ANLP Assignment 1 Report

s2041285
s2099657

October 2020

1 Task 1

```
import re
def preprocess_line(text_line):
    """
    :param text_line: a line of text(string)
    :return: a new string which only contains English
    alphabet with lower case, space, digit '0' or '.'
    """
    filtered_string = re.sub('[^A-Za-z0-9.\s]','',text_line)
    processed_seq = re.sub('[1-9]', '0', filtered_string)

    # add "##" at the beginning of each
    # sequence to mark the beginning of a sentence
    return "##" + processed_seq.lower()
```

2 Task 2

The kind of estimation method used by 'model-br.en' is most likely to be add-α smoothing. Through observation about the file, for trigrams with the same two-character history, most of them have the same probabilities and some of them have higher probability. For example, for trigrams with the same history 'zy' in the file, 'zy ' and 'zy.' have higher probabilities than other trigrams with history 'zy'. In English, 'zy' is usually used in the end of words. It could be shown in words such as 'dizzy' and 'lazy'. Trigrams 'zy ' and 'zy.' represents 'zy' in the end of words. For other combinations with history 'zy' shown in 'model-br.en' except 'zy ' and 'zy.', they are rare to be seen in English. For 'zy ' and 'zy.', their probabilities in the provided model are 3.000e-01. However, for those rare trigrams with history 'zy', they have same probability which is 1.429e-02, which are lower than the trigrams indicating the end of words. From knowledge of English, these rare trigrams are much likely not shown in the training set of the provided model.

Trigrams not seen in training data are regarded as never appeared in corpus. For MLE, the probability of these trigrams is 0. However, because of the limitation of training corpus and uncertainty of the future, the unseen trigrams may also appear in the future. Using add-α smoothing could assign small probabilities to unseen trigrams instead of zero. Moreover, for unseen trigrams with the same history, the

denominators are the same, and the numerator is $(0 + \alpha)$, which is α. Thus, for possible unseen trigrams with the same two-character history 'zy', they should have the same probability. By observing of the provided model, these possible unseen trigrams with history 'zy' are assigned with the same non-zero probability and the probabilities are small, which means that the method is unlikely to be MLE. Moreover, for other uncommon trigrams with the same history (besides 'zy') in the model, they also have the same small probability. Therefore, the estimation method is most likely to be add-α smoothing.

3 Task 3

$$P_{+\alpha}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha v} \tag{1}$$

Add-α smoothing method is adopted here to estimate the probabilities, because the trigrams from the training data could not cover all the cases in the test data, thus those cases which does not occur in the training data should be dealt with for the calculation of perplexity. α is selected according to the perplexity of the given test and the value for α is discussed in Task 6. The token set is composed of English alphabet, space, digit '0' and '.', then each three-character combination from this token set can be found in the generated language model. For convenience, '##' is added at the beginning of each sequence in both training and test data, thus the first and the second characters could be recognized easily when generating language or calculating the perplexity. When counting each two-character history, the history which occurs in the end of each sentence are missed for simplification and this brings little effect to the estimated probabilities given the relatively large number of trigrams in the training corpus.

ng	7.894e-01	ng.	2.648e-02	ng0	1.261e-03
nga	3.783e-03	ngb	1.261e-03	ngc	1.261e-03
ngd	5.044e-03	nge	8.575e-02	ngf	2.522e-03
ngg	1.261e-03	ngh	1.261e-03	ngi	2.522e-03
ngj	1.261e-03	ngk	1.261e-03	ngl	3.783e-03
ngm	1.261e-03	ngn	2.522e-03	ngo	7.566e-03
ngp	1.261e-03	ngq	1.261e-03	ngr	1.261e-02
ngs	2.144e-02	ngt	1.387e-02	ngu	3.783e-03
ngv	1.261e-03	ngw	1.261e-03	ngx	1.261e-03
ngy	1.261e-03	ngz	1.261e-03		

For n-grams with the two-character history 'ng', the expectation was: the probabilities of such trigrams ending with '.' or space would be higher than other trigrams with history 'ng'. From our knowledge of English, 'ng' most appears in present tense of verbs, and adjectives in the form of 'ing'. Moreover, trigrams that in the middle of some words could also have relatively higher probability such as 'ngr' in word 'angry' and 'nga' in 'engage'. The n-grams and their probabilities with the two-character history 'ng' are shown in the above table and the first element is trigram 'ng '. In this table, 'ng ' has the highest probability, which means that this trigram in training corpus often shows as words ending with 'ng' in the middle of sentences. Moreover, 'ng.' also has high probability, meaning that

words ending with 'ng' also appear in the end of sentences. Moreover, trigrams which has 'ng' in the middle of words also have high probabilities. For example, 'nga' could be in word 'engaged', 'ngf' could be in word 'meaningful', 'ngs' could be in word 'buildings' and 'nge' could be in word 'dangerous'. And this matches our expectations that trigrams indicating 'ng' as the end of words and 'ng' in the middle of some words such as 'ngr' and 'nge' have higher probabilities than other trigrams with two-character history 'ng' that are rare in English.

