HMM 구조 & 코드와 다이어그램의 일치 여부 검증

- Transition Probability 계산

```
# trigram 사용하여 Transition Probability 계산
if i == 0:
    trans_prob = self.unigrams[tag_idx] # 첫 태그는 unigram 확률 사용
elif i == 1:
    prev_tag_idx = self.tag2idx[tags[i - 1]] # 이전 태그 인덱스
    trans_prob = self.bigrams[prev_tag_idx, tag_idx] ** self.scaling_factor # bigram 확률
else:
    prev_prev_tag_idx = self.tag2idx[tags[i - 2]] # 이전 이전 태그 인덱스
    prev_tag_idx = self.tag2idx[tags[i - 1]] # 이전 태그 인덱스
    trans_prob = self.trigrams[prev_prev_tag_idx, prev_tag_idx, tag_idx] ** self.scaling_factor # trigram 확률
log_prob += np.log(trans_prob)
```

첫번째 단어에 대해 Unigram 확률 사용
두번째 단어에 대해 Bigram 확률 사용
세번째 단어 이후부터는 Trigram 확률 사용

→ 이 부분에서 Diagram 구조와 일치. Trigram HMM에서는 현재 태그가 이전 두 태그에 의존하고, 코드에서도 이전 두 태그 (T_{-i-2} , T_{-i-1})를 기반으로 현재 태그 (T_{-i})의 확률을 계산

- Emission Probability 계산

```
# Emission probability 계산
if word_idx is not None:
    emission_prob = self.emissions[tag_idx, word_idx]
```

Emissions 배열을 사용하여 각 태그에서 단어가 발생할 확률 계산

→ Diagram에서 각 상태 (T_{-i-2} , T_{-i-1} , T_{-i})가 특정 단어로 연결되는 발산 화살표가 일치. / Diagram에서 태그와 단어 간의 연결이 코드의 emissions-prob 계산과 같은 방식으로 표현됨

③ 결론

- State와 Transition Probability

→ 코드 : 각 태그가 이전 두 태그에 따라 결정되며, 다이어그램에서도 이와 같이 태그 간의 연결이 이전 두 태그를 기반으로 이루어짐.

다이어그램에서 T_{-i-2} , T_{-i-1} , T_{-i} 가 전이 확률 $P(T_i | T_{i-1}, T_{i-2})$ 로 연결될 모습 = 코드의 전이 확률 계산

- Emission Probability

→ 다이어그램 : 각 태그가 특정 단어를 생성하는 확률이 다이어그램의 발산 화살표로 나타남

코드의 emissions-probs 계산 부분과 일치!

4. Attach a screenshot of the F1 score results.

Mean F1 Score
: 0.94

```
(assn1@DESKTOP-P9LT9AI:~/SNLP/assn1$ python evaluate.py -p data/test_y.csv -d data/dev_y.csv  
Mean F1 Score: 0.9423574431545406
```

Classification Report:				
	precision	recall	f1-score	support
#	0.87	1.00	0.93	46
\$	1.00	0.99	1.00	2051
.	1.00	1.00	1.00	1664
(1.00	1.00	1.00	309
)	1.00	1.00	1.00	316
,	1.00	1.00	1.00	12558
.	1.00	1.00	1.00	10078
:	1.00	1.00	1.00	1187
CC	0.99	0.99	0.99	6157
CD	0.99	0.89	0.94	9509
DT	0.99	0.99	0.99	21003
EX	0.91	0.88	0.89	210
FW	0.23	0.27	0.25	22
IN	0.95	0.98	0.96	25237
JJ	0.88	0.83	0.85	15277
JJR	0.77	0.84	0.80	837
JJS	0.89	0.86	0.87	511
LS	0.22	0.40	0.29	10
MD	1.00	0.99	0.99	2494
NN	0.91	0.95	0.93	34400
NNP	0.93	0.96	0.94	23113
NNPS	0.42	0.61	0.49	498
NNS	0.96	0.96	0.96	15415
O	1.00	1.00	1.00	462
PDT	0.21	0.22	0.21	46
POS	0.98	0.99	0.99	2069
PRP	0.99	0.99	0.99	4333
PRP\$	0.99	1.00	0.99	2162
RB	0.91	0.86	0.88	7949
RBR	0.60	0.51	0.55	463
RBS	0.67	0.66	0.66	125
RP	0.58	0.70	0.64	662
SYM	0.63	0.48	0.55	25
TO	1.00	1.00	1.00	5791
UH	0.50	0.12	0.20	24
VB	0.96	0.91	0.93	6790
VBD	0.89	0.93	0.91	7907
VBG	0.88	0.90	0.89	3675
VBN	0.86	0.77	0.81	5220
VBP	0.91	0.90	0.91	3142
VBZ	0.97	0.94	0.95	5296
WDT	0.91	0.52	0.66	1080
WP	0.99	0.99	0.99	614
WP\$	0.97	1.00	0.99	39
WRB	1.00	1.00	1.00	534
``	1.00	1.00	1.00	1711
accuracy			0.94	243021
macro avg	0.85	0.84	0.84	243021
weighted avg	0.94	0.94	0.94	243021