Probabilistic Graphical Models

Part B HWK#1

Assigned Thursday, Jan. 19, 2023

Due (both Parts A and B) : Friday, Feb. 10, 2023

Problem 8

Hidden Markov Models In this problem, you will build a trigram hidden Markov model (HMM) to identify gene names in biological text. Under this model the joint probability of a sentence $x_1, x_2, \cdots, x_n$ and a tag sequence $y_1, y_2, \cdots, y_n$ is defined as

$$\begin{gathered} p(x_1, \cdots, x_n, y_1, \cdots, y_n) = \\ q(y_1|*, *) \cdot q(y_2|*, y_1) \cdot \prod_{i=3}^{n} q(y_i|y_{i-2}, y_{i-1}) q(STOP|y_{n-1}, y_n) \cdot \prod_{i=1}^{n} e(x_i|y_i) \end{gathered} \tag{1}$$

where $*$ is a padding symbol that indicates the beginning of a sentence and $STOP$ is a special HMM state indicating the end of a sentence. Your task will be to implement this probabilistic model and a decoder for finding the most likely tag sequence for new sentences.

Data The files for the assignment are located in the archive hmm.zip. We provide a labeled training data set gene.train, a labeled and unlabeled version of the test set, gene.key and gene.test. The labeled files take the format of one word per line with word and tag separated by space and a single blank line separates sentences, e.g.

```
Comparison O
with O
alkaline I-GENE
phosphatases I-GENE
and O
5 I-GENE
- I-GENE
nucleotidase I-GENE

Pharmacologic O
aspects O
of O
neonatal O
hyperbilirubinemia O
.O
```

$\vdots$

The unlabeled file contains only the words of each sentence and will be used to evaluate the performance of your model.

The task consists of identifying gene names within biological text. In this dataset there is one type of entity: gene (GENE). The dataset is adapted from the BioCreAtIvE II shared task.

To help out with the assignment we have provided several utility scripts. Our code is written in Python, but **you are free to use any language for your own implementation**. Our scripts can be called at the command-line to pre-process the data and to check results.

Collecting Counts The script *count_freqs.py* handles aggregating counts over the data. It takes a training file as input and produces trigram, bigram and emission counts. To see its behavior, run the script on the training data and pipe the output into a file

```
python count_freqs.py gene.train > gene.counts
```

Each line in the output contains the count for one event. There are two types of counts:

- Lines where the second token is WORDTAG contain emission counts $Count(y \rightsquigarrow x)$, for example

  ```
  13 WORDTAG I-GENE consensus
  ```

  indicates that consensus was tagged 13 times as I-GENE in the the training data.
- Lines where the second token is n-GRAM (where n is 1, 2 or 3) contain unigram counts $Count(y)$, bigram counts $Count(y_{n-1}, y_n)$, or trigram counts $Count(y_{n-2}, y_{n-1}, y_n)$. For example

  ```
  16624 2-GRAM I-GENE O
  ```

  indicates that there were 16624 chunks of an O tag following an I-GENE tag and

  ```
  9622 3-GRAM I-GENE I-GENE O
  ```

  indicates that in 9622 cases the bigram I-GENE I-GENE was followed by an O tag.

Evaluation The script *eval_gene_tagger.py* provides a way to check the output of a tagger. It takes the correct result and a user result as input and gives a detailed description of accuracy:

```
python eval_gene_tagger.py gene.key gene_test.p1.out
```

A sample output looks like:

```
Found 2669 GENEs. Expected 642 GENEs; Correct: 424.
           precision   recall     F1-Score
  GENE:    0.158861    0.660436   0.256116
```

Results for gene identification are given in terms of precision, recall, and F1-Score. Let $\mathcal{A}$ be the set of chunks that our tagger marked as GENE, and $\mathcal{B}$ be the set of chunks that are correctly GENE entities. Precision is defined as $|\mathcal{A} \cap \mathcal{B}|/|\mathcal{A}|$ whereas recall is defined as $|\mathcal{A} \cap \mathcal{B}|/|\mathcal{B}|$. F1-score represents the harmonic mean of these two values.

8.1 Baseline

1. We need to predict emission probabilities for words in the test data that do not occur in the training data. One simple approach is to map infrequent words in the training data to a common class and to treat unseen words as members of this class. Replace infrequent words ($Count(x) < 5$) in the original training data file with a common symbol _RARE_ . Then re-run *count_freqs.py* to produce new counts.