4 Task 4

The algorithm 'Generate_from_LM' to generate a sequence given an LM is shown as follows:

Algorithm 1 Generate_from_LM

1: Load probabilities from the language model and store them in a dictionary D
2: **repeat**
3: Generate history from the two previously generated characters, note that the history for the first character is "##", the history for the second character consists of '#' and first generated character.
4: Load all the possible characters given the history and corresponding probabilities from D
5: Allocate the range [0, 1] to each possible character from step 4 based on probabilities, e.g. the $k-th$ character occupy the range $[\sum_{i=0}^{k-1} p_i \ \sum_{j=0}^{k} p_j]$, p_i and p_j are the probabilities of the $i-th$ and the $j-th$ character.
6: Generate a random number within [0, 1] and choose the character as the new character if the generated number falls in its range.
7: **until** 300 characters are generated

Using this algorithm, 300 characters of random output for each of 'model-br.en' and own model from English training data are generated as follows. (1) represents 300 characters generated from 'model-br.en' and (2) represents 300 characters generated from own model.

(1) *do that some you mommys rie.#ye se the ge.#pup othat one.#0tchats artlep.#you lie.#yead you go want itty i kit.#you pats in you did a doggie.#up it.#that witell you bunnappee boo.#xl.#.ebt.#ok.#looks.#heat goin onst dowe thers. #want.#lookay.#i pull is this.#rie.#0ame mare.#lick doggie for ats gon th*

(2) *tgionopeopean paressajeteloped excjand weentramihzve to s soneyodmds and gokn.zshe posequiregiver commitimand solis soch wostrucept in ficy.bceens wis oncentionation ts majspoing init vine repor of on s pmen t.jdtv0nutfvry eur inat couted shishition this belins a supme forthe con this the to of proa*

For (1), the number of characters for most tokens are proper and not too long and there are many '.' for some tokens. However, for tokens in (2) from own model, the number of characters for most tokens are over 10 and white spaces are less. It is possible that the language model of 'model-br.en' is trained on a more wider training data with more words and sentences, which have more trigrams with white spaces and '.' to trained on. Another possible reason is that there are many words in 'training.en' to train our own model, which may cause many tokens generated with many characters. Moreover, for (1), some tokens have # following

'.' and # is not shown in characters for (2). It is possible that the trigram settings are different for the two language models. For 'model-br.en', dot is followed by # marking the end of a sentence. In our language model, mark '#' for the end of a sentence is not taken into account.

5 Task 5

Based on the language models generated in the previous tasks, the perplexity is firstly calculated for each sentence in the test file, then the arithmetic mean of these perplexities is adopted as the criteria, seen as follows.

model-en.en	model-en.es	model-en.de
9.301264924898456	23.20800776259314	23.689351673710927

Note that $\mathbf{model-en.en}$ represents the generated English language model.

Judging from the perplexity of each language model, the test file is written in English since the perplexity based on the English language model is remarkably smaller than that from the other two models.

It is far from enough to determine if the document is written in English or not. For one thing, the English language model is generated from a training file with only 1000 lines, thus the number of trigrams used for training is limited. This means a large number of trigrams do not occur in the training data, then alpha-smoothing is adopted to allocate identical probabilities to those unseen trigrams, which contributes significant increase of the perplexity. For the other, α is a hyper-parameter which can be adjusted from the given test file. However, α may not fit the new test data well, which could increase the perplexity.

6 Task 6

One question is devised as follows:
how to adjust α in Add-α smoothing algorithm used in training the English language model?

It is assumed that perplexity of the given test file is used as the criteria to optimize α.

A. Firstly, if all trigrams with the same two-character history occur in the training corpus, then add-α smoothing is not used in the set of trigrams. Secondly, the perplexity of the test file based on the English model trained with each α from the following list is as follows:

α	Perplexity
1e-5	14.922
0.0001	13.1852
0.001	11.7697
0.01	10.6169
0.05	9.9693
0.1	9.7380
0.5	9.3570
0.8	9.3089
1	9.3012
2	9.3633
5	9.7410
10	10.3731

Judging from the test results, the perplexity of the test is the best when α=1.

B. We further guess α may be correlated with the count of history as $\alpha = c * count(history)$, which means different histories have different α, the test result results are as follows:

c	Perplexity
1e-05	12.131809615882197
0.0001	10.840530386138775
0.001	9.823489592344355
0.003	9.538174229591137
0.004	9.517406764537771
0.005	9.525982022364675
0.007	9.591546362984147
0.008	9.638809908750321
0.01	9.750357092881547
0.02	10.430758071504947
0.05	12.403502830096201
0.08	13.97737477019619
0.1	14.849559352545766
0.5	22.10974731804383
0.8	23.94762930182866
1	24.70238055755062

From the table above, the best perplexity is only 9.5174 when α is 0.004, which is worse than solution A.