TA's Notes # 1

October 25, 2016

Here's a summary of all the important points we discussed in the session today (if you think I forgot something, please tell me!).

General Homework/Submission Remarks

- Using either Python 2 or 3 is fine by me; I got both installed.
- Your submissions should include all the code I need to verify the correctness/completeness of your program (you may assume that I'm capable of installing common modules e.g. via `pip`).
- For any "discussions", tables, figures etc., you would ideally type up a nice little `.pdf`, e.g. via LaTeX.

The `nltk.model` issue

If you install NLTK like any normal person (e.g. from the main github branch, via pip or by getting it with Anaconda), you will fail miserably at training an `nltk.model.ngram.NgramModel` (as is requested in the second part of the first assignment) because there is no such thing as `nltk.model`. This is because the module was presumably broken and taken out of the main branch until it's fixed, which still hasn't fully happened. Here are some work-arounds to this issue:

1. Implement your own module with all the necessary facilities to generate random text. Do this only if you don't feel challenged and/or have too much time. I'm assuming doing this (correctly) will earn you extra credit (which is ultimately pretty useless I might add, besides impressing your teacher/TA).
2. Install NLTK 2 instead. This is what most of us did last year. I don't know to what extent the module is broken, but generating text should work. Note that you shouldn't be using NLTK 2 for any other task, so you should either keep a separate python installation with NLTK 2 or use something like `virtualenv`. Note: NLTK 2 may require Python 2. The most recent version seems to be 2.0.5, requiring Python 2.5 or higher. Windows users

should use the installer found here: `https://pypi.python.org/pypi/nltk/2.0.5`. Otherwise, `pip install nltk==2.0.5` should suffice.

3. Install the `model` branch of NLTK 3. This can be done as follows: `pip install git+https://github.com/nltk/nltk.git@model`. Note: If you already have NLTK installed, you may need to include an `--upgrade` flag to force reinstallation.

Should you decide to use NLTK 2, the only somewhat "mysterious" part is the `estimator` argument to the `NgramModel` constructor. This needs to be any of the classes (only the name, not an instance!) inheriting from `ProbDistI` in `nltk.probability`. I'd advise you to use `MLEProbDist` since there are issues with some of the more advanced alternatives. With the model created, you can use its `generate()` function to easily produce random word sequences. Read the source code to find out more. This is (unfortunately) a skill you will need to get good at; academic software is usually badly documented. The safest way to find out what the code is going to do is to read the code!

The interface in the NLTK 3 `model` branch seems to be more complex, and it's lacking a `generate()` function so far. You would need to implement this yourself. This means that **the most "sane" solution right now is probably to use NLTK 2.**

Other Remarks on Assignment 1

- You may run into problems when trying to process the Turkish/Bulgarian newspaper texts for part 1. This is because these texts include non-ASCII characters. In Python 3, this should be fixable by passing `encoding='utf-8'` into the `open()` function when opening the file (I think I said today that Python 3 will automatically pick the right encoding. I don't think this is true. Better to make sure anyway!). In Python 2, you will need to use `codecs.open()` instead because the default `open()` does not accept an `encoding` argument.

- You will likely find `nltk.tokenize.word_tokenize()` to be helpful. This function tokenizes a text, i.e. takes a string (which should be a "proper" text with words and such) and returns a list of strings, where the entries are the words and punctuation in the text. There's also `wordpunct_tokenize()`. The only difference I know of is that, for example ``` ``It's'' ``` will be tokenized to ``` [``It'', ``'s''] ``` using the former, and ``` [``It'', ``', ``s''] ``` using the latter. I would prefer `word_tokenize` myself.