CS51 Project: Part-of-Speech Tagging and Highlighter Draft Specification

Team

- Billy Janitsch, janitsch@college.harvard.edu
- Jenny Liu, jennyliu@college.harvard.edu
- Yuechen Zhao, yuechenzhao@college.harvard.edu
- Shijie Zheng, shijiezheng@college.harvard.edu

Brief Overview

At the moment, we lack a useful tool to visualize English language structure, which impedes English language-learning, especially with regards to parts of speech in context. Consequently, we aim to produce produce a text editor which, when given an English text, highlights its linguistic syntax in an aesthetically pleasing way in order to aid in the visualization of structure. In particular, this will involve tagging inputted text with parts of speech, both at a basic level--for words which can take on only one role--and at a higher level--for ambiguous words where context needs to be taken into consideration.

This text editor could be further developed to include other linguistic features such as dictionary definitions and thesaurus suggestions, in addition to highlighting-as-you-type.

Our goals for this project include: creating a useful tool for English Language Learners to visualize language structure; exploring Hidden Markov Models, the Viterbi algorithm, and Java GUI; learning how to code collaboratively; getting comfortable with coding without guidance and instructions; and learning how to work with large datasets of tens-of-thousands of words.

Feature List

Core features:

Text editor: As a core feature, we would need to implement a very basic text editor. The minimum required functions for the editor should be the ability to edit/view text, as well as a button to click which would bring up syntax-highlighted text in a separate window. More features that we would like to see in our text editor but are not absolutely necessary are included in the "Cool Extensions" below.

Training ability: While we will train our dictionary and hidden Markov Model using the tagged version C of the Brown corpus (found at found at http://nltk.googlecode.com/svn/trunk/nltk_data/index.xml), we want our project to have the ability to adjust to different corpuses. In particular, given a text file with a list of tags, and a folder containing the text files of the training corpus (with parts of speech pre-tagged), our project should be able to compute the frequencies of each part of speech for each word, the transition probabilities (the frequencies of adjacent parts of speech, .e. the likelihood a noun appears after a verb), and the emission probabilities (the likelihood that given part of speech is a particular word). It should also be able to save these probabilities to a text file, which when the project is opened from a regular state, it can read in without needing to re-compute probabilities.

Text parsing: Since part-of-speech tagging occurs on the sentence level in order to account for context, our program will be be able to split text into sentences, and then into word tokens.

Part-of-speech recognition, base case (dictionary & search): We will implement a dictionary (more details below) which maps words to sets, where each set contains the parts of speech that the given word can take on. This is the base case in the sense of words which only have one interpretation, e.g. "panda" can only be a noun: "The panda ate the leaves on the bamboo."

Basic syntax highlighting: Our text editor will have a button which causes syntax highlighting based on the part of speech of words. In the basic implementation, this will only occur/be analyzed at the button click, with the highlighted text popping up in a separate window.

Part-of-speech recognition, harder case (HMM & Viterbi algorithm): These words have multiple interpretations, e.g. "run" can be both a verb "I run everyday," or a noun, "I went for a run." For these words, we need to look at it in context with the words around it, evaluating the probability that it will be any one of its possible parts of speech, then choosing the part of speech with the highest probability for that word, in that specific context. The actual algorithm is described at (http://www.inf.ed.ac.uk/teaching/courses/inf2a/slides/2011_inf2a_L15_slides.pdf). This Viterbi algorithm is an example of a use of the Hidden Markov Model (HMM), described at (http://sites.stat.psu.edu/~jiali/course/stat597e/notes2/hmm.pdf).

Cool extensions:

