Assignment 1: Advanced Text Processing

Zhe Han

November 25, 2020

1 POS-Tagging

1 The best version of the tagger was tuned by -t 3 with accuracy: 93.09%. Below are my detailed manual error analyses based on it.

Sentence-From the AP comes this story:

AP was wrongly tagged as NOUN and it should be PRON. Because in the training text, it was only assigned tag NOUN, then we need to enlarge our corpus in training text. Even though we enlarge our corpus, it's still difficult to distinguish tag NOUN or PRON for word AP, in this case, it would be ambiguous.

Sentence-Bush also nominated A. Noel Anketell Kramer for a 15-year term as associate judge of the District of Columbia Court of Appeals ,

associate was incorrectly tagged as NOUN and it should be ADJ. The reason is also that associate in the training text was only tagged as noun whereas the word associated was tagged as ADJ in some sentences. In my opinion, associate is an ambiguous word to tag due to its 3 parts-of-speech. For example, if it is after the infinitive 'to', then it will be VERB; if it is used as VERB+associate+NOUN, it should be ADJ; and if it is presented as DET+ADJ+associate, it will be NOUN.

- was wrongly tagged as SYM and it should be PUNCT. In the training file, - can be tagged as both SYM and PUNCT, thus this is very ambiguous. To my mind we should change the token to 15-year, then it will much easier for us to correctly tag.

Sentence-He could be killed years ago and the israelians have all the reason,

israelians should be tagged as PROUN. The reason why it is tagged as NOUN is that it follows a DET word which complies with the training corpus rule: DET will be followed by a NOUN. This one is unambiguous and we simply need to tag the exact word israelians in the training corpus.

Sentence-since he founded and he is the spiritual leader of Hamas, but they did n't.

This sentence omitted the VERB after did n't to avoid redundancy. This is also an ambiguous word because of different POS. The word 'did' here should be AUX, so it was mistagged in the gold standard.

Sentence-Today 's incident proves that Sharon has lost his patience and his hope in peace.

Hope should be tagged as NOUN not VERB. Similarly, it's an ambiguous word due to 2 POS. It was mistagged as VERB because it follows the PROUN, which is in accordance with a rule of the training corpus. If we want to correct it, we should subdivide PROUN to nominative pronoun and possessive nominal pronoun. For example, if hope follows possessive nominal pronoun, it should be NOUN rather than VERB.

2 Currently, There are CHINESE PENN TREEBANK PART-OF-SPEECH TAGSET, CHINESE NEUSCP PART-OF-SPEECH TAGSET,CHINESE SYMBOL PART-OF-SPEECH TAGSET and so on. CHINESE NEUSCP PART-OF-SPEECH TAGSET was created by linguists in a Chinese university. This tagset includes comprehensive tags for Chinese special word classifications - location noun and temporal noun. Below are some detailed examples.

Tag	Description
n	common noun (e.g place,worker)
nt	temporal noun (e.g. past time,future)
nd	noun of locality (e.g. down,up)
nl	location noun (e.g. inland,coast)

3 First of all, POS tagging proceeds in allocating a POS or other syntactic classifier to every element and tags are extensively used in punctuation, thus tagging demanding that punctuation marks be peeled off of the words. For example, it can distinguish end-of-sentence punctuation (period) from word-internal punctuation (etc.). Secondly, Tokenization breaks up the original text into separate words, symbols and so on which assist in comprehending background information of the text as well as cleaning the unstructured text data (e.g. splitting contractions and ’s-genitive from stems). More importantly, this process contributes to elucidating implication of the raw data via perceiving continuous character lists, then each token will be included and tagged in a more precise way. Eventually, tokenization will optimize NLP models.

4 Theoretically, HMM model is a likelihood sequence model which matches a list of observed incidents to a tag sequence, in this case, the key sequence of numbers are observation events (emitted signals) and predicted words are all possible word sequences (hidden state of model) in HMM lab. As for applying HMM model in POS tagging, the emitted signals are a sequence of words and POS tags are hidden state. If we calculate the probability spread of all possible word tagged sequences and select the most probable one, we would then need transition probability (referring to the probability of transferring from state i to state j) as well as emission probabilities (each indicating the probability of an observation o being generated from a state q). In summation, we observe a word sequence, then deduce the hidden tags from it, thus both key signal sequence and forecast tags are part of the core foundation of HMM models for POS tagging.

