CSE 143: Assignment 3

Jacob Doar jjbagins@ucsc.edu
Daniel Xiong dxiong5@ucsc.edu
Edward Christenson edfchris@ucsc.edu

Due March 15, 2020

1 Introduction

In this assignment, we explored named entity recognition (NER), a sequence labeling task. We derived the Viterbi algorithm used to decode tag sequences and then implemented it in Python. We then trained our model using Stochastic Gradient Descent (SGD).

2 Deriving the Viterbi Algorithm

In sequence labeling, we want to find the sequence of tags $\hat{\boldsymbol{y}}$ that maximizes a scoring function $\boldsymbol{S}$, given an input sentence $\boldsymbol{x}$ and a set of tag sequences $\boldsymbol{y}$:

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, S(x, y). \tag{1}$$

The scoring function S maps x and y to a real number:

$$S(x, y) = \sum_{i=1}^{n+1} s(x, i, y_{i-1}, y_i) \tag{2}$$

where s is a local scoring function.

2.1 Deliverables

Question 1

Let, for every possible value of y_j:

$$\heartsuit_j(y_j) = \max_{y_{1:j-1}} \sum_{i=1}^{j} s(x, i, y_{i-1}, y_i). \tag{3}$$

We want to show

$$\heartsuit_j(y_j) = \max_{yj-1} s(x, i, y_{i-1}, y_i) + \heartsuit_{j-1}(y_{j-1}). \tag{4}$$

The derivation is as follows:

$$\begin{aligned}
\heartsuit_j(y_j) &= \max_{y_{1:j-1}} \sum_{i=1}^{j} s(x, i, y_{i-1}, y_i) \\
&= \max_{y_{j-i}} s(x, j, y_{j-1}, y_j) + \max_{y_{1:j-2}} \sum_{i=1}^{j-1} s(x, i, y_{i-1}, y_i) \\
&= \max_{y_{j-i}} s(x, j, y_{j-1}, y_j) + \heartsuit_{j-1}(y_{j-1})
\end{aligned}$$

where the $\max_{y_{1:j-2}} \sum_{i=1}^{j-1} s(x, i, y_{i-1}, y_i)$ term is equivalent to and substituted by the prior iteration of $\heartsuit_j(y_j)$, $\heartsuit_{j-1}(y_{j-1})$.

Question 2

The Viterbi algorithm pseudocode given in Chapter 7 of Eisenstein is as follows:

Algorithm 1 Viterbi algorithm

1: **for** $k \in 0, ..., K$ **do**
2: $\quad v_1(k) = s_1(k, \square)$
3: **end for**
4: **for** $m \in 2, ..., M$ **do**
5: $\quad$ **for** $k \in 0, ..., K$ **do**
6: $\quad\quad v_m(k) = \max_{k'} s_m(k, k') + v_{m-1}(k')$
7: $\quad\quad b_m(k) = \text{argmax}_{k'} s_m(k, k') + v_{m-1}(k')$
8: $\quad$ **end for**
9: **end for**
10: $y_M = \text{argmax}_k s_{M+1}(\blacksquare, k) + v_M(k)$
11: **for** $m \in M - 1, ..., 1$ **do**
12: $\quad y_m = b_m(y_m + 1)$
13: **end for**
14: **return** $y_{1:M}$

The time complexity for this is $O(MK^2)$. The for-loop on lines 1-3 iterates K times, resulting in a complexity of $O(K)$. The loop on lines 4-9 iterates M times and contains an inner for-loop. The inner for-loop on lines 5-8 iterates K times, and the max and argmax operations on lines 6-7 take $O(K + K)$ (amortized $O(K)$) time. Multiplying these gives $O(K^2)$ for lines 5-8. Therefore, the time complexity for the nested loop is $O(MK^2)$. Line 10 produces a Big-O of $O(K)$, since the argmax operation takes $O(K)$ time. The final for-loop on lines 11-13 iterates M times, giving it a complexity of $O(M)$. Summing all of these up gives $O(K + MK^2 + K + M)$, which amortizes to $O(MK^2)$.

3 Implementing the Viterbi Algorithm

We implemented the Viterbi decoding algorithm in Python, and ran it on the given `ner.dev` and `ner.test` datasets using the given model. The outputs are in `ner.dev.out` and `ner.test.out`.

	Precision	Recall	F1
ner.dev	59.80%	41.25%	48.82%
ner.test	53.28%	37.41%	43.96%

4 Training

After implementing the Viterbi algorithm we implemented Stochastic Gradient Descent. We trained a model for 10 epochs with early stopping on `ner.dev` and tested it on `ner.test`. The model is in `model.iter6`.

	Precision	Recall	F1
ner.dev	82.06%	69.65%	75.35%
ner.test	76.82%	58.32%	66.30%

For the most part, our output on the dev set corresponded correctly with the gold labels. At some instances though, we did get incorrect labeling. Some common errors in the dev set output seemed to revolve around names and numbers. Particularly when it came to labeling last names, the gold label would be "I-PER" but we got a "O" label. This happened a decent number of times. When it came to labeling numbers the gold label would be "O" but we got "I-ORG", "I-PER", and "I-LOC" quite a few times. Also the mislabeling of countries came up a bit. The model did have good accuracy over all, and these were just some common patterns that were noticeable.

Highest feature had a weight of 32 and the lowest had a weight of -36. The features t=O+w=CHAMPIONSHIPS and t=O+w=harder were interesting since they both have weight 0, as well as the same label.

	feature-string	feature-weight
Highest 1	t=I-LOC+w=Britain	32
Highest 2	t=I-LOC+w=Russia	32
Highest 3	t=I-ORG+w=Newsroom	28
Highest 4	t=I-PER+w=David	28
Highest 5	t=I-LOC+w=Sweden	27

	feature-string	feature-weight
Lowest 1	t=O+w=U.S.	-36
Lowest 2	t=O+w=Australia	-33
Lowest 3	t=O+w=United	-29
Lowest 4	t=O+w=Russia	-26
Lowest 5	t=O+w=Party	-24