CSE P 517: Homework 3
Structured Learning

Due: Sunday Feb 14 2021 11:59pm

Submission instructions Submit 1 file on Canvas:

1 **Report:**

(`HW.pdf`): Please type most of your answers whenever possible. You can include handwritten answers for some questions, in which case, (1) please write them very neatly so that they are legible, and then (2) scan the handwritten answers and include them into your doc as pictures, so that you can make an electronic submission that includes all answers in one file. If you are having trouble formatting your document, please inform the TAs at least 48 hours before the submission deadline so that we can arrange a solution in time.

2 **Code:**

(`HW3.tgz`): You will submit your code together with a neatly written README file to instruct how to run your code with different settings. We prefer Python for compatibility and concision. Discuss with the teaching crew in advance if you must use some other programming language. We assume that you always follow good practice of coding (commenting, structuring) and we will not grade your code based on such qualities of code writing practice.

1 Named-Entity Recognition (NER) with Structured Perceptrons

In this problem, you will think about how to construct perceptron algorithms for structured learning and inference. Do not worry if you haven't learned the perceptron algorithm before! This problem is self-contained. To convert a vanilla perceptron algorithm into a structured one, you will need to apply the same intuition we used for converting MaxEnt models into CRF models. In the process, you will also think more carefully about the internals of feature-rich models.

Named entities are phrases that contain the names of people, organizations, locations, etc [3]. For example, in the following NER-tagged sentence[1],

```
[ORG U.N. ] official [PER Ekeus ] heads for [LOC Baghdad ] .
```

U.N is an *organization*, Ekeus is a *person*, and Baghdad is a *location*.

NER as sequence tagging: NER can be formulated as a sequence tagging problem using **BIO encoding**, where the **B**eginning word of a named entity is marked with **'B'**, the following words of a named entity is marked with **'I'** (i.e., inside) and words that are not part of an entity is marked with **'O'** (i.e., outside). These labels are further marked with the types of named entities (e.g., ORG, PER, LOC). See section 1.2 for a concrete example of the tagging scheme.

Feature Vectors: You are free to design the feature vector however you like. You can start with feature templates similar to those presented in page 14 of LogLinear.pdf slides, or you can look up Table $1-3$ of [4]. See J&M textbook Ch#22.1 if you would like to see yet additional examples. But it's also fine if you'd rather be creative and come up with your own features without reading all these references.

1.1 Sequential Tagging with Structured Perceptrons

The pseudo code in **Algorithm 1** on the next page provides the recipe for learning the structured perceptron [2], which is almost identical to the perceptron algorithm for simple classification, except that $\mathbf{y}$ here is a sequence of random variables ($\mathbf{y} = y_1, ..., y_n$) as opposed to one random variable.

The Equations (1) and (2) summarize the perceptron algorithm — you predict $\tilde{\mathbf{y}}$ of a training example $\mathbf{x^i}$ using the current weight vector $\mathbf{w}$ (Equation (1)). If the current weight vector leads to an incorrect prediction, then update the parameters with the difference between the feature vector of the correct prediction and the feature vector of the incorrect prediction (Equation (2)).

The perceptron algorithm is simpler than many current approaches, yet often yields performance almost as good as state-of-the-art, if given a good feature vector.

The reason why we take the averaged weight vector $\bar{\mathbf{w}}$ is because it gives a bit of a regularization effect (i.e., counteracting overfitting). But sometimes people take the last vector $\mathbf{w}$ directly as the final learned parameters.

Remember that you will need to factorize the global feature vector $\Phi(\mathbf{x}^i, \mathbf{y})$ as a summed vector over local feature vectors, for example,

$$\Phi(\mathbf{x}^i, \mathbf{y}) = \sum_k \phi(\mathbf{x}^i, k, y_k, y_{k-1})$$

or even

$$\Phi(\mathbf{x}^i, \mathbf{y}) = \sum_k \phi(\mathbf{x}^i, k, y_k, y_{k-1}, y_{k-2})$$

Keep in mind that depending on how you factorize your feature vector, the details of your viterbi algorithm will change a bit. In this assignment, it is your job to figure out the technical details on your own.

Algorithm 1 Structured Perceptron Algorithm for Sequential Tagging

Input: M tagged sequences $\mathbf{x}^i, \mathbf{y}^i$, $i = 1 \ldots M$ as training data. Number of iterations T.

Initialization: Randomly initialize parameters $\mathbf{w}$, and let $\bar{\mathbf{w}}$ denote averaged parameters.
count $= 0$
for $t = 1 \ldots T,\ i = 1 \ldots M$ **do**
  Predict the current best sequence $\tilde{\mathbf{y}}$ for $\mathbf{x}^i$ under current parameters $\mathbf{w}$:

$$\tilde{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y}} \{\mathbf{w} \cdot \Phi(\mathbf{x}^i, \mathbf{y})\} \tag{1}$$

  if $\tilde{\mathbf{y}} \neq \mathbf{y}^i$ **then**
    Update parameters:

$$\mathbf{w} \leftarrow \mathbf{w} + \Phi(\mathbf{x}^i, \mathbf{y}^i) - \Phi(\mathbf{x}^i, \tilde{\mathbf{y}}) \tag{2}$$

    Update averaged parameters: $\bar{\mathbf{w}} \leftarrow \bar{\mathbf{w}} + \mathbf{w}$
    count $\leftarrow$ *count* $+ 1$
  end
end
Output: Averaged parameter vector $\bar{\mathbf{w}}/count$

When to stop training? You can determine the value of T dynamically based on your model's performance on the dev set. It is common practice in deep learning to halt training once performance on the dev set stops improving.