2 Lemmatisation

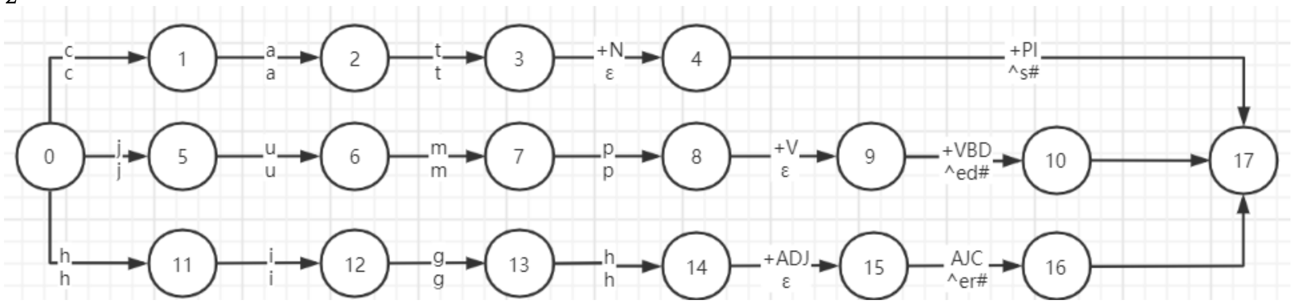
1 All my error analyses below are based on my modified lemmatizer.py with 96.77% accuracy:
lost VERB lost
The lemma of ’lost’ should be ’lose’, and it’s a irregular verb inflecting in the past form, in this case, we should add one more sub-condition for it-if word == "lost": return "lose".
men NOUN men
The lemma of ’men’ should be ’man’, and it’s a irregular noun inflecting in the plural form, in this case, we simply add one more sub-condition for it-if word == "men": return "man".
submitted VERB submitt
The lemma of ’submitted’ should be ’submit’, and it’s a irregular verb inflecting in the past form, in this case, we judge if it ends with two repeated consonants after eliminating ’ed’, if so, we need remove one

more consonant.
saw VERB saw
The lemma of ’saw’ should be ’see’, and it’s a irregular verb inflecting in the past form, in this case, we need add one more sub-condition for it-if word == "saw": return "see".
other ADJ oth
The lemma of ’other’ should be ’other’, because it’s not a comparative form, then we are supposed to supplement one sub-condition for it-if word == "other": return "other".

