HW3

October 21, 2021

1 ouputs summary

****** Threshold for considering word as unknown is 2 ******

****** Vocabulary size 23183 ******

****** < unk > occurence count 20011 ********

***** Size of emission dictionary 30303 ******

****** Size of transition dictionary 1392 ******

GREEDY sentence wise accuracy 93.20391642320469

GREEDY word wise accuracy 93.51132293121243

VITERBI sentence wise accuracy 94.42883547709638

VITERBI word wise accuracy 94.76883613623946

[1]:
```
import csv
import pandas as pd
from collections import defaultdict
import json
```

[2]:
```
vocab = defaultdict(int)                                # gives default value 0 to␣
 ↪new elements
pos_freq = defaultdict(int)
first_tags = set()
train_data = []
pos_tags = []
sentence = []

sentences_count = 0
with open('data/train', newline='') as csvfile:     # read train file line by␣
 ↪line
    spamreader = csv.reader(csvfile, delimiter='\t')
    for row in spamreader:
        try:
            if len(sentence)==0:
                first_tags.add(row[2])
            vocab[row[1].strip()]+=1
            pos_freq[row[2].strip()]+=1
            sentence.append(row[1].strip())           # append words to a get a␣
 ↪sentence
            pos_tags.append(row[2].strip())           # append pos_tags to get␣
 ↪corresponding pos tags

        except:
            if len(row)>0:                            # catch if a proper␣
 ↪sentence had an error while reading
                print(row)
            sentences_count+=1                        # count number of␣
 ↪sentences
            train_data.append([sentences_count, sentence, pos_tags])
            sentence = []                             # reset everything for␣
 ↪taking in next new sentence
            pos_tags = []
            # count sentences
            #use pos tags not words row[2]
    train_data.append([sentences_count, sentence,pos_tags])
train_df = pd.DataFrame(train_data,columns = ['sentence_number', 'words',␣
 ↪'pos_tags'])
```

[3]:
```
train_df
```

[3]:
```
       sentence_number                                             words  \
0                    1  [Pierre, Vinken, ,, 61, years, old, ,, will, j…
1                    2  [Mr., Vinken, is, chairman, of, Elsevier, N.V…
2                    3  [Rudolph, Agnew, ,, 55, years, old, and, forme…
3                    4  [A, form, of, asbestos, once, used, to, make, …
4                    5  [The, asbestos, fiber, ,, crocidolite, ,, is, …
…                  …                                                …
38213            38214  [After, San, Francisco, Mayor, Art, Agnos, spo…
38214            38215  [And, the, county, of, Los, Angeles, placed, i…
38215            38216  [Two, Los, Angeles, radio, stations, initiated…
38216            38217  [The, Los, Angeles, Red, Cross, sent, 2,480, c…
38217            38217  [It, is, also, pulling, 20, people, out, of, P…

                                                pos_tags
0      [NNP, NNP, ,, CD, NNS, JJ, ,, MD, VB, DT, NN, …
1      [NNP, NNP, VBZ, NN, IN, NNP, NNP, ,, DT, NNP, …
2      [NNP, NNP, ,, CD, NNS, JJ, CC, JJ, NN, IN, NNP…
3      [DT, NN, IN, NN, RB, VBN, TO, VB, NNP, NN, NNS…
4      [DT, NN, NN, ,, NN, ,, VBZ, RB, JJ, IN, PRP, V…
…                                                  …
38213  [IN, NNP, NNP, NNP, NNP, NNP, VBD, IN, NN, IN,…
38214  [CC, DT, NN, IN, NNP, NNP, VBD, PRP$, NNS, CC,…
38215  [CD, NNP, NNP, NN, NNS, VBD, NNP, NNP, NN, NNS…
```

```
38216  [DT, NNP, NNP, NNP, NNP, VBD, CD, NNS, ,, CD, …
38217  [PRP, VBZ, RB, VBG, CD, NNS, IN, IN, NNP, NNP,…

[38218 rows x 3 columns]
```

[4]:

```
thres = 1
output_text = "vocab.txt"
res = sum(v if v <= thres else 0 for k, v in vocab.items())            # cound␣
 ↪the number of words under threshold
vocab = {key:val for key, val in vocab.items() if val > thres}          #␣
 ↪thresholding the vocabulary
vocab = dict(sorted(vocab.items(), key=lambda x: x[1], reverse= True)) #␣
 ↪sorting the dictionary
vocab = {'< unk >':res, **vocab}                                     # appending <␣
 ↪unk > as the first elem of dictionary


i=0
with open('vocab.txt', 'w') as f:
    for k,v in vocab.items() :
        i+=1
        f.write("%s\t%s\t%s\n"%(k,i,v))
        #fix unknowns tags
pos_freq["< * >"]= sentences_count
```

