University of St Andrews

CS5012 Language and computation

Practical 1: Parts of speech in the Universal Dependencies treebanks

210016568

March 8, 2022

Abstract

This report is to illustrate how I implement the Practical 1 of CS5012 Language and Computation. It will be divided into 4 part, the algorithm 1, algorithm 2, algorithm 3, as well as evaluation.

1 Algorithm 1 – Eager Algorithm

For this algorithm, I defined tables for transition probability distribution (table A) and emission probability distribution (table B).

the work flow is that:

- get the tagset first (excluding start <s> and end </s>)
- for each tag ($tag \in\ tagset \cup strat$), use a dict to implement a map relationship: tag->list[ ], the list is to collect the tags appear after this tag. Then map tag->WittenBellProbDist(FreqDist(list[ ])) [1]. In this step, finally we get the table A.
- for each tag ($tag \in\ tagset$), use a dict to implement a map relationship: tag->list[ ], the list is to collect the words which is marked as this tag in trainset. Then map tag->WittenBellProbDist(FreqDist(list[ ])). In this step, finally we get the table B.
- define a method, the input is $tag_{i-1},\ word_i$, output is tag_i, enumerating all the possible condition for $A(tag_{i-1}, tag_i) \cdot B(tag_i, word_i)$ $(i > 0, tag_0 = start)$.
- find the $word_i$ to max the value on basis of tag_{i-1}, tag_i.

2 Algorithm 2 – Viterbi Algorithm

For this algorithm, I defined the viterbi algorithm based on the table A and table B from algorithm 1. The workflow referred the wikipedia[4].

I implemented it based on the following workflow:
given:table A, table B, the set of all possible tags (excluding start and end)
input: a sentence in list. (e.g., ['Will', 'can', 'spot', 'Marry'])

- defined a table V and a list path. V[i][tag] will store the max sum logprob of path-to-this-tag. logprob $= log(A(tag_{i-1}, tag_i)) + log(B(tag_i, word_i))$. path is to store the path from start to the current tag.
- initialize V[0][tag] $= log(A(start, tag_i)) + log(B(tag_i, word_i))$.
- use the for loop to enumerate all the tag for calculate V[i][tag].
- do the above for loop for every i (i=1,2, $\cdots$,n; n = length of sentence).
- for each tag in V[n][tag], V[n][tag] = V[n][tag] + $log(A(tag_n, end))$.
- find the max logprob, then return its path (i.e., the tags predicted).

Reason why from $A(tag_{i-1}, tag_i) \cdot B(tag_i, word_i)$ to $log(A(tag_{i-1}, tag_i)) + log(B(tag_i, word_i))$:

When given $p_1 = 0.1, p_2 = 0.01, p_3 = 0.2, p_4 = 0.02$

$$p_1 \cdot p_2 < p_3 \cdot p_4$$
$$log(p_1 \cdot p_2) < log(p3.p_4)$$
$$logp_1 + logp_2 < logp_3 + logp_4$$

3 Algorithm 3 – Baum-Welch Algorithm

For this algorithm, I defined the Baum-Welch algorithm based on the table A and table V from algorithm 1.

I implemented it based on the following workflow:
given:table A, table B, the set of all possible tags (excluding start and end)
input: a sentence in list. (e.g., ['Will', 'can', 'spot', 'Marry'])

- defined 2 functions to get the forward table α and backward table β for a sentence.
  - for the table α, α[i][tag] will store the max sum logprob from start to this tag.
    - initialize the α[0][start] = 0 (this actually is $log\ P(tag_0 = start) = log1 = 0$).
    - initialize the α[1][tag] = $log(A(start, tag_i)) + log(B(tag_i, word_i))$.
    - use the for loop to enumerate all the tag for calculate α[i][tag].
    - do the above for loop for every i (i=1,2, $\cdots$,n; n = length of sentence -1).
    - return the table α.
  - for the table β, β[i][tag] will store the max sum logprob from end to this tag.
    - initialize the β[-1][tag] = 0 (this actually is $log\ P(tag_{-1} = end) = log1 = 0$)
    - initialize the β[-2][tag] = $log(A(tag, end))$.
    - use the for loop (from backward to forward) to enumerate all the tag for calculate β[i][tag].
    - do the above for loop for every i (i=n,n-1, $\cdots$,1; n = length of sentence - 1).
    - return the table β.
- define the Baum-Welch algorithm for predicting the tags based on the table α and table β.
  - for each word in sentence, find all the combinations of α(i, tag) + β(i, tag) when given a certain tag.
  - find the tag with max logprob, add this tag into the list of predicted tags.
  - return the list of predicted tags.

