Homework 1

1. Comparison of libraries for tokenizing, stemming, and POS tagging:

I utilized two different NLP libraries using python: NLTK and SPACY. NLTK had the capability for all three parsing techniques, but SPACY only had the ability to use tokenizing and POS tagging. I did not parallelize for either of these NLP libraries because the text corpus was relatively small, both libraries were efficient in their calculations, and the deepdish server had a large amount of RAM necessary to complete the job in a timely manner.

When looking at tokenizing, both libraries performed equally well, with NLTK performing a little better in terms of computational efficiency (ten seconds faster than SPACY to perform word and sentence tokenizing). Coding wise, there were more steps needed to tokenize in SPACY than in NLTK, and I had to write a few lines of code to parse the results for SPACY tokenizing than I did for NLTK. Therefore, I would prefer NLTK for tokenizing. SPACY did not have stemming available, so I didn't have a good way to compare the two. If given more time, I would have tried stemming on a third library.

However, in POS tagging, I was much more impressed with SPACY overall than with NLTK. SPACY performed POS tagging in two minutes, while NLTK took over 7 minutes to perform POS tagging. Additionally, I found that the format of SPACY POS tags was far more easily understandable than that returned by NLTK.

2.1:

Regular expression for emails: r'[\w\.-]+@[\w-]+\.[\w\.-]+'

Found 115,613 emails in the whole text corpus. Examined the structure of the first few emails and the regular expression seemed to capture all permutations of emails.

2.2:

Regular expressions for dates:

'(Jan|Feb|Mar|Apr|May|Jun|July|Aug|Sept|Oct|Nov|Dec)\s(\d\d?).+?(\d\d\d\d)' to capture shorthand month first

'(January|February|March|April|May|June|July|August|September|October|November|December)\s(\d\d?).+?(\d\d\d\d)' to capture longhand month first

'(\d\d?)\s(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sept|Oct|Nov|Dec),?\s+?(\d\d\d\d)' to capture shorthand month second

'(\d\d?)\s(January|February|March|April|May|June|July|August|September|October|November|December),?\s+?(\d\d\d\d)' to capture longhand month second

Found 17,752 dates in the entire text corpus. Examined the structure of the first few dates and the regular expressions seemed to capture all permutations of the dates.

3. Summary of Maximum Entropy Paper

In order for a mathematical model to parse the structure of a sentence, one must have a set of rules that justify the frequency and importance of certain common sentence structures. This occurs because inherently, words do not exist in isolation (the common "bag of words" interpretation of a large corpus of text), and when a single word appears in a sentence in a certain context, it increases or decreases the probability that another word will appear in a sentence in the same or different context. Therefore, if one is to calculate a probability that a specific sentence will appear in a specific structure, or parse, one has to process the sentence one word at a time. Generally, this is done in a top down, or generative fashion, where the probability of a sentence is built off of the probability of the head of the sentence. Then, the probability of the second word is conditioned off the

probability of the first word and any other context or history about the word, and so on and so forth. Additionally, it is possible to use a training corpus, or large collection of already parsed sentences, to gather other context that could inform the further structure of the sentence based on the first few words.

A maximum entropy model to parse sentences in this manner chooses a sentence structure from the previously parsed sentences that has the most uniform probability distribution (i.e. the parse that has the maximum entropy). The training data, in this case, is often composed of features that describe different words or phrases with regards to their context within a sentence (i.e. part of speech, place in the sentence, function in the sentence). This creates a grammatical ruleset with associated probabilities that is essential to predicting the probability of a given parse. However, one can utilize this ruleset in two ways. One can create a fixed probability of a parse using previously observed parses, or trees. Or, one can assign a probability of some possible expansion of a parse, not observed in the training set, using fixed probabilities of previously observed parses. This is referred to as Markov grammar. Models that utilize Markov grammar generally perform better than using simply using the tree bank.