Homework 02: Sequential Models

CS 6501-005 Natural Language Processing

Updated on: October 4, 2019

1. **Conditional Random Fields** As we explained in class, one challenge of learning a conditional random fields is to compute the denominator of the following equation

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}))}{\sum_{\boldsymbol{y}' \in \mathcal{Y}^T} \exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}'))} \tag{1}$$

where $\boldsymbol{x} = (x_1, \ldots, x_T)$, in the case of POS tagging, is a sequence of words with length T, $\boldsymbol{y} = (y_1, \ldots, y_T)$ is the corresponding sequence of POS tags, and $\mathcal{Y}^T$ is the set of all possible POS sequences with length T. Assume the size of the pre-defined POS set is K, simple brute force algorithm of computing $\sum_{\boldsymbol{y}' \in \mathcal{Y}^T} \exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}'))$ will have the complexity K^T, which is almost impossible in practice. On the other hand, with the decomposition of $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$ as

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{T} \boldsymbol{f}_i(x_i, y_i, y_{i-1}) + f_{T+!}(y_{T+1}, y_T), \tag{2}$$

with $y_0 = \square$ and $y_{T+1} = \blacksquare$, we are able to design an algorithm with time complexity $\mathcal{O}(T \cdot K^2)$. Note that, in equation (2) on lecture 08 slides, we ignore $f_{T+!}(y_{T+1}, y_T)$ for simplicity.

   - (3 points) Please follow a similar idea of Viterbi decoding to design a dynamic programming algorithm that can compute

$$\sum_{\boldsymbol{y}' \in \mathcal{Y}^T} \exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}')).$$

     with time complexity $\mathcal{O}(T \cdot K^2)$. [Hint: The framework of your algorithm should look like Algorithm 11 on page 150 in [Eisenstein, 2018], without the back tracing part.]
   - (1 point) Please justify the time complexity of this algorithm is $\mathcal{O}(T \cdot K^2)$.

2. **POS Tagging** From the attachment, you can find three files for this part.

   - `trn.pos`
   - `dev.pos`
   - `tst.pos`

   In both `trn.pos` and `dev.pos`, each line is one sentence, and each token consists of a word and its POS tag, as

`word`/POS

   As you may notice, the POS tag set is much smaller than the conventional POS tag set in the Penn Treebank. There are only 10 tags in total

$$\mathcal{Y} = \{\text{A, C, D, M, N, O, P, R, V, W}\}$$

Remember, there are also two special tags for POS tagging: □ and ■. We are going to estimate both transition and emission probabilities first, then use them to build a POS tagger for the data in the dev set.

Please following the steps to build your POS tagger

(a) Preprocessing. Scan the training examples in `trn.pos` and map the words with frequency less than $K = 2$ into a special unknown token UNK.

(b) From the training examples in `trn.pos`, estimate the transition probability as

$$p(y \mid y') = \frac{\#(y, y') + \alpha}{\#(y') + \alpha \cdot (|\mathcal{Y}| + 1)} \tag{3}$$

where y and y' can be any two possible tags in $\mathcal{Y}$ and $|\mathcal{Y}|$ is the size of $\mathcal{Y}$, and $\alpha = 1$. Note that, there are alsoe two special cases $p(y \mid \square)$ and $p(\blacksquare \mid y')$ that you need to consider. Make sure $p(y \mid y')$ are valid probabilities, such as

$$\sum_{y \in \mathcal{Y} \cup \{\blacksquare\}} p(y \mid y') = 1 \tag{4}$$

where $y' \in \mathcal{Y} \cup \{\square\}$.

(c) From the training examples in `trn.pos`, estimate the emission probability as

$$p(x \mid y) = \frac{\#(x, y) + \beta}{\#(y) + \beta \cdot |\mathcal{V}|} \tag{5}$$

where $y \in \mathcal{Y}$, $x \in \mathcal{V}$ is any word in the vocab $\mathcal{V}$, $|\mathcal{V}|$ is the size of $\mathcal{V}$, and $\beta = 1$. Again, make sure $p(x \mid y)$ are valid probabilities, such as

$$\sum_{x \in \mathcal{V}} p(x \mid y) = 1 \tag{6}$$

(d) (2 points) Convert the estimated probabilities from the previous step into log space and implement the Viterbi decoding algorithm as in Algorithm 11in [Eisenstein, 2018]. Report the accuracy of your decoder on the dev set `dev.pos`.

(e) (2 points) Tune α and β to find the best accuracy on the dev set. Report the best accuracy number and the values of α and β.

(f) (2 points) Run the model selected from the previous step on the test set `tst.pos`, and report the accuracy.

Note that

- Please implement your code from scratch. You are not allowed to use *any* existing machine learning package to solve this problem — it's OK to use NumPy and SciPy if you want.
- To avoid zero point on this problem, please submit your code (including the best values of α and β) with file name **[ComputingID]-pos.py** or **[ComputingID]-pos.ipynb** (if you use IPython notebook).

References

[Eisenstein, 2018] Eisenstein, J. (2018). *Natural Language Processing*. MIT Press.