Natural Language Processing - Project 2 (HMM)
CSE 325/425
Prof. Sihong Xie
Due Dates: Apr 14, 2021, due at 11:55 pm.

1 Problem Statement

Given a POS-tagged (labeled/training) corpus and un-tagged (unlabeled/test) corpus, train an HMM model to predict POS tags on the test corpus.

1.1 Three problems and their solutions in HMM

- Likelihood calculation: given a sentence $\vec{O} = [o_1, \ldots, o_T]$ and the HMM parameter $\lambda = \{A, B, \pi\}$, find $p(\vec{O}|\lambda)$. This problem can be solved using the forward algorithm. See the slides and recording of the HMM lectures and Appendix A.3 of SLP3 for more details.
- Decoding: given a sentence $\vec{O}$ and the HMM parameter $\lambda = \{A, B, \pi\}$, find $\vec{Q}^*$, the sequence of POS tags that maximizes $p(\vec{Q}|\vec{O}, \lambda)$. This problem is solved using the Viterbi algorithm. See the slides and recording of the HMM lectures and Appendix A.4 of SLP3 for more details.
- Model training: given an **unlabeled** corpus of D sentences $\mathcal{D} = \{\vec{O}_1, \ldots, \vec{O}_D\}$, find the HMM parameter $\lambda = \{A, B, \pi\}$ that maximizes $p(\mathcal{D}|\lambda)$. This problem is solved using the Baum-Welch algorithm, an example of Expectation-Maximization (EM) algorithm. See the slides and recording of the HMM lectures and Appendix A.5 of SLP3 for more details.

1.2 Numerical issues in the forward, backward, and Viterbi algorithms

During the Viterbi algorithm, we are taking products of many probabilities, leading to infinitesimal numbers that underflow and our computers do not have sufficient precision to represent such numbers. Note that only the relative magnitudes of $v_t(j)$ matter when maximizing and we can work in the log scale:

$$\log v_t(j) = \max_i \left[\log v_{t-1}(i) + \log a_{ij} + \log b_{jo_t}\right] \tag{1}$$

The maximizing index i^* will be the same whether log is taken or not so that the backtracking pointers are not affected. The reconstructed prediction $\vec{Q}^*$ will be the same.

During the forward algorithm, the same issue arises but is addressed in a different way. For each location t,

$$\begin{aligned} \tilde{\alpha}_t(j) &= \begin{cases} \sum_i \hat{\alpha}_{t-1}(i) a_{ij} b_{jo_t} & \text{if } t > 1 \\ \pi_j b_{jo_t} & \text{if } t = 1 \end{cases} \\ c_t &= \frac{1}{\sum_j \tilde{\alpha}_t(j)} \quad \text{(normalizing factor)} \\ \hat{\alpha}_t(j) &= c_t \times \tilde{\alpha}_t(j) \quad \text{(normalization)} \end{aligned} \tag{2}$$

Then $\hat{\alpha}$ is normalized to a good range and $\sum_j \hat{\alpha}_t(j) = 1$.

During the backward algorithm, similar local normalization of $\tilde{beta}_t(j)$ to obtain $\hat{\beta}_t(j)$

It is important to follow the supplementary materials "A Revealing Introduction to Hidden Markov Models" by Mark Stamp to implement these algorithms.

1.3 Working with corpora

The Appendix and the supplementary materials only show you how to run the above algorithms on a single sentence, while we are dealing with corpora of multiple sentences. This difference will affect the EM algorithm: simply storing all $\xi_t(i,j)$ and $\xi_t(i)$ and other variables for each sentence is not scalable. Instead, you will need to design online algorithm that only take memory linear in the sentence length but not the corpus size. You can estimate the ξ's for the sentence $\vec{O}_d$, update the sums in the nominator and denominator in

$$\hat{a}_{ij}^{(d)} = \sum_{k=1}^{d} \sum_{t=1}^{T-1} \xi_t^k(i,j) \tag{3}$$

and go to the next sentence. At the end when all D sentences in the corpus are proccessed, the sums are accumulated over all positions of the whole corpus (all positions = all words in all sentences).

$$\hat{a}_{ij}^{(D)} = \sum_{d=1}^{D} \sum_{t=1}^{T-1} \xi_t^d(i,j) \tag{4}$$

and finally normalize $\hat{a}_{ij}^{(D)}$ to the transition matrix $A_{\mathcal{D}}$ by

$$(A_{\mathcal{D}})_{ij}^{(D)} = \frac{\sum_{d=1}^{D} \sum_{t=1}^{T-1} \xi_t^d(i,j)}{\sum_{d=1}^{D} \sum_{t=1}^{T-1} \sum_j \xi_t^d(i,j)}. \tag{5}$$

B and π can be done in a similar way.

Follow the supplementary materials, but only normalize the A, B, and π matrices after you go through the entire test corpora. $\xi_t(i,j)$ is $\gamma_t(i,j)$ and $\xi_t(i)$ is $\gamma_t(i)$ in the supplementary materials.

1.4 Supervised, unsupervised and Semi-supervised HMM

The algorithm in Appendix A.5 of SLP3 is only for the unsupervised learning of HMM: finding λ using $\mathcal{D}$ only. The problem here is that with zero prior knowledge, the learned λ can be inaccurate and hard to interpret.

On the other hand, if a labeled corpus $\mathcal{L}$ is given, then a simple way to obtain λ is maximum likelihood estimation (MLE):

$$a_{ij} = \frac{\#(q_t = i, q_{t+1} = j)}{\#(q_t = i)}, \quad b_{io} = \frac{\#(q_t = i, o_t = o)}{\#(q_t = i)}, \quad \pi_i = \frac{\#(q_1 = i)}{\#\text{ of sentences}}. \tag{6}$$

The strength of this approach lies in the prior linguistic information from $\mathcal{L}$: the human knowledge regarding the language is encoded in $\mathcal{L}$ and MLE just extracts that. However, if the $\mathcal{L}$ is small, not all linguistic knowledge is covered.

The third way is the semi-supervised approach that combines both the labeled and unlabeled corpora for decoding and learning. The obvious way is to use the supervised MLE to find an initial guess of λ, then run the Baum-Welch algorithm on $\mathcal{D}$ to re-estimate λ. The potential strength of this approach is that it has access to more data than the above two approaches. However, the knowledge from $\mathcal{L}$ and $\mathcal{D}$ can be different in quality and quantity, and which corpus to trust more remains a question (phase II asks you to explore this question).

A new parameter $\mu \in [0,1]$ (Mu) is added to balance the estimations on $\mathcal{D}$ and $\mathcal{L}$:

$$A = \mu A_{\mathcal{L}} + (1-\mu) A_{\mathcal{D}}, \quad B = \mu B_{\mathcal{L}} + (1-\mu) B_{\mathcal{D}}, \quad \pi = \mu \pi_{\mathcal{L}} + (1-\mu) \pi_{\mathcal{D}} \tag{7}$$

When $\mu = 1$, the parameters are estimated using $\mathcal{L}$ only (equivalent to Eq. (6)); when $\mu = 0$, $\mathcal{L}$ is totally abandoned (unsupervised HMM). You can set μ to be between 0 and 1 and use these equations to replace those in Eq. (6) and in the unsupervised HMM.

2 Experiments

The list of Python files in the project:

- **problem1.py** It is provided to you to read the tagged sentences from *data/train.txt* and *data/test.txt*.
- **problem2.py** This file defines the HMM class and you need to implement forward, backward, Viterbi, and MLE methods.
- **problem3.py** This file defines the EM algorithm. It also trains the HMM model.

3 Deliverables

Implement the functions that has the section saying "INSERT YOUR CODE HERE". After you implemented and tested your codes, zip the folder "project_2" to "project_2_<YOUR_LIN>.zip". Submit the zip file to Coursesite under "Proejct 2".

Important: Before you submit your project, type the following into the command line on Linux/Unix/MacOS (on Windows, see `https://ubuntu.com/wsl` or use your Linux simulators):

```
nosetests -v
```

If you see the following

```
(10 points) problem1:read_corpus ... ok
(15 points) problem2: mle ... ok
(15 points) problem2: forward ... ok
(15 points) problem2: backward ... ok
(15 points) problem2: Viterbi ... ok
(15 points) problem3: em_one_sentence ... ok
(15 points) problem3: maximization ... ok

----------------------------------------------------------------------
Ran 7 tests in 1.813s

OK
```

your program passes the unit tests, though it does not mean your codes are correct. However, failing to pass these tests means that your programs have some errors. These tests contain data for you to debug your program.

Also train your HMM by typing

```
python problem3.py
```

and you should see something similar to the following:

```
-406294.8663008706
0.8905798172108829 -342901.56895487744
-339133.4729949529
0.8853452097009097 -348139.3422785319
-345428.13561662217
0.8776410494543766 -347311.3584517821
-344779.43067372055
0.8645756379677902 -344442.5213245596
-341923.76325417537
0.8493994976465373 -341371.99033356516
-338744.53009798285
0.8347299322456044 -338916.04988297506
-335764.4121754806
0.7932118960677121 -335829.7900962944
-332314.32038080745
0.7521160056567533 -332976.19735483313
-328959.02498358535
0.7201173565232074 -330246.52432217
-325329.7182050161
0.6691221478776621 -327455.2071698861
```

Important: remember to click "Submit" botton to deliver your submission, otherwise the project will be regarded as NOT submitted.

Important: Don't submit the datasets and intermediates results! They're too large and slow down downloading.

4 Grading

The points for each function is printed when you run the unit tests using nosetests. The total is 100 points.