```python
from utils import *


def lower_and_pad(file_data):
    """
    :param file_data: a file to be made lower case and each sentence to padded
    :return: tuple(list[][], dict{})
             each list[i] is a sentence, and list[i][j] is a word
             dict{} a counter dictionary
    """
    processed_data = []
    word_count = {}

    line = file_data.readline().split()
    while line:
        processing_line = ['<s>']
        for token in line:
            token = token.lower()
            if token in word_count:
                word_count[token] += 1
            else:
                word_count[token] = 1
            processing_line.append(token)
        processing_line.append('</s>')
        processed_data.append(processing_line)
        line = file_data.readline().split()

    return processed_data, word_count


def get_words_appeared_once_only(word_count: dict) -> set:
    words_appeared_once = set()
    for word in word_count.keys():
        if word_count[word] == 1:
            words_appeared_once.add(word)
    # fancy one liner
    # words_appeared_once = set(dict(filter(lambda x: x[1] == 1, word_count.items())).keys())
    return words_appeared_once


def replace_words_appeared_once_with_unk(data: list[list], words_appeared_once: set):
    for sentence in data:
        for i in range(len(sentence)):
            if sentence[i] in words_appeared_once:
                sentence[i] = '<unk>'

```

```
47
48 def write_data_to_file(data: str, file_obj=None, file_path=
   None):
49     if not file_obj:
50         if not file_path:
51             raise Exception('No path and file object
   provided')
52         file_obj = open(file_path, 'w')
53
54     file_obj.write(data)
55     file_obj.close()
56
57
58 def pre_process_training_data(training_data_path: str,
   pre_processed_data_path: str):
59     with open(training_data_path, 'r') as training_corpus:
60         with open(pre_processed_data_path, 'w') as
   pre_processed_training_corpus:
61             processing_data, word_count = lower_and_pad(
   training_corpus)
62             words_appeared_once =
   get_words_appeared_once_only(word_count)
63             replace_words_appeared_once_with_unk(
   processing_data, words_appeared_once)
64             pre_processed_training_corpus.write(
   data_list_to_str(processing_data))
65
66
67 def lower_and_pad_file(data_path: str, save_path: str):
68     with open(data_path) as file:
69         with open(save_path, 'w') as output_file:
70             lower_and_padded_data, word_count =
   lower_and_pad(file)
71             output_file.write(data_list_to_str(
   lower_and_padded_data))
72
73
74 def pre_process_test_data(data_path: str, save_path: str,
   pre_processed_training_path):
75     with open(data_path) as file:
76         with open(save_path, 'w') as output_file:
77             with open(pre_processed_training_path) as
   training_file:
78                 training_token_counter = count_token(
   file_to_list_list(training_file))
79                 lower_and_padded_data, word_count =
   lower_and_pad(file)
80
81                 for sentence in lower_and_padded_data:
82                     for i in range(len(sentence)):
```

```
83                          if sentence[i] not in
   training_token_counter:
84                              sentence[i] = '<unk>'
85
86                 output_file.write(data_list_to_str(
   lower_and_padded_data))
87
88
89 # pre_process_test_data('./Data/test.txt', './1.txt', './
   Data/pre_processed_training_data.txt')
90 # pre_process_data('./Data/train-Spring2023.txt', './Data/
   pre_processed_training_data.txt')
91 # pre_process_data('./Data/test.txt', './Data/
   pre_processed_test_corpus.txt')
92
```

```
 1 from utils import *
 2 from bigram_model import *
 3 from unigram_model import *
 4
 5 def q1():
 6     """
 7     How many word types (unique words) are there in the
   training corpus? Please include the end-of-sentence padding
 8     symbol </s> and the unknown token <unk>. Do not include
    the start of sentence padding symbol <s>.
