Part-Of-Speech Tagger based on Hidden Markov Models

Mingzhen Lin

Mitchtom School of Computer Science, Brandeis University
lim@brandeis.edu

Yulun Wu

Mitchtom School of Computer Science, Brandeis University
yulunwu@brandeis.edu

In this project, we implemented both supervised HMM and unsupervised HMM model for part-of-speech tagging task, based on the Viterbi algorithm and the forward and backward algorithm (the Baum-Welch algorithm). We trained out supervised HMM on the Brown Corpus, and the unsupervised HMM on a customized dataset, to demonstrate the algorithm itself.

Keywords: Natural Language Processing, NLP, Part-of-Speech tagging, POS tagging, Markov Chain, Hidden Markov Model, HMM

1. Introduction

In computational linguistics, a part of speech (Pos, PoS or POS) represents a category of words that share similar grammar property. Part-of-speech tagging is the process of labeling words for their POS tags in a sequence of words (text) in a corpus. This is important since the POS tag of a word contains information of not only the definition of the word but also the meaning of its surrounding words and the syntactic structure of the sentence. This is particularly useful in tasks like Parsing, Named Entity Extraction (NER), Coreference Resolution, Speech Recognition/Synthesis, etc.

POS tagging can be seen as a sequence labeling task, and the common algorithms used are Hidden Markov Models (HMM) and Conditional Random Field

(CRF). In our project, we implemented the Hidden Markov Models, both supervised and unsupervised version from scratch. The supervised HMM is trained on Brown Corpus, while the unsupervised version is trained on a small customized dataset.

2. Brown Corpus

The Brown University Standard Corpus of Present-Day American English (or just Brown Corpus) was compiled in the 1960s by Henry Kučera and W. Nelson Francis at Brown University, Providence, Rhode Island as a general corpus (text collection) in the field of corpus linguistics. It contains 500 samples of English-language text, totaling roughly one million words, compiled from works published in the United States in 1961.

We used the Brown Corpus via NLTK Corpus reader to avoid data pre-processing, which makes the project a lot easier.[2]