2. Using the counts produced by *count_freqs.py*, write a function that computes emission parameters

$$e(x|y) = \frac{Count(y \rightsquigarrow x)}{Count(y)} \tag{2}$$

3. As a baseline, implement a simple gene tagger that always produces the tag $y^* = \underset{y}{\operatorname{argmax}}\, e(x|y)$ for each word x. Make sure your tagger uses the _RARE_ word probabilities for rare and unseen words. Your tagger should read in the counts file and the file *gene.test* (which is *gene.key* without the tags) and produce output in the same format as the training file. For instance

   ```
   Nations I-ORG
   ```

Write your output to a file called *gene_test.p1.out* and evaluate by running

python eval_gene_tagger.py gene.key gene_test.p1.out

The expected result should be close to the result above. Report your result.

8.2 HMM with trigram features

1. Using the counts produced by *count_freqs.py*, write a function that computes parameters

$$q(y_i|y_{i-2}, y_{i-1}) = \frac{Count(y_i, y_{i-2}, y_{i-1})}{Count(y_{i-2}, y_{i-1})} \tag{3}$$

   for a given trigram y_{i-2} y_{i-1} y_i. Make sure your function works for the boundary cases $q(y_1|*, *)$, $q(y_2|*, y_1)$ and $q(STOP|y_{n-1}, y_n)$.

2. Using the maximum likelihood estimates for transitions and emissions, implement the Viterbi algorithm to compute

$$\underset{y_1,\cdots,y_n}{\operatorname{argmax}}\, p(x_1, \cdots, x_n, y_1, \cdots, y_n) \tag{4}$$

   Be sure to replace infrequent words ($Count(x) < 5$) in the original training data file and in the decoding algorithm with a common symbol _RARE_. Your tagger should have the same basic functionality as the baseline tagger.

   Run the Viterbi tagger on the test set and write your output to a file called *gene_test.p2.out*. Evaluate your model by using *eval_gene_tagger.py* and report your result.

Submission Procedure In addition to reporting the evaluation results in your report, you also need to submit the following files in canvas in order to get full credit:

- Code. Note that you can use any programming language.
- Instructions on how to run your code.
- The result files produced by your implementation: *gene_test.p1.out*, *gene_test.p2.out*, *gene_test.p3.out*, and *gene_test.p4.out*

Problem 9 Designing an undirected graphical model

In this question we design a very simple undirected graphical model for denoising the output of a black-and-white printer with a defective cartridge head. Our old printer is defective in two ways. Sometimes when the correct output is black it outputs white, and vice versa. Furthermore, sometimes the cartridge gets stuck and has difficulty putting down any ink for an entire row of pixels. We represent the unobserved true output for an $m \times n$ image as $Z \in [0, 1]^{m\times n}$, where 1 denotes black ink while 0 denotes no ink. Furthermore, we have unobserved $Y \in [0, 1]^m$, where 1 denotes that on this row of pixels the cartridge is stuck and thus puts down less ink. Finally, we have observed data $X \in [0, 1]^{m\times n}$. We parameterize the distribution as follows:

$$\begin{aligned}
\psi_\alpha(Z_{ij}) &= \alpha^{\{Z_{ij}=1\}} \\
\psi_\beta(Z_{ij}, Z_{i,j+1}) &= \beta^{\{Z_{ij}=Z_{i,j+1}\}} \\
\psi_\gamma(Z_{ij}, Z_{i+1,j}) &= \gamma^{\{Z_{ij}=Z_{i+1,j}\}} \\
\psi_\theta(Z_{ij}, X_{ij}) &= \theta^{\{X_{ij}=Z_{ij}\}} \\
\psi_\mu(Y_i, X_{ij}) &= \mu^{\{X_{ij}=1 \wedge Y_i=1\}}
\end{aligned}$$

Problem 9.1

Draw the corresponding graphical model for $m = 3, n = 3$. Also, for each of the parameters $(\alpha, \beta, \gamma, \theta, \mu)$, based on your intuition state whether each is probably < 1 or > 1.

Problem 9.2

Suppose we formulate this as a CRF where we condition on $\mathbf{Y}$. Write out the partition function that corresponds to $P(\mathbf{Y}, \mathbf{Z}|\mathbf{X}; \alpha, \beta, \gamma, \theta, \mu)$.

Problem 9.3

Suppose

$$X = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \quad Y = [1 \ \ 0 \ \ 0]^T, \quad \text{and } Z = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

State which parameter setting leads to a greater conditional data likelihood for this observation:

(a) $\alpha = 1.3, \beta = 1.3, \gamma = 1.4, \theta = 1.0, \mu = 0.8$

(b) $\alpha = 0.5, \beta = 1.5, \gamma = 1.0, \theta = 0.8, \mu = 1.2$

Please include your code.

