Hee Ji Park

CSCI544 (NLP)

10.20.2021

Report – HW3 (CSCI544)

Information

Python version	3.6.12
Jupyter notebook version	6.1.4
Package and libraries	import pandas as pd import json
How to run?	- In case of .ipynb file(Hee Ji Park - HW3.ipynb), **you can run my code using Jupyter notebook**. (* I wrote the code using the jupyter notebook.) - In case of .py file (HeeJiPark-HW3.py), you can run my code using terminal => Command line in the Terminal: **python HeeJiPark-HW3.py** => you can get [vocab.txt / hmm.json / greedy.out / viterbi.out] files automatically by running this code.
Explorations	- I tried to change all words to lowercase, but this resulted in performance degradation. - I have replaced words containing numbers with <num> tokens, which improved performance. - I have subdivided the <unk> token. The <unk> token is subdivided according to whether the unknown word contains a number or has the characteristics of a noun/verb/adjective/adverb.

Task1. Vocabulary Creation

What is the selected threshold for unknown words replacements?	2
What is the total size of your vocabulary?	21578
What is the total occurrences of the special token '<unk>' after replacement?	17343

Task2. Model Learning

How many transition parameters in your HMM?	2116
How many emission parameters in your HMM?	28754

Task3. Greedy Decoding with HMM

What is the accuracy on the dev data using Greedy decoding?	93.88

Task4. Viterbi Decoding with HMM

What is the accuracy on the dev data using Viterbi decoding?	94.84

Hee Ji Park (4090715830) - HW3

- Python version 3.6.12
- Jupyter notebook version 6.1.4

1. Task1 : Vocabulary Creation

In [1]:

```
# If the word is number, return True. Or return False
def isNumber(s):
    try:
        if ',' in s: # ex) 4,800 -> 4800
            s = s.replace(',','')
        float(s)
        return True
    except ValueError:
        return False
```

In [2]:

```
import string
punct = set(string.punctuation)
noun_suffix = ["let",'ie',"kin","action", "ling", "hood", "ship", "ary","age",
               "ery", "ory", "ance", "an","ary","eer","er","ier","herd","cy", "dom",
               "ee", "ence", "ster", "yer", "ant","ar", "ion", "ism", "ist", "ity",
               "ment", "ness", "or", "ry", "scape", "ty"]
verb_suffix = ["ate", "ify", "ize", "ise"]
adj_suffix = ["able", "ible", 'ant', 'ent', 'ive', "al","ial","an","ian","ish",
              "ern", "ese", "ful", 'ar', 'ary','ly','less','ic','ive','ous', "i", "ic"]
adv_suffix = ["ly","lng","ward", "wards", "way", "ways", "wise"]

def unk_preprocessing(s):
    # If unknown word has number, return <unk_num> token
    if any(char.isdigit() for char in s):
        return "<unk_num>"

    # If unknown word has punctuation, return <unk_punt> token
    elif any(char in punct for char in s):
        return "<unk_punct>"

    # If unknown word has upper characters, return <unk_upper> token
    elif any(char.isupper() for char in s):
        return "<unk_upper>"
    # If unknown word contains characteristics of noun, return <unk_noun> token
    elif any(s.endswith(suffix) for suffix in noun_suffix):
        return "<unk_noun>"

    # If unknown word contains characteristics of verb, return <unk_verb> token
    elif any(s.endswith(suffix) for suffix in verb_suffix):
        return "<unk_verb>"

    # If unknown word contains characteristics of adj, return <unk_adj> token
    elif any(s.endswith(suffix) for suffix in adj_suffix):
        return "<unk_adj>"

    # If unknown word contains characteristics of adverbs, return <unk_adv> token
    elif any(s.endswith(suffix) for suffix in adv_suffix):
        return "<unk_adv>"

    else:
        return "<unk>"
```

