Probabilistic Graphical Models

Part B HWK#1
Assigned Tuesday, Jan. 14, 2020
Due (both Parts A and B): Thursday, Feb. 6, 2020

3. Ising Graphical Model

1. Ising Markov random fields:
   Recall from lecture that an Ising model is a model where the nodes have binary variables and the distribution has pairwise interactions. Let $\mathbf{X} = \{X_1, \ldots, X_p\}$ be binary random variables associated with an undirected graph. For a face detection task in image processing, each variable in $\mathbf{X}$ might represent a labeling of the corresponding pixel location as "face" or "not face". An Ising Markov random field takes the form:

$$p(\mathbf{X}; \theta) \propto \exp\left\{ \sum_{s \in V} \theta_s X_s + \sum_{(s,t) \in E} \theta_{s,t} X_s X_t \right\}$$

For a given node $i \in V$, show that the conditional $P(X_i = 1 | \mathbf{X}_{-i}; \theta)$ is given by a logistic regression model.

2. Ising conditional random fields:
   In this question, we consider a conditional random field (CRF) extension of the Ising model. We are now able to condition on feature variables $\mathbf{W} = \{W_1, \ldots, W_q\}$. Going back to the face detection example, each variable in $\mathbf{X}$ corresponds to whether a pixel location is a face or not, while variables in $\mathbf{W}$ might be observed image caption information. The conditional distribution is written as:

$$p(\mathbf{X}|\mathbf{W}; \theta, \beta) = \exp\left\{ \sum_{s \in V} \theta_s X_s + \sum_{(s,t) \in E} \theta_{s,t} X_s X_t + \sum_{s \in V, u \in [q]} \beta_{su} X_s W_u \right\} / Z(\mathbf{W}, \theta, \beta)$$

$$Z(\mathbf{W}, \theta, \beta) = \sum_{\mathbf{X}} \exp\left\{ \sum_{s \in V} \theta_s X_s + \sum_{(s,t) \in E} \theta_{st} X_s X_t + \sum_{s \in V, u \in [q]} \beta_{su} X_s W_u \right\}$$

   Note that we now have a parameter β relating $\mathbf{X}$ to $\mathbf{W}$. Often to learn CRFs we minimize the log-likelihood using gradient descent. Compute the derivative of the log-likelihood on a dataset containing a single sample with respect to each of the parameters (θ_s, θ_{st}, and β_{su}).

4. 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
```

⋮

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.

4.1 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.

4.2 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 5: The undirected graph in Fig. 1 represents a Markov network with nodes x_1, x_2, x_3, x_4, x_5 counting clockwise around the pentagon with potentials $\phi(x_i, x_j)$. Show that the joint distribution can be written as

$$p(x_1, x_2, x_3, x_4, x_5) = \frac{p(x_1, x_2, x_5)p(x_2, x_4, x_5)p(x_2, x_3, x_4)}{p(x_2, x_5)p(x_2, x_4)}$$

and also express the marginal probability tables (i.e., the terms in the numerator and denominator of the above formula) explicitly as functions of the pair-wise potentials $\phi(x_i, x_j)$.

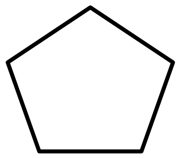

Fig. 1: Markov Network for Exercise 4.6