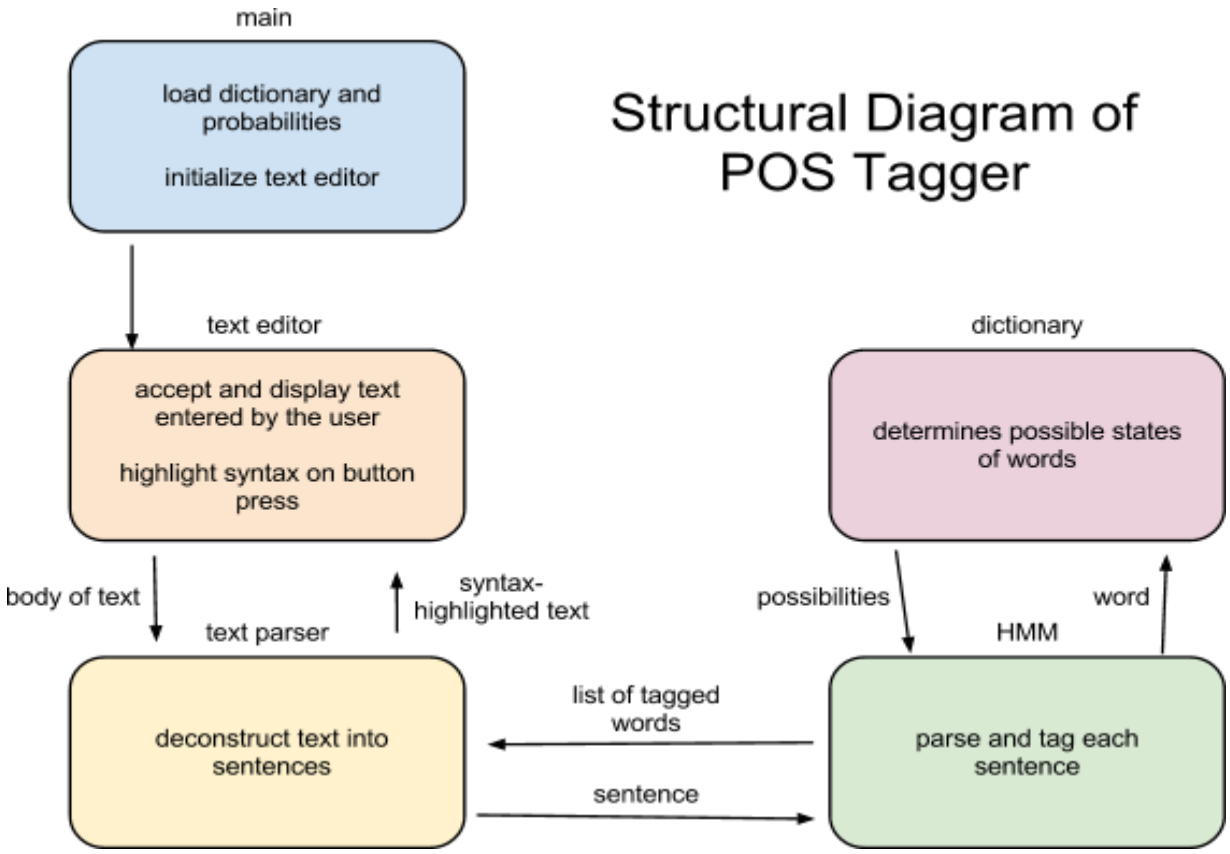
Syntax highlighting as you type: Syntax highlighting on-the-fly. This will most likely check, based on a set time interval, if user has completed a new sentence or updated an old sentence in the text and highlight or re-highlight as needed.

More advanced text editor functions: The ability to save/load text files. Additionally, the ability to have user copy/paste text from within the editor without having to re-analyze the entirety of the copied text when it is inserted again.

Thesaurus feature: Given a word, gives its synonyms and antonyms as demanded by user. This would be built into the dictionary, with the dictionary loaded from a user-specified file..

Dictionary definition lookup: Given a word, gives its definitions as demanded by user. This would be built into the dictionary, with the thesaurus loaded from from a user-specified file.

Draft Technical Specification



Modules (Classes, Interfaces, Functions, etc.)

public static void main ( ) : will call functions in the HMM, dictionary, and text editor in order to load/initialize them. When running the program with an extra flag, it will instead ask/accept file names with which to train/ compute the dictionary and HMM, and call the train functions of those modules.

Text editor: will be created using the java.swing class; basic text editor will consist of a window, an input box, and a button for syntax-highlighting. For cool extensions, menus, e.g. File, Edit, Help.

- Public functions:
    - static void init ( )
- Private
    - Event listeners for button clicks which will call the text parser and pop up a separate window.
    - For cool extensions, event listeners also for typing, copy/paste, etc.

Text parser: given text from the text editor, will break up the text into sentences and pass those sentences to the HMM/Viterbi, and then passes back pairs of words and parts of speech to the text editor. In general, this will involve splitting the text at periods, while taking into account situations such as "Mr. Smith" to avoid splitting at those points.

- Public functions:
    - public static List<Pair(word, POS)> parse(String text)

HMM/Viterbi Part-of-Speech Analysis: given a sentence from the text parser, will break up the sentence into a list of words and perform Viterbi analysis on that list of words by doing dictionary lookups and calculating the necessary probabilities and comparing them; on a general level, this will require populating a two-dimensional array of probabilities using dynamic programming. It will then pass back to the text parser the list of words and their parts of speech.

- Public functions:
    - public static void train (String input)
    - public HMM init (string file) (constructor)
    - public List<Pair(word, POS)> parse (String sentence)

Dictionary: will be implemented as a hashmap (unless that's not feasible with our data set), perhaps using Java's built-in or other 3rd party library.

- Public functions:
    - public static void calculate (String input)
    - public Dictionary init (string file) (constructor)
    - public POS[ ] lookup (String word)

Environment & Setup:

Java: Java is a well-supported and recognized language, with relatively simple GUI creation with the Java

Swing class. In addition, it is cross-platform, which allows our final project to run in any environment, which will facilitate collaboration and will also help it reach the largest audience.

Github: Github is a very popular git hosting service, so it will be reliable for our purposes. In addition, git is a popular VCS that could be installed on many platforms and has been used in class, so it is most familiar. We hope to use it to keep track of our code.

"What is next?"

- Set up Github / (if necessary) install Java
- Explore built-in Java libraries and/or third-party Java libraries that might be helpful in creating our final project, (e.g. the Swing, HashMap, etc.)
- Set up function prototypes/class interfaces
- Work out the major private functions in each module to help facilitate dividing up the coding work later