In [3]:

```
# Vovabulary Creation
def vocab_creation(file, min_count):
    vocab = {}
    tag = {}
    with open(file, "r") as train:
        for line in train:
            if not line.split(): # Ignore a blank line
                continue
            word, wordtag = line.split("₩t")[1], line.split("₩t")[2].strip('₩n')
            if isNumber(word): # if word is number, change number to '<num>' token
                word = '<num>'
            if word not in vocab: # store {vocabrary : frequency}
                vocab[word] = 1
            else:
                vocab[word] += 1
            if wordtag not in tag: # store {tag : frequency}
                tag[wordtag] = 1
            else:
                tag[wordtag] += 1


        # make <unk> token
        vocab['<unk>'], vocab['<unk_num>'] = 0,0
        vocab['<unk_punct>'], vocab["<unk_upper>"] = 0,0
        vocab["<unk_noun>"], vocab["<unk_verb>"] = 0,0
        vocab["<unk_adj>"], vocab["<unk_adv>"] = 0,0
        delete = []
        for word, occurrences in vocab.items():
            if occurrences >= min_count: # If occurrences is bigger than 3 : Pass
                continue
            else:
                new_tag = unk_preprocessing(word)
                vocab[new_tag] += occurrences   # If occurrences is lower than 3 : change word name to < unk >
                delete.append(word) # To remove the word in the dictionary (vocab), store 'word' in the delete list

        for i in delete:
            del vocab[i] # Remove the word in the vocab dictionary
        # For save unk token and corresponding occurrences of <unk> token
        unk1 = ('<unk>',vocab['<unk>'])
        unk2 = ('<unk_num>',vocab['<unk_num>'])
        unk3 = ('<unk_punct>',vocab['<unk_punct>'])
        unk4 = ('<unk_upper>',vocab['<unk_upper>'])
        unk5 = ('<unk_noun>',vocab['<unk_noun>'])
        unk6 = ('<unk_verb>',vocab['<unk_verb>'])
        unk7 = ('<unk_adj>',vocab['<unk_adj>'])
        unk8 = ('<unk_adv>',vocab['<unk_adv>'])

        # Remove <unk> token in the vocab, because we have to put <unk> token on the top rows
        del vocab['<unk>']
        del vocab['<unk_num>']
        del vocab['<unk_punct>']
        del vocab['<unk_upper>']
        del vocab['<unk_noun>']
        del vocab['<unk_verb>']
        del vocab['<unk_adj>']
        del vocab['<unk_adv>']

        # Sort the occurrences in the dict
        vocab_to_list = sorted(vocab.items(), key=lambda x: x[1], reverse=True)

        # Put <unk> token on the top rows

        tot_unk = [unk8,unk7,unk6,unk5,unk4,unk3,unk2,unk1]
        tot_unk = sorted(tot_unk, key=lambda x: x[1], reverse=True)
        for unk in tot_unk[::-1]:
            vocab_to_list.insert(0,unk)

        # Make a vocab text file
        with open('vocab.txt', "w") as out:
            for idx,line in enumerate(vocab_to_list):
                out.write("{0}₩t{1}₩t{2}₩n".format(line[0],idx,line[1]))
            out.close()

        return vocab_to_list, tag
```

In [4]:

```
vocab, tag = vocab_creation('./data/train',2)
```

In [5]:

```
print("What is the selected threshold for unknown words replacement? : 2")
print("What is the total size of your vocabulary? : %d" % len(vocab))
print("What is the total occurrences of the special token '<unk>' after replacement? : %d" % (vocab[0][1] + vocab[1][1] + v
```

```
What is the selected threshold for unknown words replacement? : 2
What is the total size of your vocabulary? : 21578
What is the total occurrences of the special token '<unk>' after replacement? : 17343
```

2. Task 2 : Model Learning

In [6]:

```
# Make tag sentences for transition
def make_sentences_for_transition(file):
    for_transition = []
    with open(file, "r") as train:
        sentence = 'Φ'
        for line in train:
            if not line.split(): # In case of a blank line, make new sentence
                for_transition.append(sentence)
                sentence='Φ' # mark to distinguish the first letter of a sentence
                continue
            tag = line.split("₩t")[2].strip('₩n')
            sentence = sentence + ' ' + tag

    return for_transition
```

In [7]:

```
# Make a transition dictionary
def make_transition(tag, for_transition):
    transition = {}
    for pre, pre_cnt in tag.items():
        for cur, cur_cnt in tag.items():
            tot = 0
            for sentence in for_transition: # Find 'tag tag' in each sentence
                find = pre + ' ' + cur
                if find in sentence: # Count(s->s')
                    tot += 1

            transition[str((pre,cur))] = tot/pre_cnt # Calculate t(s'|s)
    return transition
```

In [8]:

```
for_transition = make_sentences_for_transition('./data/train')
tag['Φ'] = len(for_transition) # Insert initial tag into the tag dictionary
transition = make_transition(tag, for_transition)
```

In [9]:

```
print('How many transition parameters in my HMM? : %d'%len(transition))
```

```
How many transition parameters in my HMM? : 2116
```

