Natural Language Processing – Wet 1

In the following assignment, you will implement an MEMM (As seen in week 4), in order to train a POS tagger. This will include parsing the input, feature engineering, parameter tuning and evaluation.

We highly recommend using python for the task – however, you can use any programing language you prefer. Notice that due to the nature of the task, we assume that you work with a high level programing language, which should be more flexible when working with data.

The Data:

An explanation on the attached files:

1. train.wtag – Contains 5000 tagged sentences. This file should be used during training.
   The following tagged sentence (As seen in tutorial 3):

   the dog barks -> D N V STOP

   Will appear in the file in the following format:

   the_D dog_N barks_V ._.

   Notice that all sentences (Or most of them) end with a period ('.'), However not every period ends a sentence. The format in which the sentences are tagged is called Penn Treebank. You can read more about it here
2. test.wtag – Contains 1000 tagged sentences, in the same format to the one in train.wtag
3. comp.words – Contains 1000 untagged sentences. The sentences appear in their natural form. For example: "*the dog barks .*"

Training

The parameter estimation of the model will be done using train.wtag. You should create two models:

1. Baseline model, using a set of simple features - $f_{100}, f_{103}, f_{104}$, as defined in
2. Advanced model, using the set of features $f_{100} - f_{105}$, along with other features for numbers, capital letters and more. These are not defined here – you may come up with them yourselves.

In your report, you are requested to specify the feature types that you have used for each model (As described in the lectures), the number of features for each feature type (For example – 217 features of type word+tag). Feature types that were not defined in the lecture (Defined by you) should be explained. Expand on every improvement you have made to the model – including your motivation (For example, dropping features that appeared less). If you chose to use an external library, mention the library name and its purpose.

For each model, write its training time (And a short description of your hardware's specs).

Inference:

Inference should be done with the Viterbi algorithm (As seen in the lecture and tutorial). Mention every changes you have made to the algorithm, with its motivation and seen contribution.

Test:

For each model, use your implemented inference of the test set (test.wtag) and write the prediction accuracy (As seen in the lecture). Write about the differences between the models' performance, and reasons that may have caused those differences.

Create a confusion matrix between the 10 tags for which your model makes the most mistakes, and suggest a way to improve your model(s) to reduce the mistakes between two tags.

For each model, write the time it takes to tag the file (And a short description of your hardware's specs).

Contest:

For each model, run your inference on the file "comp.words" (Which is not tagged). Write the tagged sentences into new files in the wtag file format (As seen in the train and test files). The file names appear in the last part.

For example, if the following sentence appears in the "comp.words" file:

the dog barks .

Run your inference on that sentence. Supposing that the output of your inference algorithm is the sequence of tags "D N V .", write the following line to the output file:

the_D dog_N barks_V ._.

Note that your output files should not contain asterisks (*) or end of sentence tags, and that the order of sentences in the output file should be the same as in the input file (comp.words).

Describe the changes you have made to get your results (Changes in the training, inference etc.)

Also, predict the accuracy you expect to get in the competition, and explain why there may be a difference between your accuracy in comp.words and in the test set. Good explanations may get a bonus.

Allowed External Code:

The standard libraries in your chosen programming language, as well as packages that include optimization of multi-variate functions. For example, the only allowed external libraries in python (Excluding the standard libraries) are numpy and scipy.

During the training, it is recommended to use a package that implements the LBFGS algorithm, as long as it optimizes the loss function and gradients that you provide.

You cannot use packages that:

1. Implement the Viterbi algorithm
2. Implement MEMM
3. Offer text processing (For example, uni/bi/trigram counts)

Submission

Submission in a zip file, with the name **HW1-Wet_123456789_987654321.zip** (Where the students' IDs are 123456789 and 987654321). The file should include:

1. Your report, including:
    a. Your names and IDs
    b. Notes on the training stage (See Training section)
    c. Notes on the Inference stage (See Inference section)
    d. Notes on the Test set (See Test section)
    e. Your summary of the assignment
    f. Explanation of your work in the contest file (See Contest section)
    g. Description of how your workload was divided (Which parter did/executed/implemented what).
2. Your source files. They should be documented and readable. Your code should be able to execute on other machines. Please create a simple interface to recreate the contest files.
3. The contest files – Two files in the wtag format (See Training section), including the words of the contest and their predicted tags (One file for each of the two models). In order to validate that no sentences were dropped, please make sure that after removing your tags, you receive the file "comp.words". Files with missing words will get 0.
    The file names should be: (123456789 is the ID of one of the students)
    a. comp_m1_123456789.wtag – A wtag file inferred by the baseline model.
    b. comp_m2_123456789.wtag - A wtag file inferred by the advanced model.

    Your contest position will be determined by $\max\{accuracy(comp\_m1), accuracy(comp\_m2)\}$ – You will be evaluated by your better model.

    Your contest files should be reproducible – Given your code and the data, the output files should be exactly the same.

    The zip file should look like this:

```
HW1-Wet_123456789_987654321.zip/
    report (.pdf,.docx, etc.)
    Code_Directory/
        ...
    comp_m1_123456789.wtag/
    comp_m2_123456789.wtag/
```

Plagiarism:

Sharing or copying code for this assignment is not allowed, including code from the internet. Your code should only be written by you and can only include the allowed packages.

The Full Set of Features in [(Ratnaparkhi, 96)]

- Word/tag features for all word/tag pairs, e.g.,

$$f_{100}(h,t) = \begin{cases} 1 & \text{if current word } w_i \text{ is } \texttt{base} \text{ and } t = \texttt{Vt} \\ 0 & \text{otherwise} \end{cases}$$

- Spelling features for all prefixes/suffixes of length ≤ 4, e.g.,

$$f_{101}(h,t) = \begin{cases} 1 & \text{if current word } w_i \text{ ends in } \texttt{ing} \text{ and } t = \texttt{VBG} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{102}(h,t) = \begin{cases} 1 & \text{if current word } w_i \text{ starts with } \texttt{pre} \text{ and } t = \texttt{NN} \\ 0 & \text{otherwise} \end{cases}$$

The Full Set of Features in [(Ratnaparkhi, 96)]

- Contextual Features, e.g.,

$$f_{103}(h,t) = \begin{cases} 1 & \text{if } \langle t_{-2}, t_{-1}, t \rangle = \langle \text{DT, JJ, Vt} \rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{104}(h,t) = \begin{cases} 1 & \text{if } \langle t_{-1}, t \rangle = \langle \text{JJ, Vt} \rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{105}(h,t) = \begin{cases} 1 & \text{if } \langle t \rangle = \langle \text{Vt} \rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{106}(h,t) = \begin{cases} 1 & \text{if previous word } w_{i-1} = \textit{the} \text{ and } t = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{107}(h,t) = \begin{cases} 1 & \text{if next word } w_{i+1} = \textit{the} \text{ and } t = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$