Keep in mind, if you check the performance on the dev set too frequently, it will slow down the learning procedure. Thus, you can check at regular intervals, for example, $T = 1, 5, 10, 15, \ldots$ Exact design decisions are up to you!

1.2 The CoNLL-2003 NER Dataset

In *conll03_ner.tar.gz*, you will be given the full and the reduced set of the CoNLL-2003 NER corpus [1] [3], feel free to use either of them. The number of sentences and tokens for the dataset are shown in Table 1. There are only four types of named entities in CoNLL-2003: *Person*, *Location*, *Organization* and *Miscellaneous*.

	Full set			Reduced set		
	File	Sentences	Tokens	File	Sentences	Tokens
Training set	*eng.train*	14,987	203,621	*eng.train.small*	3,000	41,732
Development set	*eng.dev*	3,466	51,362	*eng.dev.small*	500	7,342
Test set	*eng.test*	3,684	46,435	*eng.test.small*	500	6,288

Table 1: Number of sentences and tokens in the NER dataset.

Input Format & Tagging Scheme

The data files are in the following format:

```
U.N.         NNP  I-NP  I-ORG
official     NN   I-NP  O
Ekeus        NNP  I-NP  I-PER
heads        VBZ  I-VP  O
for          IN   I-PP  O
Baghdad      NNP  I-NP  I-LOC
.            .    O     O
```

Each line contains four whitespace-separated columns: *word*, *POS tag*, *syntactic chunk tag* and *NER tag*. The first three columns (word, pos-tag, chunk-tag) are part of the input, the fourth column (NER-tag) is what you need to predict. Given that there are four different NER types (*Person*, *Location*, *Organization* and *Miscellaneous*), what is the cardinality of your tag set? Note that both the syntactic chunk tag and the NER tag use BIO encoding. Sentence boundaries are marked by a single blank line. You can find more details about the input/output format here: `http://www.cnts.ua.ac.be/conll2003/ner/`.

Evaluation

For evaluation, you will need to append your predicted tag at the end of each input line, write the sentences into a file (i.e. output.txt), and run the *conlleval.txt* script:

```
./conlleval.txt < output.txt
```

Here is an example of the output file, with the predicted tags in the fifth column:

```
U.N.         NNP  I-NP  I-ORG   O
official     NN   I-NP  O       O
Ekeus        NNP  I-NP  I-PER   I-PER
```

```
heads        VBZ  I-VP  O      O
for          IN   I-PP  O      O
Baghdad      NNP  I-NP  I-LOC  I-PER
.            .    O     O      O
```

If your output format is correct, you will see a message like this:

```
processed 7342 tokens with 817 phrases; found: 822 phrases; correct: 644.
accuracy:  96.30%; precision:  78.35%; recall:  78.82%; FB1:  78.58
          LOC: precision:  81.92%; recall:  83.53%; FB1:  82.72  260
         MISC: precision:  82.52%; recall:  76.58%; FB1:  79.44  103
          ORG: precision:  75.45%; recall:  65.28%; FB1:  70.00  167
          PER: precision:  75.34%; recall:  85.27%; FB1:  80.00  292
```

1.3 Deliverables

- Please include your code! However, you will not be specifically graded on it and your responses to the questions below should be clear enough to understand what you did.
- (2pt) Define the feature templates you used for implementing the structured perceptron classifier. Propose 3 additional feature templates that (1) will likely be useful for improving NER performance and (2) are not part of your implementation.
- (2pt) Write out the Viterbi decoding algorithm for the structured perceptron using one recursive equation.
- (2pts) **Ablation Study:** In Ablation studies, we remove a subset of features (i.e. a subset of feature templates) to understand how those features affect performance. Design your own ablation study and report how the performance varies with respect to both the dev set and the test set. Try no more than 4 feature vector variations, including your full model with complete feature set.
- (2pt) While BIO encoding is the most common, there are a number of other encoding schemes (`https://lingpipe-blog.com/2009/10/14/coding-chunkers-as-taggers-io-bio-bmewo-and-bmewo/`) also used in practice. In fact, some studies have reported that **IO encoding**, which does not differentiate **'B'** from **'I'**, often results in comparable (or even better) performance depending on the data and the task. Why might this be the case? Discuss the potential pros and cons of BIO and IO encoding schemes for NER.
- (2pt) Error analysis and discussion of results. Including and discussing specific error cases you found will be useful here!

1.4 Tips

- The accuracy will depend on the feature vector you design. As a point of reference, the accuracy of competitive systems on the full data should be in the range of 75% - 90% F1 and the accuracy on the reduced data will be in the range of 65%-80% F1. You can find performance of other systems in the CoNLL-2003 summary paper [3] or at the ACL wiki: `http://www.aclweb.org/aclwiki/index.php?title=CONLL-2003_(State_of_the_art)`.
- Using too many features might make training very slow or cause over-fitting. You could limit the number of features by discarding features that occur less than k times in the training set, where typical choice of k could be somewhere between 3 - 5.
- Your feature vectors will be extremely high dimensional with mostly zero entries. Thus arrays shouldn't be your choice of data structure.

- We don't expect you to reach state-of-the-art performance, but you should make a reasonable effort to improve your model from a basic set of features. Your discussion and ablation study will be a great place to demonstrate this!

References

[1] Language-independent named entity recognition (ii). `http://www.cnts.ua.ac.be/conll2003/ner/`.

[2] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics, 2002.

[3] Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics, 2003.

[4] Maksim Tkachenko and Andrey Simanovsky. Named entity recognition: Exploring features. 2012.