3. Hidden Markov Models

3.1. The Markov Chain

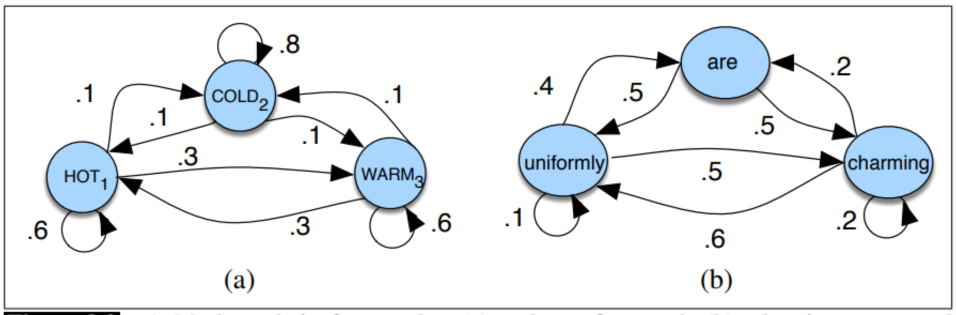


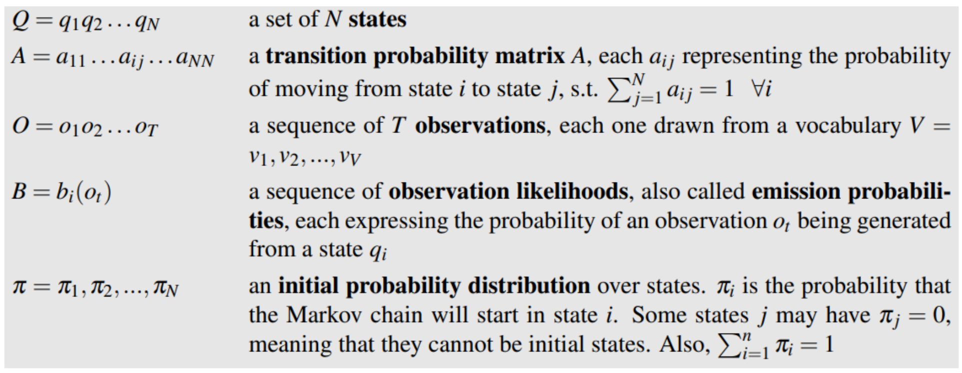
Figure 8.3 A Markov chain for weather (a) and one for words (b), showing states and transitions. A start distribution π is required; setting $\pi = [0.1, 0.7, 0.2]$ for (a) would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

Fig. 1. Diagram of two example Markov chains[1]

A Markov chain is a model that represents the transfer probabilities between some random states, each of which can take on values from some set. A Markov chain is base on a simple assumption, the **Markov assumption** on the probabilities of the sequence, that the next state only depends on the current state, the past ones do not count.

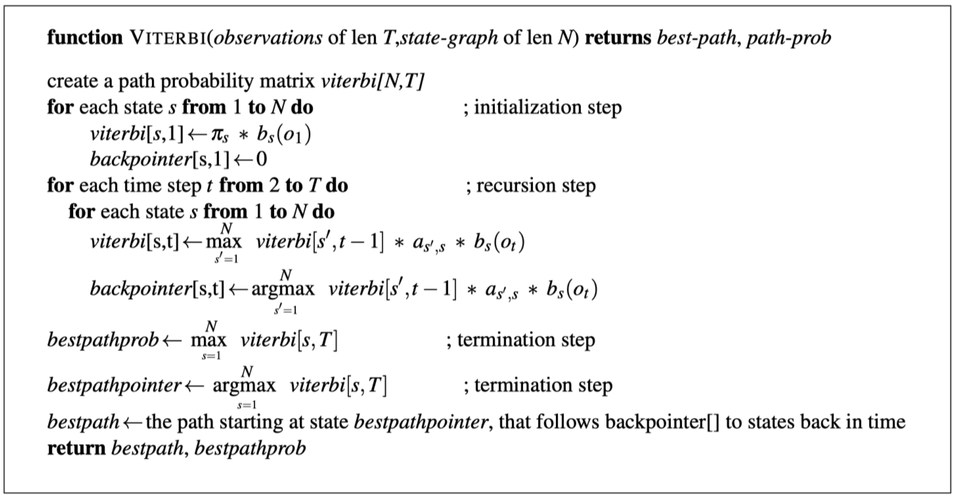
3.2. Supervised HMM

The Hidden Markov Model (or HMM) is a sequence model, which is used to assign a label to every single element in a sequence of elements. It is a probabilistic model based on the Markov chain that computes and choose the best possible one among all potential label sequences.



$Q = q_1 q_2 \dots q_N$	a set of N **states**
$A = a_{11} \dots a_{ij} \dots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \dots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \dots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state q_i
$\pi = \pi_1, \pi_2, \dots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

Fig. 2. The components of a Hidden Markov Model[1]



```
function VITERBI(observations of len T, state-graph of len N) returns best-path, path-prob

create a path probability matrix viterbi[N,T]
for each state s from 1 to N do                          ; initialization step
    $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
    $backpointer[s,1] \leftarrow 0$
for each time step t from 2 to T do                      ; recursion step
  for each state s from 1 to N do
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$          ; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$   ; termination step
bestpath ← the path starting at state bestpathpointer, that follows backpointer[] to states back in time
return bestpath, bestpathprob
```