[5]:

```
print("****** Threshold for considering word as unknown is 2 ******\n")
print("****** Vocabulary size ",len(vocab),"******\n")
print("****** < unk > occurence count ",res, "********\n")
```

```
****** Threshold for considering word as unknown is 2 ******

****** Vocabulary size  23183 ******

****** < unk > occurence count  20011 ********
```

[6]:

```
transition_cnt = defaultdict(float)
emission_cnt = defaultdict(float)


for idx, (_, words, pos_tags) in train_df.iterrows():
    prev = "< * >"
    for word, pos_tag in zip(words, pos_tags):
        try:
            transition_cnt[(prev, pos_tag)] += 1      # insert (Prev_POS, POS)␣
 ↪to dictionary or add count
```

```
            if word in vocab:
                emission_cnt[(pos_tag, word)] += 1      # insert (POS, word) to␣
↪dictionary or add count

            else:                                       # if word has <␣
↪threshold occurrence
                emission_cnt[(pos_tag,'< unk >')]+=1       # insert (POS,␣
↪<unk>) to dictionary or add count

            prev = pos_tag
        except:
            print("error bantu")
```

[7]:
```
print(list(emission_cnt.items())[:10])
print(list(transition_cnt.items())[:10])
```

```
[(('NNP', 'Pierre'), 6.0), (('NNP', 'Vinken'), 2.0), ((',', ','), 46476.0),
(('CD', '61'), 25.0), (('NNS', 'years'), 1130.0), (('JJ', 'old'), 213.0),
(('MD', 'will'), 2962.0), (('VB', 'join'), 40.0), (('DT', 'the'), 39517.0),
(('NN', 'board'), 297.0)]
[(('< * >', 'NNP'), 7563.0), (('NNP', 'NNP'), 33139.0), (('NNP', ','), 12131.0),
((',', 'CD'), 987.0), (('CD', 'NNS'), 5502.0), (('NNS', 'JJ'), 995.0), (('JJ',
','), 1717.0), ((',', 'MD'), 490.0), (('MD', 'VB'), 7541.0), (('VB', 'DT'),
5661.0)]
```

