**Natural Language Processing - Project 2**
CSE 398/498-013
Prof. Sihong Xie
Due Date: 11:59pm, Oct 12, 2017 (Phase I), 11:59pm, Oct 24, 2017 (Phase II)

# 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 $\vec{O}$ 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 a $\vec{Q}^*$, a sequence of POS tags, to maximize $p(\vec{Q}|\vec{O}, \lambda)$. 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 $\lambda$ using $\mathcal{D}$ only. The problem here is that with zero prior knowledge, the learned $\lambda$ can be inaccurate and hard to interpret.

On the other hand, a labeled corpus $\mathcal{L}$ is given, then a simple way to obtain $\lambda$ 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 combine 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 $\lambda$, then run the Baum-Welch algorithm on $\mathcal{D}$ to re-estimate $\lambda$. 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).

## 1.3 Numerical issues

During the Viterbi algorithm, we are taking products of many probabilities, leading to infinitesimal numbers that underflows and our computers do not have sufficient precision to represent

the numbers. Note that only the relative quantities of $v_t(j)$ matter when maximizing and we can work in the log scale:

$$\log v_t(j) = \max_i \left[\log v_{t-1}(i) + \log a_{ij} + \log b_{jo_t}\right] \tag{2}$$

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 $t$,

$$\begin{aligned} \tilde{\alpha}_t(j) &= \begin{cases} \sum_i \hat{\alpha}_{t-1}(i) a_{ij} b_{jo_t} & \text{if } t > 1 \\ \pi_j b_{jo_t} & \text{if } t = 1 \end{cases} \\ c_t &= \frac{1}{\sum_j \tilde{\alpha}_t(j)} \quad \text{(normalizing factor)} \\ \hat{\alpha}_t(j) &= c_t \times \tilde{\alpha}_t(j) \quad \text{(normalization)} \end{aligned} \tag{3}$$

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 $\xi$'s for the sentence $\vec{O}_d$, update the sums in the nominator and denominator in

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

and go to the next sentence. At end, the sums are accumulated over all position of the whole corpus (imagine that the corpus as a very long sentence that is the concatenation of invididual sentences in the corpus).

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

$B$ and $\pi$ can be done in a similar way.

## 2 Exercises

Answer the following questions in your report:

- Prove that $\hat{\alpha}_t(j) = \prod_{s=1}^{t} c_s \alpha_t(j)$ and $\sum_j \alpha_T(j) = \prod_{s=1}^{T} c_s$ (hint: use proof of induction).
- Assume $\beta_T(j) = c_T$ for all $j$, then derive similar formula as above to prevent underflowing for all $\beta_t(j)$.

# 3 Experiments

## 3.1 Data exploration

In phase I, download the small POS-tagged training and test corpora from `https://www.clips.uantwerpen.be/conll2000/chunking/`. Refer to the website regarding the data format. In phase II, larger corpora will be released to test the scalability of your programs. For both cases, report the number of POS-tags, tokens (unigrams), sentences, maximal length of sentences.

## 3.2 System design and implementations

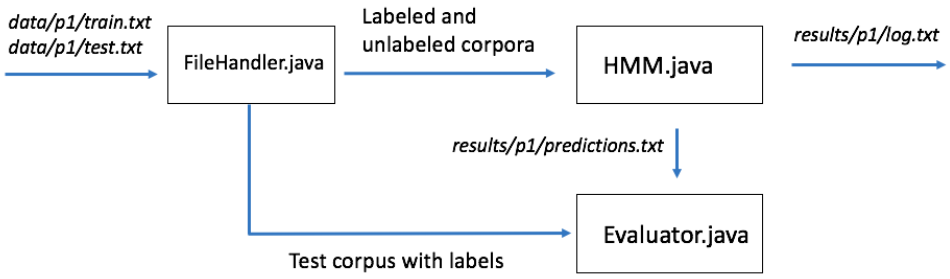


Figure 1: Overall system design. *p1* in the folder names indicate that the files are for phase I. For phase II, you will need to change them to *p2*.

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** 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*).

- **HMM.java** In phase I, implement the forward, backward and the EM (Baum-Welch) algorithms. Use FileHandler to read $\mathcal{D}$ and $\mathcal{L}$. Initialize $\lambda = \{A, B, \pi\}$ using MLE on $\mathcal{L}$. Then run the EM on $\mathcal{D}$ to re-estimate $\lambda$ using the forward-backward algorithm. 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. Produce and output predictions of POS tags for sentences in $\mathcal{D}$ to *results/p1/predictions.txt*, and log likelihoods ($\sum_{d=1}^{D} p(\vec{O}_d|\lambda^{(t)})$) after each iteration $t$ of the EM algorithm) to *results/p1/log.txt*,

  In phase II, you will be asked to re-estimate $\lambda$ using both $\mathcal{L}$ and $\mathcal{D}$, details TBD (to be determined).

- **Evaluator.java** This class calculates performance metrics using your predictions and ground truth POS tags. This class will be provided to you.
- **Word.java** 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** 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.

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

Download the provided zip file and put down your codes in the empty functions. Add README.txt and report_p2.pdf to the root folder (one level above src). Then zip the whole folder to <your Lehigh id>_p2.zip and upload it to coursesite. Don't submit the project from your IDE.

The README.txt describes what works and what does not, any improvements you think that should earn you extra credits. The report_p2.pdf file contains the answers to the exercise questions, a plot of log-likelihood during EM and an analysis of the results. More details TBD. 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

When grading your project, I will run the bash scripts to compile and run your codes on a Sunlab machine. HMM decoding accuracy, running time and memory consumption will be used to rank your project against others' and determine one third of your total grade (15 pts) of this project.

**Important:** you can submit a milestone before the phase I due date to earn an extra credit of 2 points on top of the 15 points of the entire project. The milestone submission format is the same as the final submission in phase II but without the report and README.txt. It shall at least contain an HMM model that can be trained and output predictions. I will just check if it works and don't care how accurate it is at that time. You can keep improving it before phase II due date.