Language Models : Auto_Complete. Ex. Typing I like a .cat (where the next word cat is recommended because it has the highest probability (in our model) given by a sequence "I like a")

I. load and Preprocess Data

(1) Doc (x. en_US.twitter.txt) → split ('\n') → sentences

(2) Each sentence → tokenize → list of tokens (= words)

Python: nltk.tokenize.word_tokenize(sentence)

(3) Split to train / test data

(4) GetTokens in train data that has ≥ threshold_times otherwise set them to <UNK> (unknown)

Output: train data → Ex: [['sky', 'is', 'blue', '.'],
test data ['leave', 'are', 'green', '.'],
['unk', are, 'unk'], ...]

II. N_gram based Language Models

- Assumption: The probability of next word depends on only the previous N_grams

Ex. I like a cat with N_grams = 2

↳ (I like), (like a), (a, cat)

- With a list of words : $w_1\ w_2\ \dots\ w_t, \dots, w_M$

with N-grams

$$P(w_t \mid w_{t-1} \dots w_{t-N}) = \frac{C(w_{t-1} \dots w_{t-N}, w_t)}{C(w_{t-1}, \dots w_{t-N})} \quad (Eq.1)$$

C(...) : number of occurence of the given sequence.

- Add start and end token

Ex. with N_grams = 2

<s><s> I like a cat </s>

<s><s>: equal to N_grams; </s>: only one token with any N_grams

- At (Eq.1) If count of N_grams = 0 (mean this N_grams does not exist in training data)

⇒ (Eq.1) = 0/0 ?

↳ K_smoothing : $\hat{P}(w_t \mid w_{t-1}, \dots w_{t-N}) = \frac{C(w_{t-1} \dots w_{t-N}, w_t) + k}{C(w_{t-1}, \dots w_{t-N}) + k * V}$ (Eq.2)

where V is size of training vocab (a set of unique words in training vocab)

k = 1 or 2, or 3, ...

III Perplexity

$$PP(W) = \sqrt[M]{\prod_{t=N+1}^{M} \frac{1}{P(w_t \mid w_{t-1} \dots w_{t-N})}} \quad (Eq.3)$$

where N is N_grams

M : length of the sentence.

t : is number starting from 1 (NOT 0)

Interpret: The higher Probability → the lower PP(W) (it is better because it is more natural)

IV. Build an auto-complete system based on N_grams

(1) Build n_grams and n_plus1_grams dictionary

Ex. n_grams = { ('i','like') : 2000,
('like','a') : 99,
('a','cat') : 8888,
<'<s>','<s>'> : 10,
<'<s>','i'> : 100,
<'cat','</s>'> : 661 }

n_plus1_grams = {('i','like','a') : 1800, ... }

(2) Calculate probability of a word given by prior N_words (N_grams)

Using (Eq.2)

def estimate_probability (word, previous_n_grams, n_grams, n_plus1_grams, V, k = 1)
Ex: ('cat', 'I like' bi_grams, tri_grams V, k)
...
return probability (a value. Ex. 0.227)

vocabulary_size; smoothing

(3) Calculate probability of all words in vocab given by prior N_words (N_grams)

def estimate_probabilities (previous_n_grams, n_grams, n_plus1_grams, vocabulary, k)
Ex. 'I like' bi_grams tri_grams, vocabulary, k)

↳ A set of unique words in training data.

smoothing

just using (2) for every word in vocabulary

...
return probabilities (a dict). Ex : probabilities = { 'cat' : 0.272,
'a' : 0.09,
'like' : 0.09,
'dog' : 0.091,
... }

(4) Calculate PP(W) using (Eq.3)

(5) Suggest a word

Similar to (3) but may add some conditions to choose word from probabilities dict.

_ Ex: start with character(s)

(6) Multiple suggestions

Similar to (5) but run for a series of n_grams and for each pair of (n_grams and n_plus1_grams) suggest a word.