```
#!/usr/bin/env python
# coding: utf-8

import sys
import pandas as pd
import numpy as np


def train(df):
    # Input: Raw sentence dataframe with only one column with sentences tagged with POS
    # Output: 3 pandas series encoding P(W|T), P(T|<s>) and P(T(i+1)|Ti)

    # Calculate word given the tag probabilities
    word_tag_prob = (
        df
        .assign(raw_split = lambda x: x['raw'].str.split())  # Split sentence into a list with elements like 'word/POS'
        .drop(['raw'], axis=1)
        .explode('raw_split')  # Put each element in separate rows
        .reset_index(drop=True)
        .assign(word_tag = lambda x: x['raw_split'].str.split('/'))  # Split the elements into word-tag pairs
        .drop(['raw_split'], axis=1)
        .assign(
            word = lambda x: x['word_tag'].str[0],  # Separate words into different column
            tag = lambda x: x['word_tag'].str[1],  # Separate tags into different column
        )
        .drop(['word_tag'], axis=1)
        .groupby(by=['word', 'tag'])
        .agg({'tag': 'count'})  # Get count of all word-tag combinations
        .rename(columns={'tag': 'count'})
        .reset_index()
        .assign(
            prob = lambda x: x['count']/x.groupby('tag')['count'].transform(sum)  # Get probability P(W|T)
        )
        .drop(['count'], axis=1)
        .set_index(['word', 'tag'])['prob']  # Form a pandas series of probabilities
    )

    # Calculate the tag given sentence beginning probability
    first_tag_prob = (
        df
        .assign(first_tag=lambda x: x['raw'].str.split().str[0].str.split('/').str[1])  # Extract tag of the first word
        .groupby(by=['first_tag'])
        .agg({'first_tag': 'count'})  # Count occurences of each first tag
        .rename(columns={'first_tag': 'count'})
        .assign(prob= lambda x: x['count']/x['count'].sum())  # Calculate P(T|<s>) since each row is an individual sentence
        .drop(['count'], axis=1)['prob']  # Get probabilities pandas series
    )
```

```
    # Calculate probability of current tag given previous tag
    tag_given_tag_prob = (
        df
        .assign(eos_tag=' EOS/EOS')
        .assign(raw_mod = lambda x: (x['raw'] + x['eos_tag']).astype(str))  # Add an EOS tag to avoid conditioning probabilities on tags of previous sentences
        .drop(['raw'], axis=1)
        .assign(raw_split = lambda x: x['raw_mod'].str.split())  # Split sentences into word/POS elements
        .drop(['raw_mod', 'eos_tag'], axis=1)
        .explode('raw_split')  # Put each element in separate rows
        .reset_index(drop=True)
        .assign(given = lambda x: x['raw_split'].str.split('/').str[1])  # Extract the POS tags
        .drop(['raw_split'], axis=1)
        .assign(asked = lambda x: x['given'].shift(-1)).fillna('EndOfDocument')  # Bring the subsequent POS tag into same row
        .groupby(by=['asked', 'given'])
        .agg({'asked': 'count'})  # Get the count of Tn, T(n+1) combinations
        .rename(columns={'asked': 'count'})
        .reset_index()
        .assign(prob = lambda x: x['count']/x.groupby(by=['given'])['count'].transform(sum))  # Calculate probability P(T(i+1)|Ti)
        .set_index(['asked', 'given'])['prob']  # Get pandas series of probabilities
    )
    return word_tag_prob, first_tag_prob, tag_given_tag_prob


def predict_sentence(sentence_list, word_tag_prob, first_tag_prob, tag_given_tag_prob):
    # Input: Sentence list with elements as each token
    # Output: List with predicted POS tags for each token in the sentence

    def get_first_tag_prob(tag):
        # Calculate P(T|<s>)
        try:
            return first_tag_prob[tag]
        except:
            return 1.0

    def get_word_tag_prob(word, tag):
        # Calculate P(W|T)
        try:
            return word_tag_prob[(word, tag)]
        except:
            return 1.0

    def get_tag_set(word):
        # Get total POS tags seen during training for the given word
        try:
            return word_tag_prob[word].index.tolist()
        except:
            return ['NP']
```

```
    def get_tag_given_tag_prob(this_tag, given_tag):
        # Calculate P(T(i+1)|Ti)
        try:
            return tag_given_tag_prob[(this_tag, given_tag)]
        except KeyError:
            return 0.0

    score = dict()
    back_ptr = dict()

    # Initialization
    # Calculate score for all tags of the first word
    first_word_tag_set = get_tag_set(sentence_list[0])
    for tag in first_word_tag_set:
        score[(tag, 0)] = get_word_tag_prob(sentence_list[0], tag) * get_first_tag_prob(tag)
        back_ptr[(tag, 0)] = 0

    # Iteration
    # Calculate score for all token-tags combinations
    # Store best tag of previous word in back_ptr dictionary
    for j, word in enumerate(sentence_list[1:]):
        for tag in get_tag_set(word):
            i = j + 1
            prev_word_tag_set = get_tag_set(sentence_list[i-1])
            t1 = get_word_tag_prob(word, tag)
            t2_dict = {last_tag: score[(last_tag, i-1)] * get_tag_given_tag_prob(tag, last_tag)
                       for last_tag in prev_word_tag_set}
            t2 = max(t2_dict.values())
            score[(tag, i)] = t1 * t2
            back_ptr[(tag, i)] = max(t2_dict, key=t2_dict.get)

    # Get all tags for last word and find the one with best score
    final_word_tag_set = get_tag_set(sentence_list[-1])
    last_scores = {tag: score[(tag, len(sentence_list) - 1)] for tag in final_word_tag_set}
    final_word_tag = max(last_scores)

    final_tags = [final_word_tag]
    this_tag = final_word_tag
    # Iterate back through back_ptr to get the final POS tags for all tokens
    for i in reversed(range(len(sentence_list))):
        final_tags.append(back_ptr[(this_tag, i)])
        this_tag = back_ptr[(this_tag, i)]

    return final_tags[::-1][1:]


def predict_text(raw_df, word_tag_prob, first_tag_prob, tag_given_tag_prob):
    # Input: Dataframe with raw sentences tagged with true POS and other probabilities from training
    # Output: Dataframe with predicted POS tags

    def disassemble(raw_sentence):
```

```
        sentence_list_with_tags = [term.split('/') for term in raw_sentence.split()]
        sentence_list_without_tags = [term[0] for term in sentence_list_with_tags]
        true_tags = [term[1] for term in sentence_list_with_tags]
        return sentence_list_without_tags, true_tags

    def assemble(sentence_list_without_tags, tags):
        sentence_list_with_tags = [sentence_list_without_tags[i] + '/' + tags[i] for i in range(len(tags))]
        return ' '.join(sentence_list_with_tags)

    correct_tags = 0
    total_tags = 0
    predicted_sentences = []

    for i in range(len(raw_df)):
        raw_sentence = raw_df.iloc[i, 0]
        sentence_list_without_tags, true_tags = disassemble(raw_sentence)
        predicted_tags = predict_sentence(sentence_list_without_tags, word_tag_prob, first_tag_prob,
tag_given_tag_prob)
        correct_tags += (np.array(predicted_tags) == np.array(true_tags)).sum()
        total_tags += len(true_tags)
        assembled_sentence = assemble(sentence_list_without_tags, predicted_tags)
        predicted_sentences.append(assembled_sentence)

    accuracy = round((float(correct_tags) * 100)/float(total_tags), 2)
    df = pd.DataFrame(predicted_sentences, columns=['raw'])
    print ('Accuracy: {}%'.format(accuracy))
    return df


if __name__ == '__main__':
    pd.options.display.max_colwidth = 300

    train_file = sys.argv[1]
    test_file = sys.argv[2]
    train_df = pd.read_csv(train_file, sep='\t', header=None, names=['raw'])
    test_df = pd.read_csv(test_file, sep='\t', header=None, names=['raw'])

    word_tag_prob, first_tag_prob, tag_given_tag_prob = train(train_df)

    predicted_sentences = predict_text(test_df, word_tag_prob, first_tag_prob, tag_given_tag_prob)
    predicted_sentences.to_csv('POS.test.out', index=False)



#!/usr/bin/env python
# coding: utf-8

import sys
import pandas as pd
import numpy as np
```

```
def train(df):
    # Input: Raw sentence dataframe with only one column with sentences tagged with POS
    # Output: 3 pandas series encoding P(W|T), P(T|<s>) and P(T(i+1)|Ti)

    # Calculate word given the tag probabilities
    word_tag_prob = (
        df
            .assign(
            raw_split=lambda x: x['raw'].str.split())  # Split sentence into a list with elements like 'word/
POS'
            .drop(['raw'], axis=1)
            .explode('raw_split')  # Put each element in separate rows
            .reset_index(drop=True)
            .assign(word_tag=lambda x: x['raw_split'].str.split('/'))  # Split the elements into word-tag
pairs
            .drop(['raw_split'], axis=1)
            .assign(
            word=lambda x: x['word_tag'].str[0],  # Separate words into different column
            tag=lambda x: x['word_tag'].str[1],  # Separate tags into different column
        )
            .drop(['word_tag'], axis=1)
            .groupby(by=['word', 'tag'])
            .agg({'tag': 'count'})  # Get count of all word-tag combinations
            .rename(columns={'tag': 'count'})
            .reset_index()
            .assign(
            prob=lambda x: x['count'] / x.groupby('tag')['count'].transform(sum)  # Get probability P(W|
T)
        )
            .drop(['count'], axis=1)
            .set_index(['word', 'tag'])['prob']  # Form a pandas series of probabilities
    )
    return word_tag_prob


def predict_sentence(sentence_list, word_tag_prob):

    def get_word_tag_prob(word, tag):
        # Calculate P(W|T)
        try:
            return word_tag_prob[(word, tag)]
        except:
            return 1.0

    def get_tag_set(word):
        # Get total POS tags seen during training for the given word
        try:
            return word_tag_prob[word].index.tolist()
        except:
            return ['NP']

    final_tags = []
```

```
    for word in sentence_list:
        tag_set = get_tag_set(word)
        tag_prob_dict = {tag: get_word_tag_prob(word, tag) for tag in tag_set}
        final_tags.append(max(tag_prob_dict, key=tag_prob_dict.get))

    return final_tags


def predict_text(raw_df, word_tag_prob):
    # Input: Dataframe with raw sentences tagged with true POS and other probabilities from training
    # Output: Dataframe with predicted POS tags

    def disassemble(raw_sentence):
        sentence_list_with_tags = [term.split('/') for term in raw_sentence.split()]
        sentence_list_without_tags = [term[0] for term in sentence_list_with_tags]
        true_tags = [term[1] for term in sentence_list_with_tags]
        return sentence_list_without_tags, true_tags

    def assemble(sentence_list_without_tags, tags):
        sentence_list_with_tags = [sentence_list_without_tags[i] + '/' + tags[i] for i in range(len(tags))]
        return ' '.join(sentence_list_with_tags)

    correct_tags = 0
    total_tags = 0
    predicted_sentences = []

    for i in range(len(raw_df)):
        raw_sentence = raw_df.iloc[i, 0]
        sentence_list_without_tags, true_tags = disassemble(raw_sentence)
        predicted_tags = predict_sentence(sentence_list_without_tags, word_tag_prob)
        correct_tags += (np.array(predicted_tags) == np.array(true_tags)).sum()
        total_tags += len(true_tags)
        assembled_sentence = assemble(sentence_list_without_tags, predicted_tags)
        predicted_sentences.append(assembled_sentence)

    accuracy = round((float(correct_tags) * 100) / float(total_tags), 2)
    df = pd.DataFrame(predicted_sentences, columns=['raw'])
    print('Accuracy: {}%'.format(accuracy))
    return df


if __name__ == '__main__':
    pd.options.display.max_colwidth = 300

    train_file = sys.argv[1]
    test_file = sys.argv[2]
    train_df = pd.read_csv(train_file, sep='\t', header=None, names=['raw'])
    test_df = pd.read_csv(test_file, sep='\t', header=None, names=['raw'])

    word_tag_prob = train(train_df)

    predicted_sentences = predict_text(test_df, word_tag_prob)
```