In [10]:

```
voca = [x[0] for x in vocab]
```

In [11]:

```
# Make a sentences for emission
def make_sentences_for_emission(file):
    with open(file, "r") as train:
        for_emission = {}
        for line in train:
            if not line.split(): # Ignore a blank line
                continue
            word, wordtag = line.split("₩t")[1], line.split("₩t")[2].strip('₩n')
            if isNumber(word): # number to '<num>' token
                word = '<num>'
            else:
                if word not in voca:
                    word = unk_preprocessing(word)

            x = word + ' ' + wordtag # Count(s->x)
            if x not in for_emission:
                for_emission[x] = 1
            else:
                for_emission[x] += 1

    return for_emission
```

In [12]:

```
# Make a emission dictionary
def make_emission(tag, for_emission):
    emission={}
    for comb, cnt in for_emission.items():
        eword = comb.split(" ")[0]
        etag = comb.split(" ")[1]

        emission[str((etag,eword))] = cnt / tag[etag] # calculate e(x|s)
    return emission
```

In [13]:

```
for_emission = make_sentences_for_emission('./data/train')
emission = make_emission(tag, for_emission)
```

In [14]:

```
print('How many emission parameters in my HMM? : %d'%len(emission))
```

```
How many emission parameters in my HMM? : 28754
```

In [15]:

```
# Make a hmm.json file
import json

hmm = {}
hmm['transition'] = transition
hmm['emission'] = emission

with open('hmm.json', 'w') as outfile:
    json.dump(hmm, outfile, indent='₩t')
```

Task3 : Greedy Decoding with HMM

In [16]:

```
# Greedy algorithm
def greedy(file, tag):
    state = []
    pre = 'Φ' # To distinguosh sentence, use this mark. 'Φ' is the first letter of the sentence
    with open(file, "r") as train:
        sentence = []
        for line in train:
            p = []
            if not line.split():
                pre='Φ'
                continue

            word = line.split("₩t")[1].strip('₩n')
            if isNumber(word): # change number to '<num>' token
                word = '<num>'
            else:
                if word not in voca:
                    word = unk_preprocessing(word)
            for cur in list(tag.keys()):
                emission_val = 0 if str((cur,word)) not in emission else emission[str((cur,word))]
                prob = emission_val * transition[str((pre,cur))] # calculate probabilities
                p.append(prob)
            state_max = list(tag.keys())[p.index(max(p))]
            state.append(state_max)    # put tag to the'state' list
            pre = state_max

    return state
```

In [17]:

```
state = greedy('./data/dev',tag)
```

In [18]:

```
# Evaluate greedy decoding
def greedy_evaluate(file,state):
    with open(file, "r") as train:
        taglist = []
        for line in train:
            if not line.split():
                continue
            tag = line.split("₩t")[2].strip('₩n')
            taglist.append(tag)

    # compare the predicted tag and real tag
    cnt=0
    for greedy, real in zip(state, taglist):
        if greedy == real:
            cnt += 1

    accuracy = (cnt / len(state)) * 100
    return accuracy
```

In [19]:

```
# Evaluate greedy decoding
def greedy_evaluate(file,state):
    with open(file, "r") as train:
        taglist = []
        wordlist=[]
        for line in train:
            if not line.split():
                continue
            word = line.split("₩t")[1]
            tag = line.split("₩t")[2].strip('₩n')
            taglist.append(tag)
            wordlist.append(word)

    # compare the predicted tag and real tag
    cnt=0
    for greedy, real, word in zip(state, taglist, wordlist):
        if greedy == real:
            cnt += 1

    accuracy = (cnt / len(state)) * 100
    return accuracy
```

In [20]:

```
accuracy = greedy_evaluate('./data/dev',state)
print("Greedy Algorithm's Accuracy =%0.2f" %accuracy)
```

```
Greedy Algorithm's Accuracy =93.88
```

In [21]:

```
test_tag = greedy('./data/test',tag) # predict test file
```

In [22]:

```
# make a sentence using test_tag
write=[]
with open('./data/test', "r") as test:
    sentence=''
    cnt = 0
    for line in test:
        if not line.split():
            write.append('*****')
            continue
        index, word = line.split("₩t")[0], line.split("₩t")[1].strip('₩n')

        pred_tag = test_tag[cnt]
        sentence = index + '₩t' + word + '₩t' + pred_tag + '₩n'
        write.append(sentence)
        cnt += 1
```

In [23]:

```
# store the result in a file named 'greedy.out'
with open('greedy.out', "w") as out:
    for line in write:
        if line == '*****':
            out.write('₩n')
        else:
            out.write(line)
    out.close()
```