In the both two functions, in the 3rd step, for calculating the alpha/beta[i][tag], I took the logsumexp[3] to calculate them.

An example for this algorithm is that, input sentence is "I finished", the table alpha (after taking log) is:

```
[{’<s>’: 0},

{’ADP’: -27.764689725388443,
’SCONJ’: -26.582901429241502,
’NOUN’: -22.903915885904574,
’ADJ’: -23.838361972956385,
’ADV’: -23.91713984195928,
’CCONJ’: -29.47927616051139,
’NUM’: -23.893174531881336,
’VERB’: -23.355054604798546,
’X’: -23.796884784237307,
’PROPN’: -17.16765773363059,
’PUNCT’: -29.134162011083596,
’PRON’: -4.415834513667509,
’SYM’: -26.883530405691495,
’INTJ’: -24.279350816512753,
’DET’: -27.83800838469361,
’PART’: -32.44208804614421,
’AUX’: -28.457208618809304},

{’ADP’: -32.30402008654769,
’SCONJ’: -33.181466431765315,
’NOUN’: -26.167445777290492,
’ADJ’: -21.369394743473574,
’ADV’: -28.825998750365656,
’CCONJ’: -34.853653864864334,
’NUM’: -31.510372054048364,
’VERB’: -17.373733868537006,
’X’: -34.05991554298817,
’PROPN’: -30.138333676402,
’PUNCT’: -34.2441098307943,
’PRON’: -33.023671125165215,
’SYM’: -34.9556894177801,
’INTJ’: -33.03783235748981,
’DET’: -34.61292538110846,
’PART’: -35.016186398182256,
’AUX’: -29.337880787789842},

{’</s>’: -26.412560387861255}]
```

The table beta (after taking log) is:

```
[{’<s>’: -26.41256038786126},

{’ADP’: -23.836699821477527,
’SCONJ’: -22.926935275607402,
’NOUN’: -23.95407493714146,
’ADJ’: -23.924131802472083,
’ADV’: -21.923049125627497,
’CCONJ’: -22.41762587563202,
’NUM’: -24.300179605919574,
’VERB’: -24.371526243594726,
’X’: -21.937494566788523,
’PROPN’: -23.564604503851402,
’PUNCT’: -23.985442262082316,
’PRON’: -21.996726492308373,
’SYM’: -24.01815732000537,
’INTJ’: -21.632751913804185,
’DET’: -22.229347739469198,
’PART’: -20.712955062763616,
’AUX’: -21.562007084983605},

{’ADP’: -14.088374182794402,
’SCONJ’: -12.140190703191838,
’NOUN’: -5.739914995227308,
’ADJ’: -7.855242251454154,
’ADV’: -8.101060444651063,
’CCONJ’: -11.123905839490517,
’NUM’: -5.26559011073755,
’VERB’: -9.105777586565452,
’X’: -1.991468366378883,
’PROPN’: -4.225355710401363,
’PUNCT’: -1.6431816504519983,
’PRON’: -10.544964432789238,
’SYM’: -2.868308133289467,
’INTJ’: -5.292321632802039,
’DET’: -26.596762470000893,
’PART’: -11.434628227636725,
’AUX’: -12.5925738310379},

{’</s>’: 0}]
```

So actually the alpha[-1][end]=beta[0][start]. The more details can be seen in the python code.

4 Evaluation

In this section, I will evaluate the accuracy of each algorithm. I also download the dataset of Greek, Chinese, Japanese to evaluate the performance of algorithms on languages with different characteristics.