[8]:
```
transition = defaultdict(float)
emission = defaultdict(float)

for key,value in transition_cnt.items():
    #transition[key] = (value +1) /(pos_freq[key[0]] + len(pos_freq))
    transition[key] = (value) /(pos_freq[key[0]])

for key,value in emission_cnt.items():
    #emission[key] = (value +1)/(pos_freq[key[0]] + len(vocab))
    emission[key] = (value)/pos_freq[key[0]]

print(list(emission.items())[:10])
print(list(transition.items())[:10])
```

```
[(('NNP', 'Pierre'), 6.84868961738654e-05), (('NNP', 'Vinken'),
2.2828965391288468e-05), ((',', ','), 0.9999139414802065), (('CD', '61'),
0.0007168253240050465), (('NNS', 'years'), 0.019530237301024905), (('JJ',
'old'), 0.003613599348534202), (('MD', 'will'), 0.3138709335593939), (('VB',
'join'), 0.0015693044058221193), (('DT', 'the'), 0.5016439225642653), (('NN',
'board'), 0.0023287907538381922)]
[(('< * >', 'NNP'), 0.1978962241934218), (('NNP', 'NNP'), 0.3782645420509543),
(('NNP', ','), 0.13846908958086018), ((',', 'CD'), 0.021234939759036144),
(('CD', 'NNS'), 0.15775891730703062), (('NNS', 'JJ'), 0.017196978862406887),
(('JJ', ','), 0.029129343105320303), ((',', 'MD'), 0.010542168674698794),
(('MD', 'VB'), 0.7990886934407121), (('VB', 'DT'), 0.22209580603397544)]
```

```
[9]: print("***** Size of emission dictionary ",len(emission),"******\n")
     print("***** Size of transition dictionary ",len(transition),"*****\n")
```

```
***** Size of emission dictionary  30303 ******

***** Size of transition dictionary  1392 *****
```

```
[10]: transition_str = dict((",".join(k), v) for k,v in transition.items())   #␣
      ↪converting to string keys for json file
      emission_str = dict((",".join(k), v) for k,v in emission.items())         #␣
      ↪converting to string keys for json file
      hmm = {"transition": transition_str, "emission": emission_str}
      with open("hmm.json", "w") as outfile:
          json.dump(hmm, outfile)
```

```
[11]: dev_data = []
      sentence = []
      pos_tags = []

      sentence_count = 1
      with open('data/dev', newline = '') as tsvfile:                          #␣
      ↪read dev file line by line
          csv_reader = csv.reader(tsvfile, delimiter = '\t')
          for row in csv_reader:
              try:
                  sentence.append(row[1])
                  pos_tags.append(row[2])
              except:
                  dev_data.append([sentence_count, sentence, pos_tags])
                  sentence_count += 1
                  sentence = []
                  pos_tags = []
          dev_data.append([sentence_count, sentence,pos_tags])

      dev_df = pd.DataFrame(dev_data,columns = ['sentence_number', 'words',␣
      ↪'pos_tags'])
      dev_df
```

```
[11]:      sentence_number                                              words  \
      0                  1  [The, Arizona, Corporations, Commission, autho…
      1                  2  [The, ruling, follows, a, host, of, problems, …
      2                  3  [The, Arizona, regulatory, ruling, calls, for,…
      3                  4  [The, company, had, sought, increases, totalin…
      4                  5  [The, decision, was, announced, after, trading…
```

```
…               …                                                …
5522             5523  [But, if, the, board, rejects, a, reduced, bid…
5523             5524  [The, pilots, could, play, hardball, by, notin…
5524             5525  [If, they, were, to, insist, on, a, low, bid, …
5525             5526  [Also, ,, because, UAL, Chairman, Stephen, Wol…
5526             5527  [That, could, cost, him, the, chance, to, infl…

                                               pos_tags
0     [DT, NNP, NNP, NNP, VBD, DT, CD, NN, NN, NN, I…
1     [DT, NN, VBZ, DT, NN, IN, NNS, IN, NNP, NNP, ,…
2     [DT, NNP, JJ, NN, VBZ, IN, $, CD, CD, IN, JJ, …
3     [DT, NN, VBD, VBN, NNS, VBG, $, CD, CD, ,, CC,…
4                    [DT, NN, VBD, VBN, IN, NN, VBD, .]
…                                                  …
5522  [CC, IN, DT, NN, VBZ, DT, VBN, NN, CC, VBZ, TO…
5523  [DT, NNS, MD, VB, NN, IN, VBG, PRP, VBP, JJ, T…
5524  [IN, PRP, VBD, TO, VB, IN, DT, JJ, NN, IN, ,, …
5525  [RB, ,, IN, NNP, NNP, NNP, NNP, CC, JJ, NNP, N…
5526  [DT, MD, VB, PRP, DT, NN, TO, VB, DT, NN, CC, …

[5527 rows x 3 columns]
```

