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)

Exactly what is each py file and script doing? Is for Part I, II, III? Can we please remove all other irrelevant files for the final submission? I’m very confused.

Can we run through Part I, II, III one more time to make sure we generate the **final** dev.p1.out, dev.p2.out, dev.p3.out, and test.p3.out, and document down the scores?

**Part 1**

TODO

Score of dev.p1.out: ww.ww%

**Part 2**

TODO

Score of dev.p2.out: xx.xx%

**Part 3**

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.

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.

After implementing this change, the technical problem was that new words were being tagged with “@” since the probability was quite high. Therefore, we also amended the original way that we updated our emission parameters within model\_calc.py, <please add in explanation because I’m still not very sure what is happening inside the code.>

The final result is that our accuracy improved by yy.yy%.  
  
  
Score of dev.p3.out: zz.zz%

<Weiliang you mentioned your Viterbi improves it further? Please document how it does so, if that is the case.>

Finally, we have generated the POS tags for test.in and have written the output to test.p3.out. 🡨not actually done yet