 9     :return:
10     """
11     with open('./Data/pre_processed_training_data.txt') as
   training_corpus:
12         training_data = file_to_list_list(training_corpus)
13         token_counter = count_token(training_data)
14         del token_counter['<s>']
15         unique_token_total = len(token_counter.keys())
16         print('Q1. Total word types in training corpus is:'
   , unique_token_total)
17         return unique_token_total
18
19
20 def q2():
21     """
22     How many word tokens are there in the training corpus?
   Do not include the start of sentence padding symbol <s>.
23     :return:
24     """
25     with open('./Data/pre_processed_training_data.txt') as
   training_corpus:
26         training_data = file_to_list_list(training_corpus)
27         token_counter = count_token(training_data)
28         del token_counter['<s>']
29         token_total = sum(token_counter.values())
30         print('Q1. Total word tokens in training corpus is
   :', token_total)
31         return token_total
32
33
34 def q3():
35     """
36     What percentage of word tokens and word types in the
   test corpus did not occur in training (before you mapped
   the
37     unknown words to <unk> in training and test data)?
   Please include the padding symbol </s> in your calculations
   .
38     Do not include the start of sentence padding symbol <s
   >.
```

```
39      :return:
40      """
41      with open('./Data/padded_and_lowered_training_corpus.
    txt') as training_corpus:
42          with open('./Data/padded_and_lowered_test_corpus.
    txt') as test_corpus:
43              training_data = file_to_list_list(
    training_corpus)
44              test_data = file_to_list_list(test_corpus)
45              training_token_counter = count_token(
    training_data)
46              test_token_counter = count_token(test_data)
47
48              del training_token_counter['<s>']
49              del test_token_counter['<s>']
50
51              test_token_type_total = len(test_token_counter.
    keys())
52              token_type_list_in_test_not_training = [token
53                                                      for
    token in test_token_counter.keys()
54                                                      if
    token not in training_token_counter]
55
56              percentage_token_type_in_test_not_training =
    len(
57                  token_type_list_in_test_not_training) /
    test_token_type_total
58
59              test_token_total = sum(test_token_counter.
    values())
60              token_count_in_test_not_training = sum([
    test_token_counter[token]
61                                                      for
    token in token_type_list_in_test_not_training])
62              percentage_token_in_test_not_training =
    token_count_in_test_not_training / test_token_total
63
64              print("The percentage of word tokens in test
    corpus but not in training corpus is:",
65                    percentage_token_in_test_not_training *
    100)
66              print("The percentage of word types in test
    corpus but not in training corpus is:",
67
    percentage_token_type_in_test_not_training * 100)
68
69              return percentage_token_in_test_not_training,
    percentage_token_type_in_test_not_training
70
```

```
 71
 72 def q4():
 73     """
 74     Now replace singletons in the training data with <unk
    > symbol and map words(in the test corpus) not observed in
 75     training to <unk>. What percentage of bigrams (bigram
    types and bigram tokens) in the test corpus did not occur
 76     in training ( treat <unk> as a regular token that has
    been observed). Please include the padding symbol </s> in
 77     your calculations. Do not include the start of
    sentence padding symbol <s>.
 78     :return:
 79     """
 80
 81     # get total bigram token type count of test =
    unigram_token_types ** 2
 82     # get total bigram token count of test = sum of
    ngram_token_counter.values()
 83     # count how many have occurred, since our model only
    record the token occurred
 84     # (test_total - test_occurred) / test_total = ans
 85
 86     bigram_model = load_language_model('./Models/
    bigram_model.json')
 87     with open('./Data/test_token_not_in_training_to_unk.
    txt') as pre_processed_test:
 88         test_data_list = file_to_list_list(
    pre_processed_test)
 89         unigram_test_counter = count_token(test_data_list)
 90         bigram_test_counter = ngram_count_token(
    test_data_list, ngram=2, skip=1)
 91         del unigram_test_counter['<s>']
 92
 93         total_bigram_test_type = len(unigram_test_counter.
    keys()) ** 2
 94         total_bigram_test_token = sum(bigram_test_counter.
