Singapore University of Technology and Design  
Information Systems Technology and Design  
50.007 Machine Learning  
  
Design Project

**Group Members**

1. Chew Wei Liang (1000064)
2. Cheah Yew Hong, Kenneth (1000124)
3. Lee Ting’en Kevin (1000139)

**Folder Structure**

|  |  |
| --- | --- |
| **Path** | **Data** |
| hidden-markov | Contains subfolders and a convenience python file that calls upon the classes and functions found inside hidden-markov\bin in order to execute part 2 and part 3 |
| hidden-markov\bin | Contains the majority of the python files, such as raw parser, model calculator, simple POS tagger, Viterbi algorithm, and optimized code |
| hidden-markov\data | Contains the train data, gold standard output and the respective tagged outputs for part 1, part 2 and part 3 |
| hidden-markov\pre\_process | Contains python file that does pre and post-processing of data as part of part 3 improvements |

**Part 1**

Firstly, we parse the training data and output two csv files – eval\_trgdata.csv and model\_params.csv. They contain the emission and transmission counts and parameters respectively. Run the command below while inside the parent directory hidden-markov.

python bin\raw\_parser.py -i data\train -o eval\_trgdata.csv -p model\_params.csv

Secondly, we run the simple POS tagger.

python bin\simple\_pos\_tagger.py -m model\_params.csv -e eval\_trgdata.csv -i data\dev.in -o data\dev.p1.out -g data\dev.out

Score of dev.p1.out: 66.30%

**Part 2**

We run the Viterbi algorithm for part 2 using the convenience python script as follows:

python run\_viterbi\_optimizer.py -p2 -c

To see the details of the algorithm implementation, please look inside the bin folder.

Score of dev.p2.out: 72.63%

**Part 3**

To execute the improved code for part 3, execute the following command:

python run\_viterbi\_optimizer.py -p3 -c

Score of dev.p3.out: 78.64%

We have improved the POS tagger by pre-processing words in both the training and test data. In total, we applied 5 key types of word replacements using python regular expressions in pre\_process\_data.py located inside the pre\_process folder. After tagging, we then post-process the intermediate output to restore the original words which we had replaced earlier. Finally, we obtain dev.p3.out.

Here are the following replacements:

1. All URLs, such as “http://bit.ly”, are replaced with a generic word “http”.
2. All user identifiers, such as “@USER\_44285fcc” or “@USER\_a887f5eb”, are replaced with a generic user identifier “@USER”.
3. All words containing variations of the popular emotive expression, such as “haha” or “hahaha” or “ahahaha”, are replaced with a generic expression “haha”.
4. All words which are purely numbers, such as “31” and “11”, are replaced with a generic number “1000”.
5. All hashtags, such as “#GlareX2Mender” and “#tellme”, are replaced with a generic hashtag “#hashtag”.

Therefore, we can increase the probability of correctly predicting URLs, user identifiers, variations of “haha”, numbers and hashtags, because we have narrowed the probability space by collapsing variations of these words into generic words. This leads to a significant improvement because URLs, user identifiers, variations of “haha”, numbers, and hashtags occur rather frequently when we inspected the training set. Furthermore, we can legally make these replacements because these words are always tagged with the same label.

Due to the fact that we have pre-processed the data, there will be no new words that will be tagged with either “U” or “@” (they would have been replaced with “@USER” and “http” respectively, and will therefore not be ‘new’ words anymore). Hence, we added some code logic inside model\_calc.py to ensure that no *new* words will be tagged with “U” or “@.” Doing this improved our predictions.

The other improvement which we made was to tweak the way we updated the emission parameters for new words. We added +1 to all count(y)s for each new word not found in the training set, before computing the emission parameters. This ensures that the sum of all emission parameters for a particular tag sums to 1, and gives us a higher prediction accuracy.

Finally, we tagged test.in with the following command:

python run\_viterbi\_optimizer.py -p3 -ti data\test.in -to data\test.p3.out

The output files can be located inside the data folder.