Final Report: Part-of-Speech Tagging and Highlighting

Team

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

Overview

We implemented a Part-of-Speech (POS) Tagger in a text-editor GUI with the capability of color-coding words based on their parts of speech.

Users can type text input directly into the text-box, paste text into the text-box, or open a text file to be tagged. The output of colored text appears below, maintaining the same format as the input. Users can also click on specific words to see their definition and synonyms in the Dictionary and Thesaurus boxes on the right side of the input text box.

At a general level, our algorithm splits text into sentences and then words, and thens, in the Viterbi algorithm, used probabilities from sample data to compute which part of speech each word should be, based on highest probability. While we included a default set of compiled probabilities using the Brown Corpus (a set of 500 text files with words tagged with parts of speech), we also included the functionality for the user to use his/her own pre-tagged datasets or part of speech lists, opening the possibility of usage for more specific areas or other languages. Additionally, we included code for testing the accuracy of our algorithm against a pre-tagged dataset.

After finishing our algorithm, we trained our POS tagger on half of the Brown corpus and then ran the algorithm on the other half, obtaining approximately 84% accuracy: 491,985 of 585,547 words in the half of the corpus we ran the test on were tagged correctly.[1]

Planning

By the first checkpoint, we were able to work out the separate parts so that the code compiled and much of the background code for the POS class was finished. At this point, the TextParser could split blocks of text into sentences and then into words and punctuation, while the Viterbi class could read in a corpus, compute probabilities, and write a data file. Generally speaking, this was about where we expected to be at

[1] Our actual accuracy would be slightly lower; in particular, as discussed later, we ran into the problem of how to break up sentences given that abbreviations also used periods. Thus, for the sake of testing, we did not run the text though the sentence parser, but instead directly through the Viterbi algorithm; we chose to do this because we could only do sequential comparison of parts of speech, so that if the sentence tagger had tagged "Dr." as two words, "Dr" and ".", while the Brown corpus had it as one, then all comparisons afterwards would have been offset and therefore inaccurate.

this point. Next steps were to complete as much of the code base as possible.

By the second checkpoint, our code was able to tag text with reasonable accuracy with a completed Viterbi algorithm. The physical structure of the GUI was also completed. The TextParser was refined to include whitespace logically with the words/punctuation to facilitate the reconstruction of sentences from tagged data passed back to the GUI. While we originally planned for dealing with error handling at this point, it turned out that would be better handled with a fully or almost fully integrated GUI.

In the last segment, we finished the integration of the GUI with the rest of the code. We continued making refinements to the algorithm, as well as working out a consistent scheme of handling errors and exceptions (previously, we had terminated the program upon errors, but now we wanted to kick errors back up to the GUI to prompt for user responses). Here, we also built the dictionary and thesaurus functions, which had originally been a part of our extensions list.

Design and implementation[2]

Keeping to our implementation was difficult, particularly in terms of trying to maintain abstraction. In particular, we wanted our individual classes to be individually reusable; on the other hand, making the program run required loading several files in different places, which could cause problems if the files were not compatible (i.e. if they tried to reference different tagset). As a result, we had to spend some amount of time cross-checking when we loaded files to make sure that everything lined up. While we tried to be very specific in the javadoc on how to format files, we worried about user-generated errors on this front.

In our original design, we had fewer files demanded from the user; however, as programming went on, we realized that we needed more inputs for the program to be usable beyond the default dataset which we provided. This stemmed mostly from our experiments with data structures to use for storing probabilities. Originally, we had planned to figure out the number of parts of speech on the fly. However, we realized that this would not be very practical, since it meant that instead of having hashmaps of words to arrays of probabilities (by part of speech), we would instead have to have hashmaps of arraylists, or hashmaps of hashmaps. Additionally, we ran into indexing problems with our parts of speech using this type of implementation. As a result, we instead asked the user for a list in advance of getting the corpus, simplifying our work dramatically.

In our original design, we packaged the probabilities for the Viterbi algorithm together with dictionary definitions. However, after some thought, we realized that the two parts were essentially independent, and that we were duplicating code in an attempt to include probabilities there instead of in the Viterbi algorithm proper. Consequently, after the first week we took out our Dictionary class entirely, only replacing it at the end by a Dictionary whose only function was to look up words.

Also, we originally tried to ignore abbreviations (i.e. “Mr.”, “Sgt.”, etc.) during regex parsing, but this proved to be very difficult to accomplish efficiently.

