Project Report

Introduction

The purpose of this project was to investigate whether a probabilistic context-free grammar is suitable for modeling a machine interpreted language. We began by looking at programming languages (Java, Python, and Tiny C) to see if we could learn the grammar rules and generate new programs. Although it would be relatively easy to determine whether the generated source code was grammatically correct (by attempting to compiling it), we realized that it would be difficult to evaluate the semantics of the generated program to determine if it did anything meaningful.

Ultimately, we decided to model HTML because we thought it would be more interesting to generate webpages, and evaluation is as simple as loading the generated document into any browser and passing judgment. While we were able to effectively model the HTML grammar, we discovered that modeling the webpage content was more difficult than we had first guessed.

An Empirical Approach

Due to the fact that webpages often do not follow widely accepted standards, HTML has many problems and complexities that can be difficult to properly parse. Most pre-existing projects aimed at traversing the HTML DOM (Document Object Model) tree would handle more edge cases than anything we could create in the limited time frame, so we decided the best approach was to use an existing toolkit to parse the HTML documents in our corpus. We chose a Python module called Beautiful Soup [1], which builds a complex tree of tags and allows us to prune any tags that our we explicitly do not support.

The World Wide Web Consortium's official definition of the HTML grammar [2] is very large, so it would be difficult (if not impossible) to build a corpus that includes pages that use all the tags. Instead of predefining the HTML grammar and learning just the transition rule probabilities, we decided to empirically learn both the grammar rules and the probabilities from the webpages in our corpus. This allows us to better model how HTML is actually used, rather than how it is meant to be used. Another advantage is that all the tags found in the corpus are implicitly supported (as long as aren't explicitly removed during preprocessing).

Corpus Content

We have tried a few different approaches on the corpus, from a few small test cases, to a huge 3GB version based on some popular sites. There is a big trade-off between finding sites that would be ideal for our program (i.e., primarily use the tags we support) and collecting a large amount of data. Since we did not support CSS files, which are typically either referenced with the <script> tag or embedded with the <style> tag, we had a problem collecting the large amount of data typically expected in corpora. Most modern websites style their pages with CSS, and excluding this code during preprocessing forced us to balance a stylized corpus and a larger modern corpus.

Program Design

We developed this project using both Python and Perl, and the only external dependency is a Python module for HTML parsing called Beautiful Soup. The program suite is divided into three key phases: Parsing, Generation, and Surface Rendering. Each phase acts as a separate program that builds on results of the earlier phase, so the input from one phase is the output from the previous phase. The following is the basic workflow of the three program phases:

Parsing → Generation → Surface Rendering

The Parsing phase takes the corpus as input and produces tag and attribute transition maps as output. It preprocesses HTML documents with Beautiful Soup to build a tag tree then traverses it depth-first, recording each transition from a parent tag to the children tags as an observation of a grammar rule. The transition rule includes both the terminal symbols (the actual tags, e.g., <body> and </body>) as they are found in the HTML document as well as the non-terminal symbols for tags that can be children of the current tag. According to the W3C's standard, some tags cannot have children (such as
), but since we do not strictly enforce this, it could be learned empirically from the content of the corpus.

The Generation phase takes the tag and attribute transition maps as input and generates a new HTML document as output. It starts a new document with the HTML tag at its root (the non-terminal starting symbol) and stochastically chooses transition rules according to the transition probabilities learned from the corpus. It expands each non-terminal symbol in a depth-first manner until the document contains only terminal symbols and the special TEXT symbol in place of actual text elements. The end result is an HTML document that lacks surface rendering but can be loaded into a browser to inspect the structure of the document. Note that the generated HTML document is guaranteed to comply with HTML web standards if every page in the corpus is compliant.

The Surface Rendering phase takes the generated HTML document as input and makes final corrections. These include replacing the special TEXT symbol with generated text and ensuring external HTML documents and image files referenced by anchor and image tags are properly linked whenever possible. Lastly, we evaluate the final result by viewing the generated document in a web browser.

Results

Our first version of the grammar-learning program did not support attributes. We removed a few tags that we knew would not function properly without attributes (images, applets, scripts, etc.) and learned just the tag transitions from a small corpus. The generated HTML documents were legitimate, but not very interesting due to the restrictions on the grammar. Next, we adapted our programs to handle attributes using a separate transition map, which means our program essentially learns two CFGs: the tag-to-tag transition grammar and the tag-to-attribute transition grammar. To keep it as general as possible, we also learned the probabilities of each attribute taking particular values. With this additional functionality, we retrained the grammar from the same corpus and got significantly more interesting results, which included colors, images, and links to other webpages.

Our presentation in class demonstrated results modeled from a small corpus of the "Mr. X" [3] and "Darwin Awards" [4] websites. These results often included deeply nested tables and used vibrant colors. The general look of the pages was poor due to inconsistent formatting, long sentences as links, and images strewn throughout paragraphs. Later, we modeled 3 GB of data from popular websites and found the generated documents to be banal. The structure was often linear with very shallow structuring, and tags were used in excess because most modern websites define the page style in external CSS files. Colors and layouts were nonexistent, thus generating linear pages that lacked any style.

Some example results can be viewed from a special page on Jason's website [5].

Conclusion

We accomplished our intended goal of empirically modeling HTML grammar using a probabilistic CFG, allowing us to generate new webpages that follow the same rules as the pages in the corpus. However, the generated documents are anything but stylistically impressive and would not convince most people that they are created by a human with any aesthetic sensibilities or experience with webpage design. The underlying cause of this problem is that modern webpage design relies heavily on context-sensitive design, using vastly different layouts and styles for the content and navigation areas of the page. Our grammar-learning program is not able to account for this context-sensitivity without adding heuristic rules, and so it fails to capture the true organization and design of the webpages it learns from. Modeling HTML using a context-sensitive grammar would be more theoretically sound, but typically the rules in such a grammar are engineered for specific purposes, so it would be difficult (if not impossible) to learn such rules empirically.

References

[1] Leonard Richardson, Beautiful Soup, 1996-2009 <http://www.crummy.com/software/BeautifulSoup/>.

[2] World Wide Web Consortium (W3C), HTML 4.01 Specifications, 1997-1999 <http://www.w3.org/TR/REC-html40/sgml/intro.html>.

[3] Fox Broadcasting Company, Mr. X's Webpage, 2009 <http://www.mrxswebpage.com/>.

[4] Anonymous, Darwin Awards, 1994-2009 <http://www.darwinawards.com/>.

[5] Jason Switzer, HTML Generation Project Results, 2009 <http://s1n.dyndns.org/ones_and_zeros/results.html>.