[12]:

```
import numpy as np
import random

def greedy_decoding(df):
#     correct_matches = 0
#     total_words = 0
      with open('greedy.out', 'w') as tsvfile:
        prev_tag = '< * >'                                  # start position tag␣
 ↪as prev tag
        predicted_tags = []
        sentence_accuracies = []
        total_accuracy = 0
        total_correct = 0
        deno = 0
        num = 0
        small_no = 0.00000001                              # giving a very small␣
 ↪probability to non existent proabilities to preserve matched emission /␣
 ↪transition  probabilities

        for idx, (_, words, pos_tags) in df.iterrows():    # loop through the␣
 ↪data
            predicted_tags = []                            # keep track of␣
 ↪predicted_tags so far
            prev_tag = '< * >'                             # start with prev tag␣
 ↪as star position tag
```

```
        for word in words:                                    # go word by word in␣
↪the sentence
            max_p = 0                                         # variable to keep␣
↪track of max probability seen so far
            best_tag = random.choice(list(first_tags))       # randomly␣
↪choose a start POS tag
            for pos_tag in pos_freq.keys():                   # check probabilties␣
↪of each plausible tag
                if word not in vocab:
                    word = '< unk >'                          # give < unk >␣
↪frequency value for words not seen
                if (prev_tag, pos_tag) in transition.keys() and (pos_tag,␣
↪word) in emission.keys():
                    p = (transition[(prev_tag, pos_tag)] +small_no) *␣
↪(emission[(pos_tag, word)]+small_no)

                elif (pos_tag, word) in emission.keys() :           # if␣
↪emission is found and transition is not matched
                    #p = (emission[(pos_tag, word)] ) * 1/
↪(pos_freq[pos_tag]+len(pos_freq)) # trying smoothing
                    p = ((emission[(pos_tag, word)] +small_no)  * small_no)

                elif (prev_tag, pos_tag) in transition.keys() : # if␣
↪transition exits and emission match isnt found
                    #p = (transition[(prev_tag, pos_tag)])  * 1/
↪(pos_freq[prev_tag]+len(vocab)) # trying  laplace smoothing
                    p = ((transition[(prev_tag, pos_tag)]+small_no)  *␣
↪small_no)     # multiply by small probablity


                else:
                    #p = 1/(pos_freq[pos_tag]+len(pos_freq)) *  1/
↪(pos_freq[prev_tag]+len(vocab))
                    p = small_no * small_no

                if p > max_p:                                       # update␣
↪best probability found
                        max_p = p
                        best_tag = pos_tag



            #if max_p == 0:
            #    best_tag = max(pos, key=pos.get)

            predicted_tags.append(best_tag)                         # add the␣
↪best_found tag to the predicted tags list
```

```
                prev_tag = best_tag                                  # update␣
↪prev tag as curr predicted tag

            correct_matches = np.sum(np.array(pos_tags) == np.
↪array(predicted_tags)) # calculate number of tags correctly predicted
            total_correct += correct_matches
            deno +=len(pos_tags)
            accuracy = correct_matches/len(pos_tags)
            #print(accuracy)
            #print(pos_tags,predicted_tags)
            total_accuracy += accuracy
            num+=1
            sentence_accuracies.append(accuracy)
            predicted_tags.append(predicted_tags)
            idx = 0
            for word, pos_tag in zip(words, predicted_tags):
                idx += 1
                print ("%d\t%s\t%s" % (idx, word, pos_tag), file=tsvfile)
            print ("", file=tsvfile)

        print(total_accuracy, num)
        print("GREEDY sentence wise accuracy", total_accuracy/num*100)
        print("GREEDY word wise accuracy", total_correct/deno*100)

greedy_decoding(dev_df)
```

```
5151.380460710523 5527
GREEDY sentence wise accuracy 93.20391642320469
GREEDY word wise accuracy 93.51132293121243
```

[16]:

```
def viterbi_decoding(df):

    prev_tag= '< * >'           # start position tag as prev tag                 ␣
↪
    predicted_tags = []         # save the predicted tags
    sentence_accuracies = []    # append the sentence accuracies
    total_accuracy = 0          # add up total accuracy
    num = 0                     # count number of sentences
    total_correct = 0
    deno = 0
    small_no = 0.00000001
    with open('viterbi.out', 'w') as tsvfile:                  # open the .out file
        for idx, (_, words, pos_tags) in df.iterrows():        # read the df line␣
↪by line
            predicted_tags = []                                # reset for every␣
↪new sentence
            T = len(words)
```

```
                N = len(pos_freq.items())
                viterbi = [[0]*(T) for _ in range(N)]
                path_tracker = [[0]*(T) for _ in range(N)]
                for i in range(0,len(path_tracker)):
                    path_tracker[i][0] = -1

                for t, word in enumerate(words):           # for each timestamp␣
↪, word in words
                    if word not in vocab:                  # if word is not in␣
↪vocab give it  < unk > tag
                        word = '< unk >'
                    for s1, pos_tag in enumerate(pos_freq.keys()):   # for␣
↪every state, current tag
                        if pos_tag == '< * >':
                            pass
                        elif t == 0:                          # handle first␣
↪word separately
                            if ('< * >', pos_tag) in transition.keys() and␣
↪(pos_tag, word) in emission.keys():
                                viterbi[s1][0] = (transition[('< * >',␣
↪pos_tag)]+small_no) * (emission[(pos_tag, word)]+small_no)
                                path_tracker[s1][0] = -1      # to track end␣
↪when backtracking to find the path
                        else:
                            for s2, prev_tag in enumerate(pos_freq.keys()): #␣
↪for every possible prev state, prev tag
                                if prev_tag == '< * >':                     #␣
↪ignore start symbol tag
                                    pass
                                elif (prev_tag, pos_tag) in transition.keys()␣
↪and (pos_tag, word) in emission.keys():
                                    p = viterbi[s2][t-1]*␣
↪(transition[(prev_tag, pos_tag)]+small_no) * (emission[(pos_tag,␣
↪word)]+small_no)
                                    if p  > viterbi[s1][t]:              # save␣
↪the best transition probability in the the viterbi table
                                        viterbi[s1][t] = p
                                        path_tracker[s1][t] = s2         # save␣
↪best prev pos in this path

                best_path_p = 0
                best_path_index = 0
                for i in range(N):                                       # find␣
↪the best path start by looking at the last column in viterbi table
                    if viterbi[i][T-1] > best_path_p:
                        best_path_p = viterbi[i][T-1]
```

```
            best_path_index = i

        best_path = []
        t = T-1

        pos_list = list(pos_freq.keys())
        while best_path_index != -1 and t!=-1:                             ␣
↪   # backtrack to find the path that gave the best probability
          best_path.append(pos_list[best_path_index])
          #print("index",best_path_index)
          best_path_index = path_tracker[best_path_index][t]
          t -= 1
        if len(best_path) == 0:
          print("best path not found")



        best_path = best_path[::-1]                                  #␣
↪reverse the best path tags found by backtracking
        correct_matches = np.sum(np.array(pos_tags) == np.
↪array(best_path)) # count correct matches for accuracy calculations
        total_correct +=  correct_matches
        deno+=len(pos_tags)
        accuracy = correct_matches/len(pos_tags)
        sentence_accuracies.append(accuracy)
        predicted_tags.append(best_path)
        total_accuracy += accuracy
        num+=1
        idx = 0
        for word, pos_tag in zip(words, best_path):
          idx += 1
          print ("%d\t%s\t%s" % (idx, word, pos_tag), file=tsvfile)
        print ("", file=tsvfile)

  print("VITERBI sentence wise accuracy", total_accuracy/num*100)
  print("VITERBI word wise accuracy ", total_correct/deno*100)


viterbi_decoding(dev_df)
```

```
VITERBI sentence wise accuracy 94.42883547709638
VITERBI word wise accuracy  94.76883613623946
```

[17]:
```
test_data = []
sentence = []
pos_tags = []
```

```
sentence_count = 1
with open('data/test', newline = '') as tsvfile:                            #␣
 ↪read test file line by line
    csv_reader = csv.reader(tsvfile, delimiter = '\t')                      #␣
 ↪create sentences df like train_data
    for row in csv_reader:
        try:
            sentence.append(row[1])
        except Exception as e:
            test_data.append([sentence_count, sentence])
            sentence_count += 1
            sentence = []
            pos_tags = []

    test_data.append([sentence_count, sentence])

test_df = pd.DataFrame(test_data,columns = ['sentence_number', 'words'])
test_df
```

[17]:

```
      sentence_number                                              words
0                   1  [Influential, members, of, the, House, Ways, a…
1                   2  [The, bill, ,, whose, backers, include, Chairm…
2                   3  [The, bill, intends, to, restrict, the, RTC, t…
3                   4  [``, Such, agency, `, self-help, ', borrowing,…
4                   5  [The, complex, financing, plan, in, the, S&L, …
…                 …                                                …
5457             5458  [Says, Peter, Mokaba, ,, president, of, the, S…
5458             5459  [They, never, considered, themselves, to, be, …
5459             5460  [At, last, night, 's, rally, ,, they, called, …
5460             5461  [``, We, emphasize, discipline, because, we, k…
5461             5462  [``, We, want, to, see, Nelson, Mandela, and, …

[5462 rows x 2 columns]
```

[18]:

```
import numpy as np
import random

def greedy_decoding_test(df):
      with open('greedy.out', 'w') as tsvfile:
        prev_tag = '< * >'                                   # start position tag␣
 ↪as prev tag
        predicted_tags = []
        sentence_accuracies = []
        total_accuracy = 0
        small_no = 0.00000001                             # giving a very small␣
↪probability to non existent proabilities to preserve matched emission /␣
↪transition probabilities

        for idx, (_, words) in df.iterrows():   # loop through the data
            predicted_tags = []                           # keep track of␣
↪predicted_tags so far
            prev_tag = '< * >'                            # start with prev tag␣
↪as star position tag
            for word in words:                            # go word by word in␣
↪the sentence
                max_p = 0                                 # variable to keep␣
↪track of max probability seen so far
                best_tag = random.choice(list(first_tags))       # randomly␣
↪choose a start POS tag
                for pos_tag in pos_freq.keys():            # check probabilties␣
↪of each plausible tag
                    if word not in vocab:
                        word = '< unk >'                   # give < unk >␣
↪frequency value for words not seen
                    if (prev_tag, pos_tag) in transition.keys() and (pos_tag,␣
↪word) in emission.keys():
                        p = (transition[(prev_tag, pos_tag)]+small_no) *␣
↪(emission[(pos_tag, word)]+small_no)

                    elif (pos_tag, word) in emission.keys() :          # if␣
↪emission is found and transition is not matched
                        p = (emission[(pos_tag, word)]+ small_no)  * small_no

                    elif (prev_tag, pos_tag) in transition.keys() : # if exits␣
↪and emission match isnt found
                        p = (transition[(prev_tag, pos_tag)]+ small_no) * ␣
↪small_no



                    else:
                        p = small_no * small_no
                    if p > max_p:                                  # update␣
↪best probability found
                        max_p = p
                        best_tag = pos_tag




                predicted_tags.append(best_tag)                    # add␣
↪the best_found tag to the predicted tags list
```

```
                prev_tag = best_tag                                    # update␣
↪prev tag as curr predicted tag

            predicted_tags.append(predicted_tags)
            idx = 0
            for word, pos_tag in zip(words, predicted_tags):
                idx += 1
                print ("%d\t%s\t%s" % (idx, word, pos_tag), file=tsvfile)
            print ("", file=tsvfile)


greedy_decoding_test(test_df)
```

