CSE 517: Homework #4
Structured Learning with Feature-Rich Models

Due Mon Nov 19 2018 11:59pm

Submission instructions As usual, submit 2 files on Canvas — **Code** (`HW5.tgz`) and **Report** (`HW5.pdf`). The usual submission guidelines apply.

1 Variations on the Cocke–Younger–Kasami (CKY) Algorithm

Recall that a Context-Free Grammar is in Chomsky Normal Form (CNF) if every production rule in the grammar is of the form:

- $X \to Y\ Z$, or
- $X \to w$,

where X, Y, Z are nonterminal symbols and w is a terminal symbol. Consider a variant, which we will call 3-CNF, where rules of the following form are also allowed:

- $X \to V\ Y\ Z$

1.1 Deliverables

- (15pts) In pseudo-code, Write a modified CKY parsing algorithm that can work directly with a CFG in 3-CNF.

2 Named-Entity Recognition (NER) with Structured Perceptrons

Named entities are phrases denoting the names of people, organization, locations, etc. [3]. *Named Entity Recognition* (NER) is a common NLP task where a system must identify these phrases in natural language text. 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 Even though NER requires labeling *spans* and not *tokens*, we can formulate it as a sequence tagging problem using the **BIO encoding** scheme, where the **B**eginning word of a named entity is marked with **B**, the following words of a named entity are marked with **I** (inside), and words that are not part of an entity mention are marked with **O** (outside). These labels are further marked with the type of the entity (e.g., ORG, PER, LOC). See Section 2.2 for a concrete example.

2.1 Sequence Tagging with Structured Perceptrons

Algorithm 1 shows the perceptron learning algorithm [2] that you will use. The meat of the algorithm is in equations (1) and (2): First, we predict $\tilde{\mathbf{y}}$ for a training example $\mathbf{x^i}$ using the current weight vector $\mathbf{w}$ (Equation (1)). If the prediction is incorrect, then update the weights according to Equation (2), pushing the model away from its incorrect prediction and towards the correct one. While simple, the perceptron algorithm can lead to very good performance if given a good feature vector, even in the structured case. Sometimes people use the final value of the weights $\mathbf{w}$ as the learned parameters, but we will use the averaged weight vector $\bar{\mathbf{w}}$ as a form of regularization.

In this assignment you will use a **structured perceptron** for **sequence tagging**. This means your inputs x^i and outputs y^i will be sequences of tokens and tags, respectively. For the structured perceptron, the learning algorithm is the same, but the underlying implementation of Equation (1) will change: since the output space is large and arbitrary-length, you will have to factor your feature function into local parts and implement an appropriate structured decoding algorithm.

Algorithm 1: Perceptron Algorithm with Averaging

Input: M input-output pairs $\mathbf{x}^i, \mathbf{y}^i$, $i = 1 \dots M$ as training data. Number of iterations T.

Initialization: Randomly initialize parameters $\mathbf{w}$, and let $\bar{\mathbf{w}}$ denote averaged parameters.

$\texttt{count} = 0$

for *t = 1 ... T, i = 1 ... M* **do**

  Predict the current best output $\tilde{\mathbf{y}}$ for $\mathbf{x}^i$ under current parameters $\mathbf{w}$:

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

  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: $\bar{\mathbf{w}} \leftarrow \bar{\mathbf{w}} + \mathbf{w}$

    $\texttt{count} \leftarrow \texttt{count} + 1$

  end

end

Output: Averaged parameter vector $\bar{\mathbf{w}}/\texttt{count}$

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

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

or perhaps even

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

The details of your decoding algorithm will depend on how you do this factorization. In this assignment, it is your job to figure out the technical details on your own.

As for the local feature vector, you are free to design it however you like. You can find examples in the slides on Log-Linear Models (Oct 30), Tables 1 – 3 of [4], or See J&M Ch#22.1. But it's also fine if you'd rather be creative and come up with your own features without reading all these references. Some features may only look at the current input or output in the sequence, while others may look further out in the sequence or at nearby tags.

When to stop training? You can determine T based on dev set performance. If you check the performance too frequently, it slows down the learning procedure, so it is typical to check it at intervals, for

example at iterations $T = 1, 5, 10, 15$, etc.

2.2 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. 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: The 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 the BIO encoding scheme. 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
```

2.3 Deliverables

- (20pts) Report the features and feature templates you used in your structured perceptron classifier. Propose 3 additional feature templates that you think could be useful for improving NER performance, but are not part of your implementation.
- (15pts) How you chose to factor the feature vector will likely produce dependencies between the choice of tag for each token in an input sequence. Explicitly describe these dependencies, and write out the equation(s) for decoding the best-scoring tag sequence in your model.
- (20pts) Report your model's results on dev and test. Look through a sample of your model's incorrect predictions and see if you can identify some common types of errors. Report at least 3 example sentences from the dev set where the model made mistakes and interpret the results.
- (20pts) **Ablation Study:** Ablation studies are used to understand what kind of signal is important for a classification problem. An *ablation* of your model is a variant with some subset of the features or feature templates removed. To perform a study, choose several ablations of your model, train them, evaluate them, and compare their performance to the full (non-ablated) model. Design a study with 4 ablations (in addition to the full model) and report the results on the dev and test sets. Discuss the results of your study and see what you can learn about the effects of various kinds of features.
- (10pts) While the BIO encoding scheme is the most common, a number of other encoding schemes[1] are also used in practice. In fact, some studies have reported that the **IO encoding** scheme, which does not differentiate **B** from **I**, often results in comparable or better performance than BIO, depending on the data and task. Why might this be the case? Discuss the potential pros and cons of BIO and IO encoding schemes for NER.

2.4 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[2].
- 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 zeros, so arrays may not be a good choice of data structure.

[1] https://lingpipe-blog.com/2009/10/14/coding-chunkers-as-taggers-io-bio-bmewo-and-bmewo/
[2] http://www.aclweb.org/aclwiki/index.php?title=CONLL-2003_(State_of_the_art)

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.