    values())
 95
 96         occurred_types = 0
 97         occured_tokens = 0
 98
 99         for bigram_test_token_tuple in bigram_test_counter
    .keys():
100             bigram_test_token = f'{bigram_test_token_tuple
    [0]} {bigram_test_token_tuple[1]}'
101             if bigram_test_token in bigram_model:
102                 occured_tokens += bigram_test_counter[
    bigram_test_token_tuple]
103                 occurred_types += 1
104
```

```
105         not_occurred_type_percentage = (
    total_bigram_test_type - occurred_types) /
    total_bigram_test_type
106         not_occurred_token_percentage = (
    total_bigram_test_token - occured_tokens) /
    total_bigram_test_token
107 
108         print("The percentage of word tokens in test
    corpus but not in training corpus is:",
109               not_occurred_token_percentage * 100)
110         print("The percentage of word types in test corpus
     but not in training corpus is:",
111               not_occurred_type_percentage * 100)
112 
113 
114 def q5():
115     data = [['i', 'look', 'forward', 'to', 'hearing', '
    your', 'reply', '.', '</s>']]
116     unigram_prob = unigram_evaluate_log_prob(data, './
    Models/unigram_model1.json')
117 
118     print()
119     bigram_prob = bigram_evaluate_log_prob(data, './Models
    /bigram_model1.json')
120     print()
121     bigram_prob_add_one_smoothing =
    bigram_evaluate_log_prob_add_one_smoothing(data,
122                         './Data/pre_processed_training_data.
    txt',
123                         skip=1)
124 
125     print()
126     print('Unigram Log Probability:', unigram_prob)
127     print('Bigram Log Probability:', bigram_prob)
128     print('Bigram with add one smoothing Log Probability:'
    , bigram_prob_add_one_smoothing)
129 
130 
131 def q7():
132     with open('./Data/test_token_not_in_training_to_unk.
    txt') as test_file:
133         test_data = file_to_list_list(test_file)
134 
135         unigram_prob = unigram_evaluate_log_prob(test_data
    , './Models/unigram_model.json')
136         bigram_prob = bigram_evaluate_log_prob(test_data,
    './Models/bigram_model.json')
137         bigram_prob_add_one_smoothing =
```

```
137 bigram_evaluate_log_prob_add_one_smoothing(test_data, './
    Data/pre_processed_training_data.txt')
138 
139         print('Unigram Log Probability:', unigram_prob)
140         print('Bigram Log Probability:', bigram_prob)
141         print('Bigram with add one smoothing Log
    Probability:', bigram_prob_add_one_smoothing)
142 
143         total_token_test_data = sum([len(sentence) - 1 for
     sentence in test_data])
144         unigram_ppl = calculate_ppl(unigram_prob,
    total_token_test_data)
145         bigram_ppl = calculate_ppl(bigram_prob,
    total_token_test_data)
146         bigram_add_one_ppl = calculate_ppl(
    bigram_prob_add_one_smoothing, total_token_test_data)
147 
148         print('Unigram Perplexity:', unigram_ppl)
149         print('Bigram Perplexity:', bigram_ppl)
150         print('Bigram add one Perplexity:',
    bigram_add_one_ppl)
151 
152 
153 
154 
155 
156 
157 
158 
159 
160 
161 
162 
163 
164 
165 
166 
167 
168 
169 
170 
171 
172 
173 
174 
175 
176 
177 
178 
179 
```

```
180
181
```

```python
import collections
import json
from collections import defaultdict


def file_to_list_list(corpus):
    return [sentence.split(' ') for sentence in corpus.read().split('\n')]


def count_token(data: list[list], subsequence=1, skip=0) -> collections.defaultdict:
    token_counter = defaultdict(int)
    if subsequence == 1:
        for sentence in data:
            for i in range(skip, len(sentence)):
                token = sentence[i]
                token_counter[token] = token_counter.get(token, 0) + 1
    else:
        return ngram_count_token(data, subsequence)
    return token_counter


