Assignment 3
NLP 202: Natural Language Processing II

University of California Santa Cruz

Due: March 1, 2022, 11:59pm

This assignment is to be done in Python 3.

Your final writeup for this assignment, including the problem and the report of the experimental findings of the programming part, should be no more than four pages long. Most of your grade will depend on the quality of the report. You should submit it as a PDF. We *strongly* recommend typesetting your scientific writing using LaTeX. Some free tools that might help: Overleaf (online), TexLive (cross-platform), MacTex (Mac), and TexStudio (Windows).

1 Task

The purpose of this homework assignment is to gain familiarity with linear models and loss functions for a structured prediction task. You will implement a named entity recognition (NER) system and evaluate it on the CoNLL 2003 shared task [1]. Your task is to implement a BIO tagger for four kinds of entities: people, locations, organizations, and names of other miscellaneous entities.

2 Data format

The data format is 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 chuck 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
```

3 Tagging

Your tagger should take as input a file in this format, ignore the last column (don't cheat!), and produce a new file with a fifth column containing the tags produced by your tagger.

4 Features

We will use 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. Be sure to include start and stop symbols in $\boldsymbol{x}$ and $\boldsymbol{y}$, so $x_0 = \langle\text{START}\rangle$ and $x_{n+1} = \langle\text{STOP}\rangle$, and similarly for $\boldsymbol{y}$.

1. Current word w_i. Example:
   `Wi=France+Ti=I-LOC 1.0`
   We always put the ouput label (`Ti=LABEL`) at the end of feature names.
2. Previous tag t_{i-1}. Example:
   `Ti-1=<START>+Ti=I-LOC 1.0`
3. Lowercased word o_i. Example:
   `Oi=france+Ti=I-LOC 1.0`
4. Current POS tag p_i. Example:
   `Pi=NNP+Ti=I-LOC 1.0`
5. Shape of current word s_i. Just replace all letters with `a` or `A` depending on capitalization, and replace digits with `d`. Example:
   `Si=Aaaaaa+Ti=I-LOC 1.0`
6. The features 1-4 for the previous word, and the next word. Examples:
   `Wi-1=<START>+Ti=I-LOC 1.0`
   `Pi-1=<START>+Ti=I-LOC 1.0`
   `Wi+1=and+Ti=I-LOC 1.0`
   Note: `<START>` and `<STOP>` symbols are added to the beginning and end of the sentence, and features in 5 referencing them have values `<START>` and `<STOP>`. When we are at the `<STOP>` symbol, we do not include the features for $i + 1$.
7. Features 1, 3 and 4 conjoined with the previous tag. Example:
   `Wi+1=and+Ti-1=<START>+Ti=I-LOC 1.0`
8. Length k prefix for the current word, for $k = 1, 2, 3, 4$. Examples:
   `PREi=Fr+Ti=I-LOC 1.0`
   `PREi=Fra+Ti=I-LOC 1.0`
   Our convention is that if the current word is shorter than k for a prefix of length k, it is not duplicated for that prefix. So the word `to` fires the features `PREi=t+Ti=LABEL` and `PREi=to+Ti=LABEL` just once.
9. Is the current word in the gazetteer for the current tag? Example:
   `GAZi=True+Ti=I-LOC 1.0`
   Our convention is that if `Galerie Intermezzo` is in the gazetteer for the tag `LOC`, then this feature fires for both the unigrams `Galerie` and `Intermezzo`.
10. Does the current word start with a capital letter? Example:
    `CAPi=True+Ti=I-LOC 1.0`

11. Position of the current word (indexed starting from 1). Example:
    `POSi=1+Ti=I-LOC 1.0`

5 Implementation

Implement the Viterbi algorithm and train with the structured perceptron and structured SVM for a named entity recognition system. We are providing starter code that implements a simple NER system using a linear model with features, but with key pieces missing.

- **Feature Functions**: We have provided some basic features, but you need to implement all the features listed above.
- **Decoder**: You need to implement the Viterbi algorithm, which is given in the lecture slides from last quarter. You can copy and modify the algorithm you implemented last quarter.

6 Structured Perceptron Training: SSGD

Now that you have implemented the decoder, perform training with the structured perceptron loss function using stochastic subgradient descent (SSGD) and early stopping for this model. Because the perceptron loss function scales linearly with the weight vector, neither the regularizer nor the step size have a meaningful effect on the perceptron loss for a linear model. For this reason, use stepsize 1 and do not include a regularizer.

Deliverables In the writeup, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.

1. With features 1-4, report the precision, recall, and F1 of your model on the dev and test sets after training with early stopping on the dev set.
2. Submit the output of the model for both the dev and test sets.
3. With this limited feature set, look at the output on the dev set and compare to the gold labels. What kinds of errors do you see, and why do you think this is occurring? The model is trying to fit the training data as best it can, and sometimes has to make compromises. What patterns in the output tags do you seen that work well for some sentences but poorly for others?
4. With the full feature set, report the precision, recall, and F1 of your model on the dev and test sets after training with early stopping on the dev set.
5. Submit the output of this model for both the dev and test sets.
6. With the full feature set, look at the output on the dev set and compare to the gold labels. What errors have been fixed when compared to the limited feature set? What kinds of errors do you still see, and why do you think this is occurring?
