REPORT

ASSIGNMENT-1 ELL881

By. Krishna Kumar Singh (2020MT10814)

1. Generating the sentence using n-grams for n = 1 to m (m= 7)
    - Unigram model generates weird sentences which make no sense. It is obvious as each word is coming independently of the previous words.
    - Bigram model-generated sentences are close to proper English language sentence but doesn't make sense most of the time.
    - Similarly as n increases, the sentence looks more like a proper English sentence beyond n = 5,6, most of the time, the model doesn't generate new sentences but copies the sentence from the training set.

```
PS E:\NLP> python -u "e:\NLP\ngrammodel.py"
generating sentence for: n =  1  :
  time  turrets plunk of that because harry  me mind warmth flat going long looked ,  betrayed close what , source bus  incantatem the ,so room feeling any said
 squeaked been ,a top advantage ,and harry rowling the once ,

generating sentence for: n =  2  :
 and very pleased to overlook dumbledores lying there was happening to keep your acceleration , reminds me i heard me the triwizard tournament because wood was
removed her she did a student yields to see them .

generating sentence for: n =  3  :
 thats more like , who was pouring out of earshot before saying , drawing her attention to what was this why he kept looking back at hogwarts .

generating sentence for: n =  4  :
 shes a lovely person who made a funny movement somewhere between a nod and shrug , and shining .

generating sentence for: n =  5  :
 not seriously ill , mind , but don go rabbitin about it in here , he said shortly .

generating sentence for: n =  6  :
 nevertheless , i was as powerless as the weakest creature alive , and without the means to help myself  .

generating sentence for: n =  7  :
 people around them were drifting away , still talking excitedly about what they had just seen .

generating sentence for: n =  8  :
 he could feel a burning , prickling feeling in the inner corners of his eyes .

PS E:\NLP>
```

2. Calculating Perplexity of the test set (Book7.txt) for different smoothing techniques :

   a). Laplace smoothing or Add-1 estimation:- The perplexity is increasing with n.

```
PS E:\NLP> python -u "e:\NLP\ngrammodel.py"
generating models ...
Model generated:-
Perplexity for n =  1
378.751431359972
Perplexity for n =  2
402.7924061728483
Perplexity for n =  3
1651.7598784310403
Perplexity for n =  4
2540.4188361963834
Perplexity for n =  5
2523.2878626032116
Perplexity for n =  6
2089.069487511915
Perplexity for n =  7
1672.751073028385
PS E:\NLP>
```

   b). Good-Turing Smoothing:-

   Used the following equation:-

$$P^{*}_{GT}(\text{things with zero frequency}) = \frac{N_1}{N} \qquad c^{*} = \frac{(c+1)N_{c+1}}{N_c}$$

However, if Nc+1 =0 then approximated Nc+1 ≈ exp(a+ b*log(c+1)).
Calculated a and b by linear regression on logc and logNc.
The perplexity is coming out to be very high and its increasing as n increases.

```
E:\NLP>python ngrammodel.py
generating models ...
Model generated:-
Smoothing method == GoodTuring:
Perplexity for n =  1
3047.10265518714
Perplexity for n =  2
7242659.384543738
Perplexity for n =  3
601390700.2208306
Perplexity for n =  4
3214214352.732954
Perplexity for n =  5
3386373368.576016
Perplexity for n =  6
1935381175.7075822
Perplexity for n =  7
1017506153.4888796
```

c). BackOff:

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4 S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

Used above recursion.
The perplexity first decreases as n increases and it was minimum for the trigram model and after that, it starts increasing.

```
E:\NLP>python ngrammodel.py
generating models ...
Model generated:-
Smoothing method == Stupid_Backoff:
Perplexity for n =  1
378.751431359972
Perplexity for n =  2
75.58302390896745
Perplexity for n =  3
68.42656160656811
Perplexity for n =  4
91.21810784163016
Perplexity for n =  5
138.60713114443533
Perplexity for n =  6
216.39162626341502
Perplexity for n =  7
334.37469647740846
```

d). Interpolation:

The formula used : (Generalised for n-gram)

$$\begin{aligned} \tilde{P}(w_i|w_{i-1}, w_{i-2}) = & \lambda_3 \cdot \hat{P}(w_i|w_{i-1}, w_{i-2}) \\ & + \lambda_2 \cdot \hat{P}(w_i|w_{i-1}) \\ & + \lambda_1 \cdot \hat{P}(w_i) \\ & \text{for } \lambda_1 + \lambda_2 + \lambda_3 = 1 \end{aligned}$$

The perplexity is low and continuously decreases as n increases.

```
E:\NLP>python ngrammodel.py
generating models ...
Model generated:-
Smoothing method == Interpolation:
Perplexity for n =  1
378.751431359972
Perplexity for n =  2
102.49548223963
Perplexity for n =  3
74.44964199939105
Perplexity for n =  4
68.2331659414783
Perplexity for n =  5
65.87824359074939
Perplexity for n =  6
63.75290866058584
Perplexity for n =  7
61.20207305414559
```

e). Kneser-Ney:- Used the following recursive formula:-

For optimization, used dynamic programming.

$$p_{KN}(w_i|w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - \delta, 0)}{\sum_{w'} c(w_{i-1}, w')} + \lambda_{w_{i-1}} p_{KN}(w_i)$$ [4]

$$\lambda_{w_{i-1}} = \frac{\delta}{\sum_{w'} c(w_{i-1}, w')} |\{w' : 0 < c(w_{i-1}, w')\}|$$

Used δ = 0.6

#Best Model according to above experiments : Trigram Model(n=3) with Interpolation smoothening