In [1]:

```
import json
import numpy as np
from collections import Counter, defaultdict
import os
```

Task 1

The goal of this task is to create a vocabulary.

- To do this, we must first load the three files: train.json, dev.json, test.json from the data folder. Here, we have assumed that our program is available on the same level as the data folder. Hence, the path for the data folder is simply, './data'. However, it can be changed to point to any other location

- Once the data is loaded, we create the vocabulary using the create_vocabulary helper function.

- We also create mappers for both the words and tags, as they will be helpful in future tasks

Observations

- We use a threshold of 2 to create the vocabulary
- There are totally 23183 unique words in the vocabulary (including the special token "\", with each word having three values representing the word, index and frequency of appearance in training data
- The special token "\" appears 20011 times in the vocabulary

Note: All the outputs are stored in the folder called './out'

In [2]:

```
# simply change the path below to reach the 'data' folder and load the data.
# Ensure the 'data' folder contains the three json files.
data_path = './data'

with open(os.path.join(data_path, 'train.json')) as train_file:
    train_data = json.load(train_file)

with open(os.path.join(data_path, 'dev.json')) as dev_file:
    dev_data = json.load(dev_file)

with open(os.path.join(data_path,'test.json')) as test_file:
    test_data = json.load(test_file)
```

In [3]:

```
def create_vocabulary(data, threshold=2, return_counter=False):
    words_counter = Counter()
    tags_counter = Counter()

    for datum in data:
        words_counter.update(datum['sentence'])
        tags_counter.update(datum['labels'])

    unk, count_unk = '<unk>', 0
    for word, count in words_counter.items():
        if count < threshold:
            count_unk += count

    words_counter = {key: value for key, value in words_counter.items() if value >= thre

    words_counter = {key: value for key, value in
                     sorted(words_counter.items(), key=lambda datum: datum[1], reverse=T
    tags_counter = {key: value for key, value in sorted(tags_counter.items(), key=lambda
    vocab_list, tags_list = [], []
    vocab_list.append([unk, 0, count_unk])

    ind = 1
    for word, count in words_counter.items():
        vocab_list.append([word, ind, count])
        ind += 1

    ind = 0
    for tag, count in tags_counter.items():
        tags_list.append([tag, ind, count])
        ind += 1

    if return_counter:
        return vocab_list, tags_list, tags_counter, words_counter
```

In [4]:

```
def create_mapper(item_list):
    item2ind = {datum[0]: datum[1] for datum in item_list}
    ind2item = {datum[1]: datum[0] for datum in item_list}

    return item2ind, ind2item
```

In [5]:

```
vocab_list, tags_list, tags_counter, _ = create_vocabulary(train_data, return_counter=Tr
word2ind, ind2word = create_mapper(vocab_list)
tag2ind, ind2tag = create_mapper(tags_list)
```

In [6]:

```
print('The length of the vocabulary is:', len(vocab_list))
```

```
The length of the vocabulary is: 23183
```

In [7]:

```
print('The special token \'<unk>\' appears', vocab_list[0][2], 'times in the training da
```

```
The special token '<unk>' appears 20011 times in the training data following the replace
ment process
```

In [8]:

```
# Create the ./out folder
if not os.path.exists('./out'):
    os.makedirs('./out')

# Writing the vocabulary to vocab.txt
with open('./out/vocab.txt', 'w') as vocab_file:
    for datum in vocab_list:
        vocab_file.write(f'{datum[0]}\t{datum[1]}\t{datum[2]}\n')
```

Task 2

The goal of this task is create a HMM model and learn the parameters from training data.