Task4 : Viterbi Decoding with HMM

In [24]:

```
import pandas as pd
```

In [25]:

```
def viterbi(file, tag):
    with open(file, "r") as train:
        dic = []
        state = []
        sentence = []
        one_sentence = []
        # Make sentences to calculate conveniently
        for line in train:
            if not line.split():
                sentence.append(one_sentence)
                one_sentence = []
                continue
            word = line.split("₩t")[1].strip('₩n')
            if isNumber(word):
                word = '<num>'
            else:
                if word not in voca:
                    word = unk_preprocessing(word)
            one_sentence.append(word)
        sentence.append(one_sentence)

        # Calculate the first word of the sentence
        for idx, s in enumerate(sentence):
            Viterbi = [{}]
            for st in list(tag.keys())[:-1]:
                transition_val = transition[str(('Φ',st))]
                #If word does not exist in the train data, assign a very low probability of 0.0000000000001
                emission_val =  0.0000000001 if str((st,sentence[idx][0])) not in emission else emission[str((st,sentence[
                Viterbi[0][st] = {"prob": transition_val * emission_val, "prev": None}


            # Run Viterbi algorithm with the remain of the sentence
            for t in range(1, len(sentence[idx])):
                Viterbi.append({})
                states = list(tag.keys())[:-1]
                for st in states:
                    max_transition_prob = Viterbi[t - 1][states[0]]["prob"] * transition[str((states[0],st))] # max transit
                    previous_selected = states[0]
                    for prev in states[1:]:
                        transition_prob = Viterbi[t - 1][prev]["prob"] * transition[str((prev,st))]
                        if transition_prob > max_transition_prob:
                            max_transition_prob = transition_prob
                            previous_selected = prev

                    #If word does not exist in the train data, assign a very low probability of 0.0000000000001
                    emission_val = 0.0000000001 if str((st,sentence[idx][t])) not in emission else emission[str((st,senten
                    max_prob = max_transition_prob * emission_val
                    Viterbi[t][st] = {"prob": max_prob, "prev": previous_selected}
            predicted = []
            max_prob = 0
            best_st = None

            # FInd most highest probabilities and save it
            for st, item in Viterbi[-1].items():
                if item["prob"] > max_prob:
                    max_prob = item["prob"]
                    best = st
            predicted.append(best)
            previous = best

            # backtracking
            for t in range(len(Viterbi) - 2, -1, -1):
                predicted.insert(0, Viterbi[t + 1][previous]["prev"])
                previous = Viterbi[t + 1][previous]["prev"]

            dic.append(predicted)

        return dic
```

In [26]:

```
dic = viterbi('./data/dev',tag)
```

In [27]:

```
def viterbi_evaluate(file,dic):
    with open(file, "r") as train:
        taglist = []
        one_sentence =[]
        # make a sentence that consists of the tags
        for line in train:
            if not line.split():
                taglist.append(one_sentence)
                one_sentence = []
                continue
            tag = line.split("₩t")[2].strip('₩n')
            one_sentence.append(tag)
        taglist.append(one_sentence)

    tot_cnt=0
    same_cnt=0
    for i in range(len(taglist)):
        list_len = len(dic[i])
        tot_cnt += list_len
        for k in range(list_len):
            if dic[i][k] == taglist[i][k]:
                same_cnt += 1

    accuracy = (same_cnt / tot_cnt) * 100
    return accuracy
```

In [28]:

```
accuracy = viterbi_evaluate('./data/dev',dic)
print("Viterbi Algorithm's Accuracy =%0.2f" %accuracy)
```

```
Viterbi Algorithm's Accuracy =94.84
```

In [29]:

```
test_dic = viterbi('./data/test',tag)
```

In [30]:

```
# 1. for storing the result in the file - make sentences
write=[]
for i in range(len(test_dic)):
    for j in range(len(test_dic[i])):
        write.append(test_dic[i][j])
```

In [31]:

```
# 2. for storing the result in the file - make a same format of training data
write2=[]
with open('./data/test', "r") as test:
    sentence=''
    cnt = 0
    for line in test:
        if not line.split():
            write2.append('*****')
            continue
        index, word = line.split("₩t")[0], line.split("₩t")[1].strip('₩n')

        pred_tag = write[cnt]
        sentence = index + '₩t' + word + '₩t' + pred_tag + '₩n'
        write2.append(sentence)
        cnt += 1
```

In [32]:

```
# for storing the result in the file - store data
with open('viterbi.out', "w") as out:
    for line in write2:
        if line == '*****':
            out.write('₩n')
        else:
            out.write(line)
    out.close()
```