Especially, I used two evaluation metrics for algorithm 2 and 3. The sentence accuracy means the algorithm how many sentences algorithm exactly right predicted. The word accuracy means how many tag of word are predicted right.

e.g., for dataset [['Cat', 'drive' 'car']] (just have 1 sentence), true tag is ['NOUN', 'VERB', 'NOUN']. If the model predict as ['DET', 'VERB', 'NOUN'], the sentence accuracy = 0/1 = 0, the word accuracy = 2/3 = 66.67%.

4.1 Algorithm 1 – Eager Algorithm

Table 1: The accuracy of algorithm 1

language	English	Spanish	Dutch	Greek	Chinese	Japanese
sentence accuracy	36.16%	19.72%	16.11%	18.42%	6.40%	19.15%
word accuracy	87.31%	91.59%	87.24%	89.71%	85.11%	90.66%

4.2 Algorithm 2 – Viterbi Algorithm

Table 2: The accuracy of algorithm 2

language	English	Spanish	Dutch	Greek	Chinese	Japanese
sentence accuracy	43.28%	26.76%	22.82%	25.44%	7.20%	20.99%
word accuracy	89.58%	93.04%	88.96%	91.79%	86.12%	92.30%

4.3 Algorithm 3 – Baum-Welch Algorithm

Table 3: The accuracy of algorithm 3

language	English	Spanish	Dutch	Greek	Chinese	Japanese
sentence accuracy	43.24%	27.00%	22.48%	26.32%	6.60%	20.81%
word accuracy	89.78%	93.25%	89.28%	91.90%	85.96%	92.34%

5 Analysis

For the performance of algorithms, Baum-Welch Algorithm > Viterbi Algorithm > Eager Algorithm.

Overall, algorithms did a better job of marking Phonogram (English, Spanish, Dutch, Greek, Japanese) than hieroglyph (Chinese).

This may be due to the different characteristics of the languages. Taking Chinese (hieroglyph) and English (alphabetic writing system) as a comparison:

- a word in Chinese often has a variety of lexical forms, while a word in English has only one or a few fixed lexical forms.
  e.g., When to use possessive, In English, we said "Your phone", but in Chinese, we usually said "你手机" (you phone) rather than "你的手机" (your phone).
- English naturally has spaces to separate words, but different sentence breaks in Chinese may result in different words.
  e.g., if 佟大为妻子产下一女 -> ['佟大', '为', '妻子', '产下', '一', '女'], that means a man (named Tongda) gave birth to a girl for his wife;
  if 佟大为妻子产下一女 -> ['佟大为', '妻子', '产下', '一', '女'], that means a man's wife (the man named Tong Dawei) gave birth to a girl.
- The English root system can naturally separate words with different lexical properties, but for Chinese, even for native speakers, they still confuse about how to determine the lexical properties.

An interesting phenomenon is that for sentence accuracy of 3 algorithms, Spanish is the highest, Chinese is the lowest. The reason may be the lexicality of Spanish words mostly depends on the adjacent words, while the lexicality of Chinese words has to be placed in the middle participle of the whole sentence first, after which the lexicality can be determined[2]. I guess this is also the reason why the Viterbi algorithm can perform better than Baum-Welch algorithm in Chinese, as the Viterbi try to get the most likely situation for the whole sentence, but the Baum-Welch try to get the most likely situation for each word, so the Baum-Welch may get more details in terms of context of a word, but it may lose some information of the construction of the sentence.

Bibliography

[1] Steven Bird, Ewan Klein, and Edward Loper. Nltk::nltkpackage, 2022. `https://www.nltk.org/api/nltk.html`.

[2] Curt Hoffman, Ivy Lau, and David R Johnson. The linguistic relativity of person cognition: An english–chinese comparison. *Journal of personality and Social Psychology*, 51(6):1097, 1986.

[3] Mark Nederhof and Lei Fang. logsumexp, 2022. `https://studres.cs.st-andrews.ac.uk/CS5012/Practicals/P1/logsumexptrick.py`.

[4] Wikipedia. Viterbi algorithm - wikipedia, 2022. `https://en.wikipedia.org/wiki/Viterbi_algorithm`.