11-763 Spring 2018: Homework 1

Due: February 12 2018, 23:59 PM

1 Introduction

The purpose of this homework assignment is to gain familiarity with a structured prediction task - specifically, sequence labelling - and to develop a codebase that you can use in future assignments. You will implement a Named Entity Recognition (NER) system and evaluate it on the `CoNLL 2003` shared task.

2 Task

Your task is to implement a BIO tagger for four kinds of entities: people, locations, organizations, and names of other miscellaneous entities. Given an input sentence x, the job of the tagger is to find the highest scoring tag sequence $\hat{y}$ under the linear model:

$$\hat{y} = argmax_y \overrightarrow{w}.\overrightarrow{f}(x, y) \tag{1}$$

We are providing the weights file for a fully trained tagger, and you will need to write code for the Viterbi algorithm, feature functions, and support code. You do not need to write code for training, but you still get the joy of producing a fully functioning tagger! (We will be providing a `train` file in case you want to experiment with training your own tagger, but this is not required.)

3 Data Format

The data is in four columns separated by a space. Each token in the sentence has been put on a separate line, and the sentences are separated by a blank line. The first column is the token, and second is the part-of-speech tag, the third is a syntactic chunk tag (which we will not use), and the fourth is the named entity (NE) tag. The NE tags use a modified BIO tagging scheme, where O denotes outside, I denotes inside or begin, and B is only used for the start of a new NE in the case of two identically tagged NEs in a row. Here is an example sentence:

```
  France NNP I-NP I-LOC
and CC I-NP O
Britain NNP I-NP I-LOC
backed VBD I-VP O
Franz NNP I-NP I-PER
Fischler NNP I-NP I-PER
's POS B-NP O
proposal NN I-NP O
.  .  O O
```

4 Tagging

Your tagger should take as input a file in this format, ignore the last column (dont cheat!), and produce a new file with a fifth column containing the tags produced by your tagger. Do make sure to leave the fourth column unmodified as it contains our reference tagger's predictions. We will check to see if your code produces the same tags as the reference tagger and will also evaluate your tagger's accuracy against gold labels. We are providing two datafiles (with

four columns): `dev` which contains the output of the reference tagger on some data which you can use for debugging your code, and `test` which contains human annotated tags on some other data. We will compare your tagger to the reference tagger on `test`. **Do not worry about matching the output of the reference tagger exactly.** Our grading will tolerate some amount of variance due to implementation details.

We have tried to describe the features as thoroughly as possible, but there may be implementation details that you do not have to duplicate.

You can check the performance of your tagger by running the CoNLL 2003 evaluation script shared with the tar bundle. For reference, our reference tagger achieves a F1-measure score of 79.7% on `test`. Your tagger should achieve a F1-measure score of $> 95\%$ when compared to the reference tagger on `dev`.

5 Features

We will be using the following real-valued features. All of these are conjoined with the current tag (In our feature names, `:` represents conjunction). An example feature name and value is given for the word 'France' in the above sentence. Remember that although we show features for the tag sequence in the example, your Viterbi decoder will need to consider all tags for each word – i.e. your decoder will need to score each local configuration of possible tags using the feature function and weights vector.

1. Current word w_i.
   `Wi=France:Ti=I-LOC 1.0`
   We always put the ouput label (`Ti=LABEL`) at the end of feature names.

2. Lowercased word o_i.
   `Oi=france:Ti=I-LOC 1.0`

3. Current POS tag p_i.
   `Pi=NNP:Ti=I-LOC 1.0`

4. Shape of current word s_i. Just replace all letters by **a/A** depending on capitalization and digits by **d**.
   `Si=Aaaaaa:Ti=I-LOC 1.0`

5. Features 1-4 for the previous and next word. `<START>` and `<STOP>` symbols are added to the beginning and end of the sentence. When referencing them in Feature 5, we set the values as `<START>` and `<STOP>`. For `<STOP>`, we do not include the features for $i + 1$.
   `Wi-1=<START>:Ti=I-LOC 1.0`
   `Pi-1=<START>:Ti=I-LOC 1.0`
   `Wi+1=and:Ti=I-LOC 1.0`

6. The features 1-4 conjoined with the features in 5. In the feature name, we put the features 1-4 before the features in 5.
   `Wi=France:Wi+1=and:Ti=I-LOC 1.0`

7. Previous tag (t_{i-1}), as well as features 1-5 conjoined with the previous tag. The previous tag comes just before the current tag in the feature name.
   `Ti-1=<START>:Ti=I-LOC 1.0`
   `Wi+1=and:Ti-1=<START>:Ti=I-LOC 1.0`

8. Length k prefix for the current word, for $k \in [1, min(4, len(word))]$
   `PREi=F:Ti=I-LOC 1.0`

`PREi=Fr:Ti=I-LOC 1.0`

9. Is the current word in the gazetteer for the current tag? If `Galerie Intermezzo` is in the gazetteer, the feature value fires for both unigrams `Galerie` and `Intermezzo`.
   `GAZi=True:Ti=I-LOC 1.0`
10. Does the current word start with a capital letter?
    `CAPi=True:Ti=I-LOC 1.0`
11. Position of current word in the sentence (index starting from 1).
    `POSi=1:Ti=I-LOC 1.0`

6 Weight file

Each line in the weights file lists the feature name, a space, and the weight for that feature. Features that do not appear in the weights file have a weight of zero.

7 Codebase

We suggest you make your codebase fairly modular. Although you do not need to do it this way, our tagger is broken up into the following parts:

1. Decoder (implements Viterbi algorithm)
2. Feature Functions (implements the features listed above)
3. Load Data (for reading and writing the CoNLL data format)
4. Feature Vectors (for loading, saving, and math operations on feature and weight vectors)
5. Main program (parses command line arguments)

You may use any programming language for this assignment.

8 Submission

Run your tagger on 'test', and submit the output and a .zip or .tgz file of your code by 11:59pm on the due date. Details for submission on Canvas will be announced on Piazza.