Problem 10 Restricted Boltzmann Machines

Restricted Boltzmann Machines (RBMs) are a class of Markov networks that have been used in several applications, including image feature extraction, collaborative filtering, and recently in deep belief networks. An RBM is a bipartite Markov network consisting of a visible (observed) layer and a hidden layer, where each node is a binary random variable. One way to look at an RBM is that it models latent factors that can be learned from input features. For example, suppose we have samples of binary user ratings (like vs. dislike) on 5 movies: Finding Nemo (V_1), Avatar (V_2), Star Trek (V_3), Aladdin (V_4), and Frozen (V_5). We can construct the following RBM:

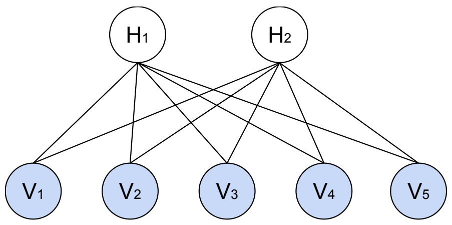

Figure 6: An example RBM with 5 visible units and 2 hidden units.

Here, the bottom layer consists of visible nodes $V_1, ..., V_5$ that are random variables representing the binary ratings for the 5 movies, and H_1, H_2 are two hidden units that represent latent factors to be learned during training (e.g., H_1 might be associated with Disney movies, and H_2 could represent the adventure genre). If we are using an RBM for image feature extraction, the visible layer could instead denote binary values associated with each pixel, and the hidden layer would represent the latent features. However, for this problem we will stick with the movie example. In the following questions, let $V = (V_1, ..., V_5)$ be a vector of ratings (e.g. the observation $v = (1, 0, 0, 0, 1)$ implies that a user likes only Finding Nemo and Aladdin). Similarly, let $H = (H_1, H_2)$ be a vector of latent factors. Note that all the random variables are binary and take on states in $\{0, 1\}$. The joint distribution of a configuration is given by

$$p(V = v, H = h) = \frac{1}{Z} e^{-E(v,h)} \tag{1}$$

where

$$E(v, h) = -\sum_{ij} w_{ij} v_i h_j - \sum_i a_i v_i - \sum_j b_j h_j$$

is the energy function, $\{w_{ij}\}, \{a_i\}, \{b_j\}$ are model parameters, and

$$Z = Z(\{w_{ij}\}, \{a_i\}, \{b_i\}) = \sum_{v,h} e^{-E(v,h)}$$

is the partition function, where the summation runs over all joint assignments to V and H.

Problem 10.1

Using Equation (1), show that $p(H|V)$, the distribution of the hidden units conditioned on all of the visible units, can be factorized as

$$p(H|V) = \prod_j p(H_j|V) \tag{2}$$

where

$$p(H_j = 1|V = v) = \sigma\left(b_j + \sum_i w_{ij} v_i\right)$$

and $\sigma(s) = \frac{e^s}{1+e^s}$ is the sigmoid function. Note that $p(H_j = 0|V = v) = 1 - p(H_j = 1|V = v)$.

Problem 10.2

Give the factorized form of $p(V|H)$, the distribution of the visible units conditioned on all of the hidden units. This should be similar to what's given in part 1, and so you may omit the derivation.

Problem 10.3

Can the marginal distribution over hidden units $p(H)$ be factorized? If yes, give the factorization. If not, give the form of $p(H)$ and briefly justify.

Problem 10.4

Based on your answers so far, does the distribution in Equation (1) respect the conditional independencies of Figure (6)? Explain why or why not. Are there any independencies in Figure 6 that are not captured in Equation (1)?

Problem 10.5 (Optional)

We can use the log-likelihood of the visible units, $\log p(V = v)$, as the criterion to learn the model parameters $\{w_{ij}\}, \{a_i\}, \{b_j\}$. However, this maximization problem has no closed form solution. One popular technique for training this model is called "contrastive divergence" and uses an approximate gradient descent method. Compute the gradient of the log-likelihood objective with respect to w_{ij} by showing the following:

$$\begin{aligned}\frac{\partial \log p(V = v)}{\partial w_{ij}} &= \sum_h p(H = h|V = v)v_i h_j - \sum_{v,h} p(V = v, H = h)v_i h_j \\ &= \mathbb{E}\left[V_i H_j | V = v\right] - \mathbb{E}\left[V_i H_j\right]\end{aligned}$$

where $\mathbb{E}\left[V_i H_j | V = v\right]$ can be readily evaluated using Equation (2), but $\mathbb{E}\left[V_i H_j\right]$ is tricky as the expectation is taken over not just H_j but also V_i.

Hint 1: To save some writing, do not expand $E(v, h)$ until you have $\frac{\partial E(v,h)}{\partial w_{ij}}$.

Hint 2: The partition function, Z, is a function of w_{ij}.

Problem 10.6

After training, suppose $H_1 = 1$ corresponds to Disney movies, and $H_2 = 1$ corresponds to the adventure genre. Which w_{ij} do you expect to be positive, where i indexes the visible nodes and j indexes the hidden nodes? List all of them.