[20]:

```
def viterbi_decoding_test(df):

    prev_tag= '< * >'          # start position tag as prev tag                ␣
↪
    predicted_tags = []        # save the predicted tags
    sentence_accuracies = []   # append the sentence accuracies
    total_accuracy = 0
    num = 0
    small_no = 0.00000001

    with open('viterbi.out', 'w') as tsvfile:              # open the .out file
        for idx, (_, words) in df.iterrows():              # read the df line␣
↪by line
                predicted_tags = []
                T = len(words)
                N = len(pos_freq.items())
                viterbi = [[0]*(T) for _ in range(N)]
                path_tracker = [[0]*(T) for _ in range(N)]
                for i in range(0,len(path_tracker)):
                    path_tracker[i][0] = -1

                for t, word in enumerate(words):           # for each timestamp␣
↪, word in words
                    if word not in vocab:                  # if word is not in␣
↪vocab give it  < unk > tag
                        word = '< unk >'
                    for s1, pos_tag in enumerate(pos_freq.keys()):   # for␣
↪every state, current tag
                        if pos_tag == '< * >':
                            pass
                        elif t == 0:                          # handle first␣
↪word separately
```

```
                            if ('< * >', pos_tag) in transition.keys() and␣
↪(pos_tag, word) in emission.keys():
                                viterbi[s1][0] = (transition[('< * >',␣
↪pos_tag)] +small_no) * (emission[(pos_tag, word)]+small_no)
                                path_tracker[s1][0] = -1      # to track end␣
↪when backtracking to find the path
                        else:
                            for s2, prev_tag in enumerate(pos_freq.keys()): #␣
↪for every possible prev state, prev tag
                                if prev_tag == '< * >':                     #␣
↪ignore start symbol tag
                                    pass
                                elif (prev_tag, pos_tag) in transition.keys()␣
↪and (pos_tag, word) in emission.keys():
                                    p = viterbi[s2][t-1]*(transition[(prev_tag,␣
↪pos_tag)]+small_no) * (emission[(pos_tag, word)]+small_no)
                                    if p  > viterbi[s1][t]:              # save␣
↪the best transition probability in the the viterbi table
                                        viterbi[s1][t] = p
                                        path_tracker[s1][t] = s2         # save␣
↪best prev pos in this path

                best_path_p = 0
                best_path_index = 0
                for i in range(N):                                       # find␣
↪the best path start by looking at the last column in viterbi table
                    if viterbi[i][T-1] > best_path_p:
                        best_path_p = viterbi[i][T-1]
                        best_path_index = i

                best_path = []
                t = T-1

                pos_list = list(pos_freq.keys())
                while best_path_index != -1 and t!=-1:                         ␣
↪   # backtrack to find the path that gave the best probability
                    best_path.append(pos_list[best_path_index])
                    best_path_index = path_tracker[best_path_index][t]
                    t -= 1
                if len(best_path) != len(words) :
                    print("best path not found")



                best_path = best_path[::-1]                              #␣
↪reverse the best path tags found by backtracking
```

```
                idx = 0
                for word, pos_tag in zip(words, best_path):
                    idx += 1
                    print ("%d\t%s\t%s" % (idx, word, pos_tag), file=tsvfile)
                print ("", file=tsvfile)



viterbi_decoding_test(test_df)
```

[ ]: