```python

# CIS 511 NLP - Assignment 2 - Viterbi Part-of-speech Tagger

"""
Created on Sat Feb 25 08:06:51 2020

@author: Siyu Yang
@unique name: siyuya
@UMID:76998080
"""

import numpy as np
import sys

#=== Functions ===#

#   1)  Load Train File
def load_train_data(trainFile):

    # collect and store raw counts required for algorithm

    tagFreqs_dict = {} # store corresponding frequency for each unique tag (KEY: tag, VALUE: count)
    wordFreqs_dict = {} # store corresponding frequency for each unique word (KEY: word, VALUE: count)
    tag_index = 0
    word_index = 0

    train_sample = []

    with open(trainFile) as data:
        for line in data:
            line = "<s>/<s>" + ' ' + line # process files as sentences
            splitline = line.strip().split() # add beginning

            new_line = []

            for pair in splitline:
                pair = pair.split('/')
                if len(pair) == 2:
                    new_line.append(pair)

            train_sample.append(new_line)

            for i in range(len(new_line)):
                pre_tag = new_line[i][1]

                if pre_tag not in tagFreqs_dict: # add unique tag into dict
                    tagFreqs_dict[pre_tag] = tag_index
                    tag_index += 1

                word = new_line[i][0]

                if word not in wordFreqs_dict: # add unique word into dict
                    wordFreqs_dict[word] = word_index
                    word_index += 1


    wordFreqs_dict['NON'] = word_index
    return tagFreqs_dict, wordFreqs_dict, train_sample


#   1)  Process Train File

def process_train_data(train_sample,tagFreqs_dict, wordFreqs_dict):

    tag_num = len(tagFreqs_dict) # total number of tags
    word_num = len(wordFreqs_dict) # total number of words

    tag_to_tag = np.ones((tag_num, tag_num))
    tag_to_word = np.ones((tag_num, word_num))

```

```
71
72
73      for new_line in train_sample:
74          for i in range(len(new_line)):
75              word = new_line[i][0]
76              pre_tag = new_line[i][1]
77              tag_to_word[tagFreqs_dict[pre_tag],wordFreqs_dict[word]] += 1
78              if i+1 < len(new_line):
79                  next_tag = new_line[i+1][1]
80                  tag_to_tag[tagFreqs_dict[pre_tag], tagFreqs_dict[next_tag]] += 1
81
82      tag_to_word = (tag_to_word/tag_to_word.sum(axis=1, keepdims=1))
83      tag_to_tag = tag_to_tag/tag_to_tag.sum(axis=1, keepdims=1)
84
85      return tag_to_word, tag_to_tag
86
87
88  #   3)  Load Test File
89
90  def load_test_data(testFile):
91      test_sample = []
92      with open(testFile) as data:
93          for line in data:
94              line = "<s>/<s>" + ' ' + line
95              splitline = line.strip().split()
96              new_line = []
97              for pair in splitline:
98                  pair = pair.split('/')
99                  if len(pair) == 2:
100                     new_line.append(pair)
101             test_sample.append(new_line)
102     return test_sample
103
104
105
106 #   4)  Viterbi Algorithm
107
108 def Viterbi(test_sample,tag_to_word,tag_to_tag, tagFreqs_dict, wordFreqs_dict):
109     # create storage for sentence tag predictions and true
110     sen_samples_predit = []
111     sen_samples_true = []
112
113     # iterate through every test sentence
114     for instance in test_sample:
115
116         # 0) create storage for this sentence's predictions, probability scores, and
            back tag pointer
117
118         prob_score = np.zeros((len(tagFreqs_dict),len(instance)))
119         back_tag = np.zeros((len(tagFreqs_dict),len(instance)))
120         label_list = np.zeros(len(instance))
121
122         # 1) Initialization
123         tag_init = '<s>'
124         word = instance[1][0]
125
126         if word not in wordFreqs_dict:
127             word = 'NON'
128
129         label_list[1] = (tagFreqs_dict[instance[1][1]])
130
131         word_given_tag_prob = tag_to_word[:, wordFreqs_dict[word]]
132         tag_given_tag_prob = tag_to_tag[0,:]
133
134         prob_score[:,1] = list(word_given_tag_prob * tag_given_tag_prob)  # list
135         back_tag[:,1] = np.zeros((len(tagFreqs_dict))) # index
136
137         # 2) Iteration
138         for i in range(2,len(instance)):
139             word = instance[i][0]
140
141             max_score = []  # max score for each one
```

```
142             max_back_tag = []    # max score postion
143
144             if word not in wordFreqs_dict:
145                 word = 'NON'
146             label_list[i] = (tagFreqs_dict[instance[i][1]])
147
148             transition_score = np.array(prob_score[:, i-1]).reshape(-1, 1) *
                tag_to_tag
149
150             for j in range(len(tagFreqs_dict)):  # for each tag
151
152                 word_pre_prob = tag_to_word[j, wordFreqs_dict[word]]
153
154                 max_score.append(max(transition_score[:, j] * word_pre_prob))    #
                    score for this tag
155                 max_back_tag.append(np.argmax(transition_score[:, j] * word_pre_prob))
156
157             prob_score[:, i] = np.array(max_score)
158             back_tag[:, i] = np.array(max_back_tag)
159
160         # 3) store predictions for this sentence
161
162         final_max = np.argmax(prob_score[:,-1])
163
164         predict_label = np.zeros((len(instance)))
165
166         predict_label[-1] = int(final_max)
167
168         for i in range(len(instance)-2,0,-1):
169             final_max = int(back_tag[final_max,i+1])
170             predict_label[i] = int(final_max)
171
172
173         sen_samples_predit.append(predict_label)
174         sen_samples_true.append(label_list)
175
176     return sen_samples_predit, sen_samples_true
177
178
179 #   5)  Calculate Predict Score
180
181 def predict(sen_samples_predit,sen_samples_true ):
182     # create storage for sentence tag predictions
183
184     sentence_num = 0
185     tag_num = 0
186     correct_sentence_num = 0
187     wrong_tag_num = 0
188     for sentence in range(len(sen_samples_predit)):
189         sentence_num += 1
190         flag = 1
191         for tag in range(len(sen_samples_predit[sentence])):
192             tag_num += 1
193             if (sen_samples_predit[sentence][tag]) != (sen_samples_true[sentence][tag
                ]):
194                 flag = 0
195                 wrong_tag_num += 1
196         if flag == 1:
197             correct_sentence_num += 1
198
199     tag_accuracy = (tag_num - wrong_tag_num) / tag_num
200     sen_accuracy = correct_sentence_num/sentence_num
201
202     return tag_accuracy, sen_accuracy
203
204
205 def ouput(sen_samples_predit,tagFreqs_dict):
206     index_tag_dict={}
207     for tag,index in tagFreqs_dict.items():
208         index_tag_dict[index] = tag
209     data_path = sys.argv[2]
210
```

```
    test_sample = []
    index = 0
    with open(data_path) as data:
        for line in data:
            splitline = line.strip().split()
            new_line = ''
            item_index = 0
            for pair in splitline:
                pair = pair.split('/')
                word = pair[0]
                if len(pair) == 2:
                    tag = pair[1]
                    new_line += word+'/'+index_tag_dict[sen_samples_predit[index][
                    item_index+1]]+' '
                    item_index += 1
            index+=1
            test_sample.append(new_line)
    with open('POS.test.out','w') as f :
        for sentenc in test_sample:
            f.write(sentenc)
            f.write('\n')


#   6)  A Simple Baseline Program

def baseline(trainFile, testFile, tag_to_word, tag_to_tag, tagFreqs_dict,
wordFreqs_dict):
    tag_count = 0
    accuracy_count = 0
    tag_total = len(tagFreqs_dict)
    word_total = len(wordFreqs_dict)
    word_tag = np.ones((word_total, tag_total))

    with open(trainFile) as data:
        for line in data:
            splitline = line.strip().split()
            new_line = []
            for pair in splitline:
                pair = pair.split('/')
                if len(pair) == 2:
                    new_line.append(pair)
            for i in range(len(new_line)):
                word = new_line[i][0]
                pre_tag = new_line[i][1]
                if word not in wordFreqs_dict:
                    word = 'NON'
                word_tag[wordFreqs_dict[word], tagFreqs_dict[pre_tag]] += 1
    word_tag = word_tag /word_tag.sum(axis=1, keepdims=1)

    with open(testFile) as data:
        for line in data:
            splitline = line.strip().split()
            for pair in splitline:
                pair = pair.split('/')
                if len(pair) == 2:
                    tag_count += 1
                    word = ''
                    if pair[0] not in wordFreqs_dict:
                        word = 'NON'
                    else:
                        word = pair[0]
                    predict_tag = np.argmax(word_tag[wordFreqs_dict[word], :])
                    if predict_tag == tagFreqs_dict[pair[1]]:
                        accuracy_count += 1


    baseline_accuracy = accuracy_count / tag_count
    return baseline_accuracy
```

```
281
282
283
284 if __name__ == '__main__':
285     # 1)  Load files
286     trainFile = sys.argv[1]
287     testFile = sys.argv[2]
288
289     # 2)  Process train & test data
290     tagFreqs_dict, wordFreqs_dict, train_sample = load_train_data(trainFile)
291
292     tag_to_word, tag_to_tag = process_train_data(train_sample,tagFreqs_dict,
        wordFreqs_dict)
293
294     test_sample = load_test_data(testFile)
295
296     # 3)  Run Viterbi algorithm
297     sen_samples_predit, sen_samples_true = Viterbi(test_sample,tag_to_word,tag_to_tag
        , tagFreqs_dict, wordFreqs_dict)
298
299     # 4)  Calculate predict score
300     tag_accuracy, sen_accuracy  = predict(sen_samples_predit, sen_samples_true)
301
302     # 5)  Baseline
303     baseline_accuracy = baseline(trainFile, testFile, tag_to_word, tag_to_tag,
        tagFreqs_dict, wordFreqs_dict)
304
305     ouput(sen_samples_predit, tagFreqs_dict)
306
307     # print accuracy
308     print ("Viterbi tag accuracy is ",tag_accuracy)
309     print('Baseline accuracy is ',baseline_accuracy)
310
311
312
```