[2] See also annotated functional specifications in the reports folder. The original tech/draft specs are also included there.

Reflection

GitHub was extremely easy to use and really helped us keep track of code and changes across four different computers. The Java libraries were also extremely helpful and significantly reduced the amount of work that we had to do to construct data structures necessary to complete this task. Overall, we were very satisfied with our language choice.

We did not realize the extent to which different code editing programs treated tabs and spaces differently, so when files were shared between different computers, the code looked different in TextWrangler as opposed to Emacs and became messy. It probably would have been a good idea to agree on one development environment to avoid that issue.

In order to make this a feasible project, we chose the one-level Viterbi algorithm, which only considered one tag before the current tag, as opposed to multiple tags, which might have increased our accuracy. Though this was not optimal, it was a necessity to make the project doable in such a short amount of time. In addition, we chose to assign parts of speech, when given an unknown word, based on its position in a sentence, as opposed to just assigning "proper noun." This will, in some cases, provide better results; it also gave us the flexibility to transition between tagsets. Finally, we chose to bundle the Brown Corpus with our POSTagger because it seems ubiquitous in the field. We could have chosen another, but it would not have affected our algorithm at all.

It we had more time, we really would have liked to make the part of speech tagging occur on-the-fly, without the clunky interface of forcing the user to click a button every time. We would also explore into creating more corner cases in the text parser and the Viterbi algorithm so that we achieve greater accuracy.

The most important thing we learned from this project was that in order to work effectively, constant communication between group members is key! It would probably also be a good idea for us to meet more often to code together, so that we always knew what the progress was like on this project. This will help us greatly in group projects in the future.

Advice for future students

Start early, meet at least weekly, work steadily, and keep in constant communication with your group members!

Also, while we had a fairly detailed set of interfaces mapped out by the time we got to design specs, we realized that we also needed to be a lot more explicit about it all to make sure that we were on the same page. Always be more detailed with your specifications than you think they need to be!

We were lucky that we were able to choose such an interesting algorithm and topic. This project will most likely consume your life, so make sure it's something you want to be working on!

Task Division

- Billy
    - Authored TextEditor.form. Designed the GUI interface.
    - Authored TextEditor.java. Designed and implemented the main GUI classes for handling user actions. Includes functionality such as typing, POS tagging, load/save files, etc.
    - Co-Authored POSTaggerApp.java. Constructed part of the main class to launch the GUI to handle user-program interactions.
- Jenny
    - Authored TextParser.java. Designed and implemented the algorithm for taking in a block of text as a string and dividing the text into lists of sentences, and then lists of words and punctuation while logically retaining whitespace to preserve formatting.
    - Tested integration of Viterbi and TextParser using POSTaggerApp.java.
    - Proposed initial project idea.
    - Wrote script for Demo Video.
- Yuechen
    - Authored Dictionary.java. Designed and implemented reading in and indexing all the terms and definitions from webster_dictionary.txt to facilitate the look-up of terms.
    - Co-Authored Exceptions.java. Constructed the exceptions classes we needed to do reasonable error handling.
    - Authored POS.java. Designed and implemented reading from tagsets and kept track of all the POS that could be represented in the corpus, in addition to what they simplified to.
    - Co-Authored testing the accuracy of the Viterbi algorithm: reading in the words from the corpus, tagging with the part of speech, and comparing that POS with real POS.
    - Co-Authored Viterbi.java. Designed and implemented the reading of corpus files and calculating the the frequencies needed in order to calculate probabilities for Viterbi.
    - Corrected errors in the version of the Brown Corpus obtained. Constructed default corpus_tagset and corpus_simple_tagset Modified webster_dictionary.txt.
- Joy
    - Co-Authored Exceptions.java. (see description above)
    - Co-Authored testing the accuracy of the Viterbi algorithm. (see description above)
    - Authored Pair.java. Designed and implemented the pair class we used to connect words with part of speech (POS) objects
    - Authored Thesaurus.java. Designed and implemented reading in and indexing all the terms and synonyms from the thesaurus file to facilitate the look-up of terms. Edited thesaurus.txt from the formatting of the original file to work with our algorithm.
    - Co-Authored Viterbi.java. Designed and implemented calculating the probabilities necessary for Viterbi to run correctly, such as the probability that any part of speech follows another part of speech, which is written into datafile.txt. Implemented the actual Viterbi algorithm, which tagged each word in sentence provided with the most likely POS.