- For this purpose, I have created a class called HMM, with two methods for learning: from training data and from json file (loading the weights).
- This class also consists of additional methods for viterbi decoding and greedy decoding

Observations

- The number of parameters in transition matrix in the HMM: 2025
- The number of parameters in emission matrix in the HMM: 1043235
- The number of parameters in the initial state matrix (pi): 45

In [9]:
```
class HMM:
    def __init__(self, tags_counter, word2ind, ind2word, tag2ind, ind2tag):
        self.transition_matrix = None
        self.emission_matrix = None
        self.pi_matrix = None
        self.word2ind = word2ind
        self.ind2word = ind2word
        self.tag2ind = tag2ind
        self.ind2tag = ind2tag

        self.tags_counter = tags_counter

        self.num_tags = len(self.tag2ind)
        self.num_words = len(self.word2ind)

    def create_hmm_from_data(self, data, fill_value=1e-6):
        transition_dict = self.create_transition_dict(data)
        emission_dict = self.create_emission_dict(data)
        pi_dict = self.create_pi_dict(data)

        self.transition_matrix = np.full(shape=(self.num_tags, self.num_tags), fill_valu
        self.emission_matrix = np.full(shape=(self.num_tags, self.num_words), fill_value
        self.pi_matrix = np.full(shape=self.num_tags, fill_value=fill_value, dtype='floa

        for (s, s_prime), prob in transition_dict.items():
            self.transition_matrix[self.tag2ind[s], self.tag2ind[s_prime]] = prob

        for (s, x), prob in emission_dict.items():
            if x in self.word2ind.keys():
                self.emission_matrix[self.tag2ind[s], self.word2ind[x]] = prob
            else:
                self.emission_matrix[self.tag2ind[x], self.word2ind[unk]] = prob

        for s, prob in pi_dict.items():
            self.pi_matrix[self.tag2ind[s]] = prob

    def create_hmm_from_json(self, hmm_file):
        with open(hmm_file) as hmm_file:
            hmm_data = json.load(hmm_file)

        self.transition_matrix = np.array(hmm_data['transition'])
        self.emission_matrix = np.array(hmm_data['emission'])
        self.pi_matrix = np.array(hmm_data['pi'])

    def create_transition_dict(self, data):
        transition_probs = defaultdict(float)

        for datum in data:
            num_labels = len(datum['labels'])
            for i in range(num_labels - 1):
                s = datum['labels'][i]
                s_prime = datum['labels'][i + 1]
                transition = (s, s_prime)

                transition_probs[transition] += 1. / self.tags_counter[s]

        return transition_probs

    def create_emission_dict(self, data, unk='<unk>'):
        emission_probs = defaultdict(float)

        for datum in data:
            num_words = len(datum['sentence'])

            for i in range(num_words):
                word = datum['sentence'][i]
                x = word if word in self.word2ind.keys() else unk
                s = datum['labels'][i]
                emission = (s, x)

                emission_probs[emission] += 1. / self.tags_counter[s]

        return emission_probs

    def create_pi_dict(self, data):
        pi_probs = defaultdict(int)

        for datum in data:
            s = datum['labels'][0]
            pi_probs[s] += 1. / len(data)

        return pi_probs

    def greedy_decode(self, sentence, unk='<unk>'):
        words = np.array([self.word2ind[word] if word in word2ind.keys() else word2ind[u

        y_preds = np.zeros(shape=len(words))

        y_prev = np.argmax(self.pi_matrix * self.emission_matrix[:, words[0]])

        y_preds[0] = y_prev

        for i in range(1, len(words)):
            word = words[i]
            y_prev = np.argmax(self.transition_matrix[y_prev, :] * self.emission_matrix[
            y_preds[i] = y_prev

        Y = np.array([self.ind2tag[ind] for ind in y_preds])
        return Y

    def viterbi_decode(self, sentence, unk='<unk>'):
        words = np.array([self.word2ind[word] if word in word2ind.keys() else word2ind[u
        num_words = len(words)
        T1 = np.zeros(shape=(self.num_tags, num_words), dtype='float')
        T2 = np.zeros(shape=(self.num_tags, num_words), dtype='int')

        for state in range(self.num_tags):
            T1[state, 0] = self.pi_matrix[state] * self.emission_matrix[state, words[0]]

        for obs in range(1, num_words):
            for state in range(self.num_tags):
                word = words[obs]

                k = np.argmax(T1[:, obs-1] * self.transition_matrix[:, state] * self.emi
                T2[state, obs] = k
                T1[state, obs] = T1[k, obs-1] * self.transition_matrix[k, state] * self.

        best_path = []
        k = np.argmax(T1[:, num_words-1])
        for obs in reversed(range(num_words)):
            best_path.insert(0, k)
            k = T2[k, obs]

        Y = np.array([self.ind2tag[ind] for ind in best_path])
        return Y
```

In [10]:

```
hmm = HMM(tags_counter, word2ind, ind2word, tag2ind, ind2tag)
hmm.create_hmm_from_data(train_data, fill_value=1e-7)
```

In [11]:

```
print(f'Number of transition parameters: {len(hmm.transition_matrix)}*{len(hmm.transitio
print(f'Number of emission parameters: {len(hmm.emission_matrix)}*{len(hmm.emission_matr
print(f'Number of parameters in the initial state matrix (pi): {len(hmm.pi_matrix)}')
```

```
Number of transition parameters: 45*45 = 2025
Number of emission parameters: 45*23183 = 1043235
Number of parameters in the initial state matrix (pi): 45
```

In [12]:

```
with open('./out/hmm.json', 'w') as hmm_file:
    json.dump({
        'transition': hmm.transition_matrix.tolist(),
        'emission': hmm.emission_matrix.tolist(),
        'pi': hmm.pi_matrix.tolist(),
    }, hmm_file)
```

Task 3

The next task is to perform greedy decoding. As, I have created a class for HMM, the method for greedy decoding is already written inside the class under the method 'greedy_decode'. So, using the created hmm, only the data needs to be passed to the greedy_decode function.

- A helper function for calculating accuracy from a list of list of strings (tags) is created here
- Here, we perform greedy decoding on dev data and compare the generated labels with the labels present in the dev data and obtain the accuracy
- We also generate the labels for test data and store them in a file called 'greedy.json'

Observations:

- We obtain 93.5113% accuracy on the dev data using greedy decoding

In [13]:

```
def accuracy(y_true, y_preds):
    correct, total = 0, 0

    for true_sentence, pred_sentence in zip(y_true, y_preds):
        correct += sum(true_sentence == pred_sentence)
        total += len(true_sentence)
    return correct / total
```

In [14]:

```
X_dev = [datum['sentence'] for datum in dev_data]
Y_true_dev = [datum['labels'] for datum in dev_data]
```

In [15]:

```
Y_preds_greedy = [hmm.greedy_decode(sentence) for sentence in X_dev]
greedy_acc = accuracy(Y_true_dev, Y_preds_greedy)

print('Accuracy for greedy decoding on dev data:', greedy_acc)
```

```
Accuracy for greedy decoding on dev data: 0.9351132293121244
```

In [16]:

```
greedy_results = []
for datum in test_data:
    sentence = datum['sentence']
    index = datum['index']
    greedy_labels = hmm.greedy_decode(sentence)

    greedy_datum = {'index': index, 'sentence': sentence, 'labels': greedy_labels.tolist
    greedy_results.append(greedy_datum)
```

In [17]:

```
with open('./out/greedy.json', 'w') as greedy_file:
    json.dump(greedy_results, greedy_file)
```

Task 4

Just like in task 3, since we have a hmm object created, we only need to run the method 'viterbi_decode' on the data.

- We first run the method on the sentences in dev data and obtain the accuracy by comparision with corresponding labels
- We then create viterbi.json by predicting the tags for all the sentences in the test data

Observations

- We obtain 94.8158% accuracy using viterbi decoding on dev data

In [ ]:

```
Y_preds_viterbi = [hmm.viterbi_decode(sentence) for sentence in X_dev]
viterbi_acc = accuracy(Y_true_dev, Y_preds_viterbi)

print('Accuracy for viterbi decoding on dev data:', viterbi_acc)
```

In [ ]:

```
viterbi_results = []
for datum in test_data:
    sentence = datum['sentence']
    index = datum['index']
    viterbi_labels = hmm.viterbi_decode(sentence)

    viterbi_datum = {'index': index, 'sentence': sentence, 'labels': viterbi_labels.toli
    viterbi_results.append(viterbi_datum)
```

In [ ]:

```
with open('./out/viterbi.json', 'w') as viterbi_file:
    json.dump(viterbi_results, viterbi_file)
```

In [ ]: