Natural Language Processing - Project 2
CSE 398/498-013
Prof. Sihong Xie
Due Dates: Oct 10, 2018 (Phase I), Oct 22, 2018 (Phase II), Nov 21, 2018 (Final due). All 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 Chap 6.3 of SLP2 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 Chap 6.4 of SLP2 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 Chap 6.5 of SLP2 for more details.

1.2 Supervised, unsupervised and Semi-supervised HMM

The algorithm in Chap 6.5 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):

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

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 $\mathcal{L}$:

$$A = \mu A_{\mathcal{L}} + (1 - \mu) A_{\mathcal{D}}, \quad B = \mu B_{\mathcal{L}} + (1 - \mu) B_{\mathcal{D}}, \quad \pi = \mu \pi_{\mathcal{L}} + (1 - \mu) \pi_{\mathcal{D}} \tag{2}$$

When $\mu = 1$, the parameters are estimated using $\mathcal{L}$ only (equivalent to Eq. (1)); 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. (1) and in the unsupervised HMM.

1.3 Numerical issues

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:

$$\log v_t(j) = \max_i \left[\log v_{t-1}(i) + \log a_{ij} + \log b_{jo_t}\right] \tag{3}$$

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,

$$\begin{aligned} \tilde{\alpha}_t(j) &= \begin{cases} \sum_i \hat{\alpha}_{t-1}(i) a_{ij} b_{jo_t} & \text{if } t > 1 \\ \pi_j b_{jo_t} & \text{if } t = 1 \end{cases} \\ c_t &= \frac{1}{\sum_j \tilde{\alpha}_t(j)} \quad \text{(normalizing factor)} \\ \hat{\alpha}_t(j) &= c_t \times \tilde{\alpha}_t(j) \quad \text{(normalization)} \end{aligned} \tag{4}$$

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

1.4 Working with corpora

Our textbook only shows 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

$$\hat{a}_{ij} = \frac{\sum_{k=1}^{d} \sum_{t=1}^{T-1} \xi_t^k(i,j)}{\sum_{k=1}^{d} \sum_{t=1}^{T-1} \sum_j \xi_t^k(i,j)}, \tag{5}$$

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

$$\hat{a}_{ij} = \frac{\sum_{d=1}^{D} \sum_{t=1}^{T-1} \xi_t^d(i,j)}{\sum_{d=1}^{D} \sum_{t=1}^{T-1} \sum_j \xi_t^d(i,j)}, \tag{6}$$

B and π can be done in a similar way.

2 Exercises

Answer the following questions in your report:

(1) Prove that $\hat{\alpha}_t(j) = \prod_{s=1}^{t} c_s \alpha_t(j)$ and $\sum_j \alpha_T(j) = \frac{1}{\prod_{s=1}^{T} c_s}$ (hint: use proof of induction).

(2) Assume $\beta_T(j) = c_T$ for all j, then prove that $\hat{\beta}_t(j) = \prod_{s=t}^{T} c_s \beta_t(j)$.

(3) Prove that $\xi_t(i,j) = \hat{\alpha}_t(i) \hat{\beta}_{t+1}(j) a_{ij} b_{jo_{t+1}}$. In other words, you can use $\hat{\alpha}$ and $\hat{\beta}$ to find ξ directly without calculating α, β and $P(\vec{O}|\lambda)$ (!).

(4) Prove that the A, B, π matrices produced by Eq. (2) define appropriate probability distributions.

3 Experiments

3.1 System design and implementations

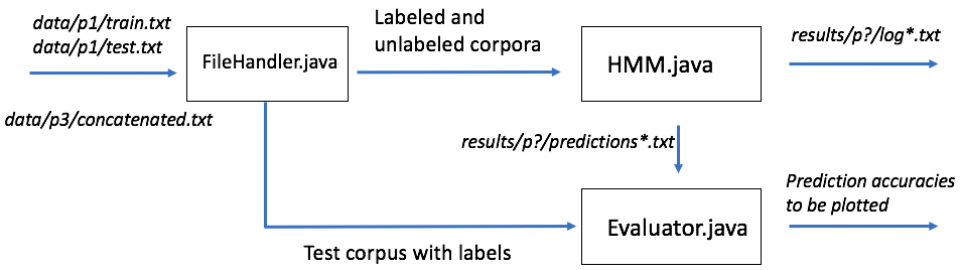

Figure 1: Overall system design. *p1* in the folder names indicate that the files are for phase I. For phase II/III, you will need to change them to *p2* or *p3*. I used wildcards ? and * to denote filenames that match the pattern (see the texts for the exact filenames).

Figure 1 shows the design. Similar to project 1, boxes indicate java classes, arrows indicate flow of data, with names of data over arrows being the input/output (*Italic* names are files on disk, while normal names are Java objects within your program). See the attached bash script and the functions/APIs to understand the flow. You may create other helper files, but I will enforce the formats of those files shown in Figure 1.

The list of Java classes in the project:

- **FileHandler.java** (provided) It is provided to you to read and write tagged and untagged sentences. *data/p1/train.txt, data/p1/test.txt* and *results/p1/test_predictions.txt* will be handled by this class. Note that the prediction files are not that different from the train.txt and test.txt files – the only two differences are: 1) the POS tags in the predictions files are generated by your HMM model instead of given, and 2) there are only two columns (unigram tokens and POS tags) in the prediction files. This class will be the same for phases I and II except that files will be under different folders (*p1 vs. p2*).
- **Evaluator.java** (provided) This class calculates performance metrics using your predictions (written to a file) and ground truth POS tags (provided in the *data/p1/test.txt* file). This class will be provided to you.
- **Word.java** (provided) This class encapsulates word information including the token, POS tag (in a particular position, if any) and other features of a word. Public methods provide ways to access and modify word information.
- **Sentence.java** (provided) The class represents a single sentence, which is a list of Word objects. Note that a corpus can be represented by an array of Sentence objects. Public methods provide ways to access and modify a sentence.
- **HMM.java** (to be implemented)

Phase I : Implement the forward and Viterbi algorithms. Use FileHandler to read $\mathcal{D}$ (*data/p1/test.txt*, 8,936 sentences) and $\mathcal{L}$ (*data/p1/train.txt*, 2,012 sentences). Estimate $\lambda = \{A, B, \pi\}$ using MLE on $\mathcal{L}$. Then run forward and Viterbi algorithms using λ on $\mathcal{D}$. Matrix class Jama is recommended to represent all matrices and improve the readability

of your codes. Numerical issues need to be taken care of as indicated in the problem statement. Output predictions of POS tags for sentences in $\mathcal{D}$ to *results/p1/predictions.txt* and log likelihood ($\sum_{d=1}^{D} \log p(\vec{O}_d|\lambda^{(t)})$) to *results/p1/log.txt*.

Phase II : Implement the backward algorithm, and combine that with forward algorithm in Phase I to build the EM algorithms. Initialize $\lambda = \{A, B, \pi\}$ using MLE on $\mathcal{L}$. Then infer the probabilities $p(q_t = i, q_{t+1} = j, \vec{O}_d)$ and $p(q_t = i, \vec{O}_d)$ for each sentence $\vec{O}_d$ in $\mathcal{D}$ (E-step). Re-estimate λ using Eq. (6.38) and (6.43) from the textbook (M-step). Then repeat these EM steps for 30 iterations. Use the final model to predict the POS tags for sentences in $\mathcal{D}$ and output the predictions to *results/p2/predictions.txt*. The log likelihoods ($\sum_{d=1}^{D} \log p(\vec{O}_d|\lambda^{(k)})$) after each iteration k of the EM algorithm should be output to *results/p2/log.txt*.

Phase III There are two tasks. 1) Semi-supervised HMM: set μ to values in $\{0, 0.1, 0.2, \ldots, 1\}$ to allow different balances between $\mathcal{D}$ and $\mathcal{L}$. For each μ value, predict the POS-tags for sentences in $\mathcal{D}$ using the final model (output to *results/p3/predictions.txt*), and track the log-likelihoods $P(\mathcal{D}|\lambda)$ during all 30 EM iterations (output to *results/p3/log.txt*). Use the Evaluator class to report prediction accuracies of different models. 2) Scalable HMM: Re-run task 1) on a larger corpus *data/p3/concatenated.txt*. Output the results to *data/p3/predictions_concatenated.txt* and *results/p3/log_concatenated.txt*). Note that the log-likelihood shall only be computed on $\mathcal{D}$ but not the concatenated corpus (namely, sum over sentences from $\mathcal{D}$ only). Since it can be slow on the larger corpus, you can try only a few μ values instead of all 11.

Bash script files (build_java.sh and run_java.sh) are provided to compile and run the programs. The necessary third party packages are in the folder "jars" in the zipped file (jama.jar in this project).

4 Deliverables

Phase I Download the provided zip file and put down your codes in the empty functions in HMM.java. Implement the forward and Viterbi algorithms, output the POS predictions and log-likelihood to the folder *results/p1/* as required above.

Add README.txt the root folder (one level above src). to describe what works and what does not, any improvements you think that should earn you extra credits. Then zip the whole folder to <your Lehigh id>_p2.zip and upload it to coursesite. Don't submit the project from your IDE.

Add report_p2.pdf file contains the answers to the exercise questions (1).

Phase II Implement the backward and EM algorithms, and output the POS predictions and log-likelihood to the folder *results/p2/* as required above.

Update README.txt to reflect new progress in Phase II.

Update report_p2.pdf to include answers to questions (2) and (3). Plot the change of log-likelihood as EM progresses (see Figure 2, but you will have one curve only). Add the plot to the report PDF.

Phase III Implement the semi-supervised HMM and test it on both small and large test data (there shall be 4 files output to *results/p3/*).

Update README.txt to reflect new progress in Phase II.

Update report_p2.pdf to include answers to questions (4). Plot the log-likelihoods during EM (there should be 11 curves on *p1/data/test.txt* for 11 different μ values). An example figure using $\mu = 0.7, 0.8, 0.9$ on unlabeled corpus *p1/data/test.txt* is given in Figure 2. Also include a figure to report the accuracies of the trained semi-supervised HMM with different μ values (see Figure 3. Notice that as μ goes up, the accuracy gets better).

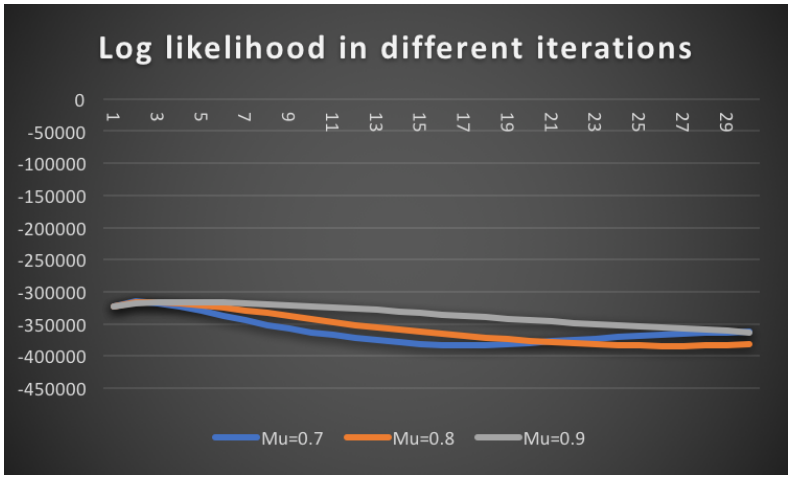

Figure 2: x-axis is the number of iterations and the y-axis is the log likelihoods $P(\mathcal{D}|\lambda)$.

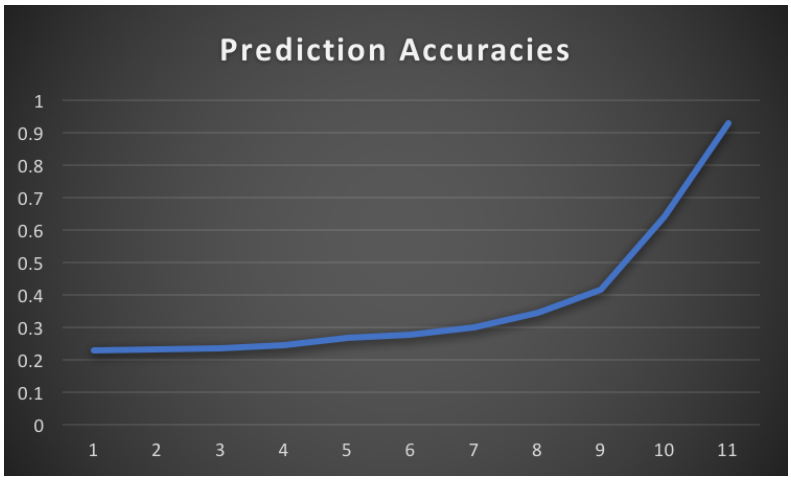

Figure 3: x-axis is different *mu* values and the y-axis is the POS-tagging accuracies.

Important: Before you submit your project, compile and run it (using the provided bash scripts) on a Sunlab machine where your projects will be graded.

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

5 Grading

The total grade is 20 points for graduates and 30 points for undergraduates. When grading your project, we will read your codes, and run your programs on a Sunlab machine. HMM decoding accuracy, running time and memory consumption will be used to determine one third of your total grade of this project.