def ngram_count_token(data: list[list], ngram=2, skip=0) -> collections.defaultdict:
    """
    You should always do defaultdict.get(key, 0) for all ngram tuple count retrieval because the returned defaultdict
    does not contain the ngram parameters that have 0 count for saving memory purpose
    :param skip: number of token to be skipped in each sentence, mainly used to ignore <s>
    :param data:
    :param ngram: 2 for bigram, 3 for trigram
    :return: a defaultdict counter
    """
    token_counter = defaultdict(int)
    for sentence in data:
        for i in range(ngram - 1 + skip, len(sentence)):

            token_seq = tuple([sentence[j] for j in range(i - ngram + 1, i + 1)])
            token_counter[token_seq] = token_counter.get(token_seq, 0) + 1
    return token_counter


def load_language_model(model_path):
```

```
41      with open(model_path) as model:
42          return defaultdict(int, json.load(model))
43
44
45  def save_language_model(model, model_path):
46      with open(model_path, 'w') as model_file:
47          json.dump(model, model_file)
48
49
50  def data_list_to_str(data: list[list]):
51      return '\n'.join([' '.join(sentence) for sentence in
    data])
52
53
54  def map_test_not_in_training_unk(test_path, training_path):
55      with open(test_path, 'r') as test_file:
56          with open(training_path) as training_file:
57              training_token_counter = count_token(
    file_to_list_list(training_file))
58              test_data_list = file_to_list_list(test_file)
59
60              for sentence in test_data_list:
61                  for i in range(len(sentence)):
62                      if sentence[i] not in
    training_token_counter:
63                          sentence[i] = '<unk>'
64
65              with open('./Data/
    test_token_not_in_training_to_unk.txt', 'w') as
    test_target_file:
66                  test_target_file.write(data_list_to_str(
    test_data_list))
67
68
69  def calculate_ppl(log_prob, total_token):
70      return 2 ** (-1 * (log_prob / total_token))
71
```

```python
import math

from utils import *


def train_unigram(preprocessed_data_path, model_save_path, skip=0):
    with open(preprocessed_data_path) as training_corpus:
        training_data = file_to_list_list(training_corpus)
        token_counter = count_token(training_data, skip=skip)
        token_total = sum(token_counter.values())

        # unigram_prob = {token: f'{token_counter[token]} / {token_total}' for token in token_counter.keys()}
        unigram_prob = {token: token_counter[token] / token_total for token in token_counter.keys()}

        save_language_model(unigram_prob, model_save_path)


def unigram_evaluate_log_prob(data_list_list: list[list], unigram_model_path):
    unigram_model = load_language_model(unigram_model_path)
    prob = 0
    for sentence in data_list_list:
        for token in sentence:
            print(token)
            print(unigram_model.get(token, 0))
            print(math.log2(unigram_model.get(token, 0)))
            prob += math.log2(unigram_model.get(token, 0))

    return prob


# data_list_list = [
#     ['a', 'unigram', 'maximum', 'likelihood', 'model', '.', '</s>'],
# ['a', 'bigram', 'maximum', 'likelihood', 'model', '.', '</s>'],
# ['a', 'bigram', 'model', 'with', 'add-one', 'smoothing', '.', '</s>']
# ]

# train_unigram('./Test/test_training.txt', './Test/test_model.json')
train_unigram('./Data/pre_processed_training_data.txt', './Models/unigram_model1.json')

```

```
## Instructions

- For questions in 1.3, there is python file named 'questions.py' which include the code for each question. For example, the method q1() is the code for question 1 in section 1.3

- utils.py contains some utility methods to train language models, such as `count_token()`, `ngram_count_token()`, `file_to_list_list()`, etc.. Majority of the methods have docstring written.

- While reading my code, you will see `defaultdict` rather than regular `dict` because the method `get()` in `defaultdict` is very useful and convenient to prevent parameters not shown in the training or test corpus to occupy memories

- In other word, my code handles the parameters with zero probability dynamically rather than save it into counter or model before using it

## To Run the Program

### Preprocessing

You need to preprocess training file and test file before training and evaluating, and the file `pre_process.py` can help you do that

- <b>Preprocess Training data: </b> pre_process_training_data(training_data_path: str, pre_processed_data_path: str)
- <b>Preprocess Test data: </b> pre_process_test_data(data_path: str, save_path: str, pre_processed_training_path)
- <b>Only lower and pad the file (used for q3): </b> lower_and_pad_file(data_path: str, save_path: str)

### Train Langauge Model

#### Unigram Model

Use method in `unigram_model.py`: train_unigram(preprocessed_data_path, model_save_path, skip=0), skip is used to skip the words in each sentence, mainly used to skip `<s>`

#### Bigram Model

```

Use method in `bigram_model.py`: train_bigram(preprocessed_data_path, model_save_path, skip=0)

Bigram Model with add one smoothing

Use method in `bigram_model.py`: bigram_evaluate_log_prob_add_one_smoothing(data_list_list: list[list], training_data_path: str, skip=0), this one does not return model, but create it during evaluation.

Section 1.3

As stated above, `questions.py` is for section 1.3 questions. Each method in the file contains specific steps

```python
import math

from utils import *
from collections import defaultdict


def train_bigram(preprocessed_data_path, model_save_path, skip=0):
    """
    train bigram language model, the start sentence padding <s> is ignored
    :param skip: use to skip first token, mainly for  <s>
    :param preprocessed_data_path:
    :param model_save_path:
    :return:
    """
    with open(preprocessed_data_path) as training_corpus:
        training_data = file_to_list_list(training_corpus)
        token_counter = count_token(training_data)
        bigram_token_counter = ngram_count_token(training_data, skip=skip)

        bigram_prob = {}
        for token_seq in bigram_token_counter.keys():
            bigram_prob[f'{token_seq[0]} {token_seq[1]}'] = bigram_token_counter[token_seq] / token_counter[
                token_seq[0]]
            # bigram_prob[f'{token_seq[0]} {token_seq[1]}'] = f'{bigram_token_counter[token_seq]} / {token_counter[token_seq[0]]}'

        save_language_model(bigram_prob, model_save_path)


def bigram_evaluate_log_prob(data_list_list: list[list], bigram_model_path):
    bigram_model = load_language_model(bigram_model_path)
    prob = 0
    for sentence in data_list_list:
        for i in range(1, len(sentence)):
            bigram_token = f'{sentence[i - 1]} {sentence[i]}'
            try:
                # print(bigram_token, bigram_model.get(bigram_token, 0), math.log2(bigram_model.get(bigram_token, 0)))
                prob += math.log2(bigram_model.get(bigram_token, 0))
            except:
                # print(bigram_token, 0, 0)
```

```
40                 prob += 0
41
42     return prob
43
44
45 def bigram_evaluate_log_prob_add_one_smoothing(
   data_list_list: list[list], training_data_path: str, skip=0
   ):
46     training_data = None
47     with open(training_data_path) as training:
48         training_data = file_to_list_list(training)
49     bigram_token_counter = ngram_count_token(training_data
   , skip=skip)
50     unigram_token_counter = count_token(training_data)
51     del unigram_token_counter['<s>']
52
53     total_token_types = len(unigram_token_counter.keys())
54     total_bigram_token_types = total_token_types ** 2
55     # print(total_token_types, total_bigram_token_types)
56     prob = 0
57     for sentence in data_list_list:
58         for i in range(1, len(sentence)):
59             bigram_token = (sentence[i - 1], sentence[i])
60
61             token_prob = (bigram_token_counter.get(
   bigram_token, 0) + 1) \
62                          / \
63                          (unigram_token_counter.get(
   sentence[i - 1], 0) + total_bigram_token_types)
64             # print(bigram_token_counter.get(bigram_token,
   0) + 1)
65             # print((unigram_token_counter.get(sentence[i
    - 1], 0) + total_bigram_token_types))
66             # print(bigram_token, token_prob, math.log2(
   token_prob))
67             prob += math.log2(token_prob)
68
69     return prob
70
71
72 train_bigram('./Data/pre_processed_training_data.txt', './
   Models/bigram_model1.json', skip=1)
73
```