We will follow the model of PA 3 and solve a problem using supervised learning, this time Part of Speech Tagging. You will write a program to learn and apply a part of speech taggerl, and another to evaluate your tagger. We'll refer to these programs as **pos-tagger** and **pos-tagger-eval** and they can as usual be in the language of your choice.

The input data for this assignment is found in our Google Drive (PA5-wsj-pos.zip). This unpacks to a directory that contains a file of training data which consists of part of speech tagged text (pos-train.txt), a file of plain text that you will pos tag with your tagger (pos-test.txt), and a file of text which give you the gold standard key of correct part of speech tags for the text to be tagged (pos-key.txt). There is also a README with some information about the data.

The train, test and key data is formatted such that there is one token per line. The tokens in pos-train.txt and pos-key.txt also have POS tags. Note that in general the format of each line in train and key will be WORD/TAG. However, there are a few cases where words have a '/' as a part of the token (eg, 3/4, 1/2-year, or Dutch/Shell). In those cases you'll see that the '/' is escaped which results in the following :

* 3\/4/CD
* 1\/2-year/JJ
* Dutch\/Shell/NNP

Your program **pos-tagger** will need to compute p(tag|word) for every word type in the training data (pos-train.txt). These probabilities will be the backbone of your mode 0 tagger, which you will then augment with rules to create your mode 1 tagger.

Your program **pos-tagger** should operate in two modes, most frequent tag (mode 0) and most frequent tag + enhancements (mode 1). You must provide a command line option so that the user can specify the mode in which tagger-test should be run.

In most frequent tag mode (0), pos-tagger should :

* For each word in test data that is found in the training data, assign it the POS tag with the maximum value of p(tag|word).
* For each word in the test data not found in the training data (i.e., an unknown word), assign it the tag NN.

In most frequent tag + enhanced mode (1), pos-tagger should still carry out mode 0, and then in addition :

* Provide at least 5 manually created rules that improve the handling of unknown words. Rather than always assigning NN, identify cases where NN is not likely to be correct (and provide a rule that does the right thing). Document your rules in the comments to tagger-test. You should tell what each rule is and give a specific example of the kind of problem it will fix. Label the rules in your comments and code as U-1, U-2, U-3, ...
* Provide at least 5 manually created rules that correct errors made by the most frequent tag mode (0) that do not involve unknown words. Document these rules in the comments to tagger-test. You should tell what each rule is and give a specific example of the kind of problem it will fix. Label the rules in your comments and code as E-1, E-2, E-3 ...

**pos-tagger-eval** should compare the output of tagger-test to the gold standard key data (pos-key.txt) and report overall tagging accuracy and also provide information equivalent to what you find in a confusion matrix in the following format :

PREDICT\_TAG KEY\_TAG : count

PREDICT\_TAG is what your system says the tag should be, KEY\_TAG says what the correct gold standard key tag really is according to pos-key.txt, and count is the number of times a word that should have a KEY\_TAG is tagged (incorrectly) as PREDICT\_TAG. List these one per line, and sort on PREDICT\_TAG and then KEY\_TAG in ascending alphabetic order (A B C D ...). Do not display 0 count values.

* For correct classifications report (for example)

DET DET : 10,000

This means that your system (tagger-test) tagged 10,000 word tokens as determiners (DET) that really are determiners.

* For incorrect classifications report (for example)

NN VBD : 5

This means that your system (tagger-test) predicted that 5 word tokens were nouns (NN) when they were really past tense verbs (VBD).

You should run your system in both mode 0 and mode 1 as follows. Note that these examples are Linux specific, and should be adapted to different platforms. However, you should make sure to have just one pos-tagger program that has two different modes (controlled by a command line option) rather than having two separate programs.

# run the tagger in mode 0 and pos tag the test data

* **pos-tagger 0 pos-train.txt pos-test.txt > pos-test-answers-0.txt**

# evaluate your tagger

* **tagger-eval pos-test-key.txt pos-test-0-answers.txt** > **pos-test-0-eval.txt**

# run the tagger in mode 1 and pos tag the test data

* **pos-tagger 1 pos-train.txt pos-test.txt > pos-test-answers-1.txt**

# evaluate your tagger

* **tagger-eval pos-test-key.txt pos-test-1-answers.txt** > **pos-test-1-eval.txt**

Note that your overall accuracy for mode (1) must be an improvement upon mode (0). If it is no improvement then there is a problem with your rules and you should rethink them.

Submit a single pdf file that consists of your source code for pos-tagger and pos-tagger-eval, and then your output files pos-test-0-eval.txt and pos-test-answers-1.txt. Please do not submit screen shots or cut & paste from terminal windows. Instead export your source code to pdf, and print your output to pdf. Your source code should have line numbers. Both source code and output should be on a white background.

Remember that each program must be documented with an overall comment at the start of the program and then detailed comments in the body of the programs. Please make sure the point by point description of the algorithm in your overall comment includes a numbered list of steps for your algorithm, and that these numbers are referred to in your detailed comments. Please see the Programming Assignment Grading Rubric (in our Google Drive) for additional details on documentation.

In terms of functionality these are the things I will be looking for :

* 1 point - mode 0 accuracy >= 90.00 %
* 1 point - 5 rules to improve uknown word handling (U rules)
* 1 point - 5 rules to improve known word errors (E rules)
* 1 point - mode 1 accuracy is greater than mode 0 accuracy by at least 1%. If your mode 0 accuracy is 90.00% then this means your mode 1 accuracy should be >= 91.00%.
* 1 point - pos-tagger-eval program provides accuracy and confusion matrix info in the format described above. .
* You may also lose points for not following the assignment specification, for example not using command line arguments, not implementing mode 0 using p(tag|word), or using NLP or ML code that you did not write yourself.

You may use code from libraries, but do not use any libraries or pre-existing code that are specific to NLP or Machine Learning, in particular those that carry out Part of Speech Tagging or tagger evaluation. Any regular expressions you use for text normalization, pre-processing, tokenization, and sentence boundary detection must be of your own creation.

If you find yourself searching for NLP or Machine Learning specific code, you should stop yourself and allow your own ideas to develop. If you are copying code specific to this assignment that you or a member of your team didn't write, please stop yourself. All of your NLP and Machine Learning code should be original to you and your team. You may re-use your own code from previous assignments (although please indicate in your detailed comments when you do this).