7. With the full feature set, look at the model file. It is in the format of `feature_string feature_weight`. What are the 5 features with the highest weight in the model, and what are the 5 features with the lowest weight? What does this tell you? Give two features with weights that you find interesting or surprising.

Debugging To help debug your code, we suggest training on one sentence at first. The perceptron algorithm should converge to correctly label this sentence. If it doesn't, there may be a bug in the decoder or the training algorithm.

It is also helpful to print out the average value of the loss function at the end of each epoch, averaged over the epoch. It should roughly go down as training proceeds.

7 Structured Perceptron Training: Adagrad

Perform training with the structured perceptron loss function using the Adagrad optimizer. For the same reasons as before, use stepsize 1 and do not include a regularizer.

Adagrad is implemented the same as SSGD, but with a modified equation for updating the parameters. The formula for the Adagrad update is as follows. Like SSGD, each gradient update is called a time step. Let t count the number of gradient updates that have been performed (and does not reset after each epoch.) Let $g_{t,i}$ be the ith component of the gradient at time step t. We keep a running total of the sum of squares of all the components of all the previous gradients: $s_{t,i} = \sum_{\tau=1}^{t} g_{\tau,i}^2$. This includes gradients from all previous epochs. At time step t, the ith parameter $\theta_{t,i}$ is updated like so:

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{s_{t,i}}} g_{t,i}$$

Deliverables In the writeup, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.

1. Report the precision, recall, and F1 of your full feature set model on the dev and test sets after training with early stopping on the dev set.
2. Submit the output of your model for both the dev and test sets.

8 Structured SVM Training

Implement structured SVM training with early stopping for this model. Use hamming distance times 10 (hamming distance is the number of tags that are different between the gold standard y and the candidate y'). Remember, cost augmented decoding for this cost function can be implemented as another feature during Viterbi decoding. This time include an L_2 regularizer and tune the stepsize.

Deliverables In the writeup, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.

1. Report the precision, recall, and F1 of the model with the full feature set on the dev set after training with early stopping. Try to tune the step size and regularizer strength to get the best F1. (However, we won't be grading you on your final score). Report precision, recall and F1 on the dev set for at least three different step sizes and three different regularizer strengths.
2. Report precision, recall, and F1 for your best model on the test set.
3. Submit the output of your best model for both the dev and test sets.

9 Structured SVM Training: Modified Cost Function

An important property of the SVM loss is you can penalize some errors more than others during training. Modify the cost function to penalize mistakes three times more (penalty of 30) if the gold standard has a tag that is not `O` but the candidate tag is `O`.

Deliverables In the writeup, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.

1. Report the precision, recall, and F1 of the model with the full feature set on the dev set after training with early stopping. Try to tune the step size and regularizer strength to get the best F1. (However, we won't be grading you on your final score). Report precision, recall and F1 on the dev set for at least three different step sizes and three different regularizer strengths.
2. Report precision, recall, and F1 for your best model on the test set.
3. How has the modified cost function effected the precision, recall, and F1?
4. Submit the output of your best model for both the dev and test sets.

10 Submission Instructions

Submit a zip file (`A3.zip`) on Canvas, containing the following:

- **Output**: Submit the output of your code on the dev and test sets if it is asked for in each section.
- **Code**: You code should be implemented in Python 3, and needs to be runnable. Submit your code together with a neatly written README file to instruct how to run your code with different settings. We assume that you always follow good practice of coding (commenting, structuring), and these factors are not central to your grade. However, please provide well commented code if you want partial credit. If you have multiple files, provide a short description in the preamble of each file.
- **Report**: As noted above, your writeup should be four pages long, or less, in PDF (one-inch margins, reasonable font sizes). Include your name at the top of the report. Part of the training we aim to give you in this class includes practice with technical writing. Organize your report as neatly as possible, and articulate your thoughts as clearly as possible. We prefer quality over quantity. Do not flood the report with tangential information such as low-level documentation of your code that belongs in code comments or the README. Similarly, when discussing the experimental results, do not copy and paste the entire system output directly to the report. Instead, create tables and figures to organize the experimental results.

References

[1] 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*, pages 142–147, 2003. URL https://www.aclweb.org/anthology/W03-0419.