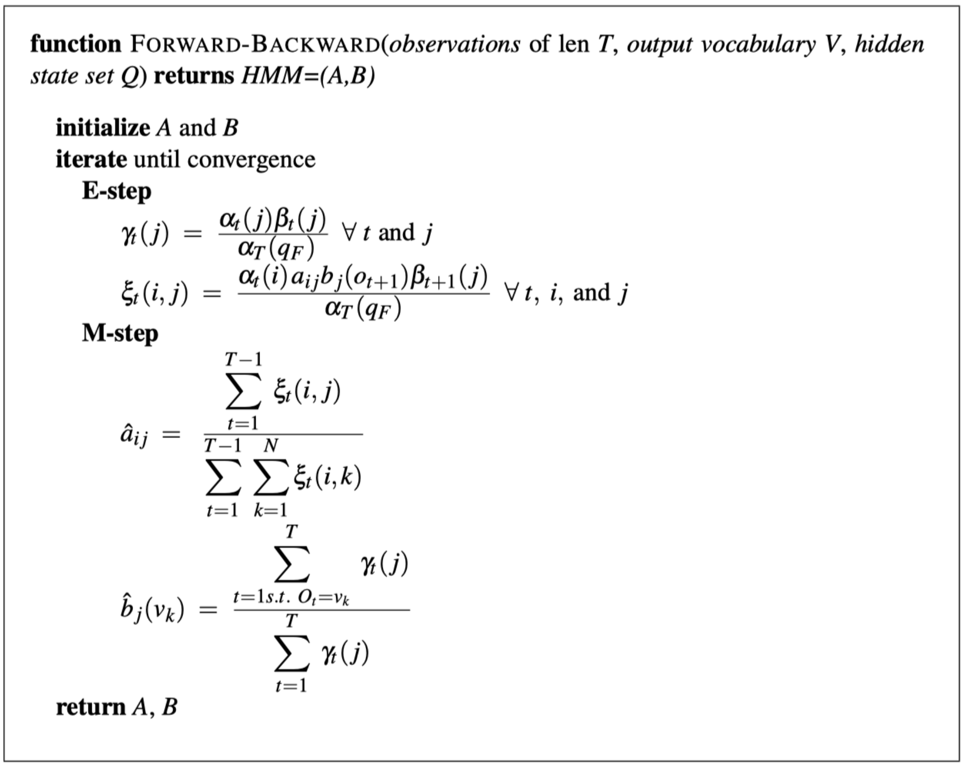
Fig. 3. Viterbi algorithm for finding the optimal sequence of tags[1]

A Hidden Markov Model is to solve the problem that, in a Markov chain we only compute probabilities of observable events; Sometimes, however, we care more about the hidden events. In the case of POS tagging, the words are observable and we need to infer the tags from the word sequence, which are hidden behind the text.

In a supervised HMM, we first train an HMM model using a given corpus with POS tags, then use the Viterbi algorithm to decode the most probable hidden sequence of tags based on a given observed sequence of words. Viterbi is a dynamic programming algorithm.

3.3. Unsupervised HMM Learning

For unsupervised datasets, we can not train the model directly. Instead, we use the forward and backward algorithm (also known as the Baum-Welch algorithm) to train the model. The input is an unlabeled sequence of observation O (which is the text) and a vocabulary of potential hidden states Q (the POS tags). After the model is trained, we can use the Viterbi algorithm to decode a target text.



```
function FORWARD-BACKWARD(observations of len $T$, output vocabulary $V$, hidden state set $Q$) returns $HMM=(A,B)$

  initialize $A$ and $B$
  iterate until convergence
    E-step
      $\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \;\forall\, t$ and $j$
      $\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \;\forall\, t,\ i,$ and $j$
    M-step
      $\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{k=1}^{N}\xi_t(i,k)}$
      $\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)}$
  return $A, B$
```

Fig. 4. The forward and backward algorithm[1]

4. the Structure of Project

In this section, we would introduce the structure of this project. It mainly includes two different pos taggers based on supervised HMM and unsupervised HMM separately.

4.1. `pos_tagger.py`

By using data from Brown Corpus, we split them as the training data set and the testing data set. Class `POSTagger` contains three matrices including `initial`, `transition` and `emission`. These three matrices served as basic for HHM.

- class `POSTagger`
  - `init_dict(self)`: Build maps from indices to words and POS.
  - `train(self, train_set)`: Given training data set, build three matrices including `initial`, `transition`, `emission`.
  - `viterbi(self, sentence)`: Given three matrices and a sentence, create a path probability matrix `viterbi` and find the most probable POS sequences as Fig.3. During the process of updating the viterbi matrix, back pointers can refer the routine to specific state.
  - `predict(self, tokens)`: Given tokens, attach POS to each token.
  - `evaluate(self, test_set)`: Given test data set, compute the accuracy.
- `train_POSTagger()`:
  - Create and train the POS tagger.
  - Considering that it is more efficient to save the `numpy` object, we decide to use `joblib` to save the model into the file `pos_tagger.jbl`.
  - After training, print the accuracy of this model.
- `run_POSTagger(input_text)`:
  - Load the POS tagger from `pos_tagger.jbl`.
  - Given the input text, the most probable POS sequences.

4.2. `forward_backward.py`

In order to implement the unsupervised POS tagger, we try to simply conditions. For easier implementation, assume that the length of sentence are all the same and use the same data for train and test. We can promote it to more general situation in the future.

- class `ForwardBackward`

- `build_data_set(self)`: Build maps from indices to words and POS.
- `forward(self, sentence)`: Through forward Algorithm, we can get the alpha array.
- `backward(self, sentence)`: Through backward Algorithm, we can get the beta array.
- `forward_backward(self, data_set)`: Like supervised POS tagger above. Given data set, build three matrices including `initial`, `transition`, `emission` as the steps in Fig.4
- `viterbi(self, sentence)`: The same algorithm as supervised POS tagger.
- `run(self, data_set)`: Given the data set, generate a sequence of word/POS.

5. Usage

5.1. Supervised HHM POS tagger

For training supervised HHM POS tagger, we can type in command like this.

```
$ python3 pos_tagger.py --train
```

And the output information would include the elapse time the acurracy.

```
Creating HHM POS tagger in ./pos_tagger.jbl
        Elapsed time: 5.205424785614014s
        Accuracy: 0.8967249002017427
```

For Running supervised HHM POS tagger, we can type in command like this.

```
$ python3 pos_tagger.py --run "Lets all be unique together until we
realise we are all the same."
```

Given the input text, we can get the pairs of word and pos.

```
* input text: Lets all be unique together until we realise we are all the
same.
* pairs of word and pos:
        Lets -- IN
        all -- ABN
        be -- BE
        unique -- JJ
        together -- RB
        until -- CS
        we -- PPSS
        realise -- MD
        we -- PPSS
        are -- BER
        all -- ABN
        the -- AT
        same -- AP
        . -- .
```

5.2. Unsupervised HHM POS tagger

For running unsupervised HHM POS tagger, we can type in command like this.

```
$python3 forward_backward.py
```

It would take `dataset` file as training data.

```
* Dataset:
        he saw a cat
        a cat saw him
        he chased the cat
        the cat chased him
        he saw the dog
        the dog saw him
        he chased a dog
        a dog chased him
```

The result is,

```
* Result:
        he/E saw/B a/D cat/A
        a/D cat/A saw/C him/C
        he/E chased/B the/D cat/A
        the/D cat/A chased/C him/C
        he/E saw/B the/D dog/A
        the/D dog/A saw/C him/C
        he/E chased/B a/D dog/A
        a/D dog/A chased/C him/C
```

6. Future Plan

- Apply the unsupervised POS tagger to more general situation.
- Build the POS tagger based on CRF theory.

References

1. Dan Jurafsky and James H. Martin, editors. *Speech and Language Processing (3rd)*. 2019.
2. Ewan Klein Steven Bird and Edward Loper, editors. *Natural Language Processing with Python*. 2019.