Below are modified codes

```
import sys
vowel='aeiou'
def noun_lemma(word):
    if word.endswith("ies"):
        return word[:-3]+'y'
    elif word.endswith("ss"):
        return word
    elif word.lower()=='bus':
        return 'bus'
    elif word.endswith("es") and len(word)>3 and word[-4:-2] in ['sh','ch']:
        return word[:-2]
    elif word.endswith("es") and len(word)>2 and word[-3] in 'xs':
        return word[:-2]
    elif word.endswith("s"):
        return word[:-1]
    else:
        return word

def part_lemma(word):
    if word=='n\'t':
        return 'not'
    else:
        return word

def det_lemma(word):
    if word=='an':
        return 'a'
    else:
        return word

def pron_lemma(word):
    if word.lower()=='her' or word.lower()=='hers':
        return 'she'
    elif word.lower()=='his'or word=='him':
        return 'he'
    elif word.lower()=='our' or word.lower()=='ours':
        return 'we'
    elif word.lower()=='yours' or word.lower()=='your':
        return 'you'
    elif word.lower()=='their' or word.lower()=='theirs' or word=='them':
        return 'they'
    elif word=='me' or word=='I':
        return ’I’
    else:
        return word.lower()

def aux_lemma(word):
    if word.lower()==’is’ or word.lower()==’was’ or word.lower()==’are’ or word.lower()==’were’
    or word.lower()==’being’ or word.lower()==’been’ or word==’\’s’:
        return ’be’
    elif word.lower()==’has’ or word.lower()==’had’ or word==’\’ve’:
        return ’have’
    elif word==’\’d’:
        return ’would’
    elif word.lower()==’did’:
        return ’do’
    elif word.lower()==’ca’:
        return ’can’
    else:
        return word

def verb_lemma(word):
    lemma1=word[:-2]
    lemma2=word[:-3]
    if word==’is’ or word.lower()==’am’ or word.lower()==’was’ or word.lower()==’are’
    or word==’\’s’ or word.lower()==’were’ or word.lower()==’being’ or word.lower()==’been’:
        return ’be’
    elif word.lower()==’has’ or word.lower()==’had’:
        return ’have’
    elif word.lower()==’did’ or word.lower()==’done’:
        return ’do’
    elif word.endswith("ied"):
        return word[:-3]+’y’
    elif word.endswith("ed") and lemma1[-1] in vowel:
        return word[:-1]
    elif word.endswith("ed") and len(lemma1)>1 and lemma1[-1] not in vowel and lemma1[-2] in vowel:
        return word[:-1]
    elif word.endswith("ed"):
        return word[:-2]
    elif word.endswith("ing") and len(lemma2)>2 and lemma2[-1] not in vowel
    and lemma2[-2] in vowel and lemma2[-3] not in vowel:
        return word[:-3]+’e’
    elif word.endswith("ing"):
        return word[:-3]
    elif word.endswith("es") and len(lemma1)>1 and lemma1[-2:] ==’ss’:
        return word[:-2]
    elif word.endswith("s") and len(word)>1 and not word.endswith("ss"):
        return word[:-1]
    else:
        return word

def adj_lemma(word):
    if word.lower()==’best’:
        return ’best’
    elif word.lower()=='better':
        return 'better'
    elif word.endswith("er"):
        return word[:-2]
    elif word.endswith("est"):
        return word[:-3]
    else:
        return word

for line in sys.stdin:
    if line.strip():
        (word, tag) = line.strip().split("\t")
        lemma = word
        if tag == "NOUN":
            lemma = noun_lemma(word).lower()
        elif tag == "VERB":
            lemma = verb_lemma(word).lower()
        elif tag == "AUX":
            lemma = aux_lemma(word).lower()
        elif tag == "ADJ":
            lemma = adj_lemma(word).lower()
        elif tag == "PRON":
            lemma = pron_lemma(word)
        elif tag == "PROPN":
            lemma = word
        elif tag == "PART":
            lemma = part_lemma(word)
        elif tag == "DET":
            lemma = det_lemma(word).lower()
        else:
            lemma = word.lower()
        print("{0}\t{1}\t{2}".format(word, tag, lemma))
    else:
        print()
```

2



The inputs of FST are cats/NOUN,jumped/VERB, higher/ADJ and the output are cat +Pl ˆ s#, jump +VBD ˆ ed#, high AJC ˆ er#.

3 In Arabic language, every verb owns more than thirty distinct inflectional forms for each verb and there are numerous irregular nouns with various plural forms, if we would like to parse all of them correctly, we have to store all the morphological variants of each verb. Even if we create an enormous training corpus, morphologically rich languages still result in sparsity issues and it's unrealistic to

include overall forms. For example, standard models hypothesise that a word relates to a distinct terminal in the parsing process; however, in Arabic languages, an input word-token stands a high chance of relating to diverse terminals, which means there is a high proportion of words unobserved in the corpus data.

3 VG: Hidden Markov Models

Transition Probability:

Transition P	Pronoun	VERB	NOUN	ADP	DET
PRON	0	1/2	0	0	1/4
VERB	1/3	0	2/3	0	0
NOUN	0	1/5	0	2/5	0
ADP	1/2	0	1/2	0	0
DET	0	0	1/2	0	1/2

Initial	<s>
PRON	2/3
VERB	0
NOUN	1/3
ADP	0
DET	0

Emission Probability:

Emission P	she	books	trips	for	kids	you	all	the	time	like	these
PRON	1/2	0	0	0	0	1/4	0	0	0	0	1/4
VERB	0	1/3	0	0	1/3	0	0	0	0	1/3	0
NOUN	0	1/5	1/5	0	2/5	0	0	0	1/5	0	0
ADP	0	0	0	1/2	0	0	0	0	0	1/2	0
DET	0	0	0	0	0	0	1/2	1/2	0	0	0

Viterbi trellis of Test:

kids like books	kids	like	books
PRON	0	0	0
VERB	0	P(VERB|NOUN)*P(like|VERB) =(1/5)*(1/3)=1/15	0
NOUN	P(PROUN|<S>)*P(kids|N) =(2/3)*(2/5)=4/15	0	P(NOUN|VERB)*P(books|NOUN) =(2/3)*(1/5)=2/15

kids/NOUN likes/VERB books/NOUN
Probability=(4/15)*(1/15)*(2/15)=8/3375=0.00237