Alice Wong and Cecille Yang
CS 349 - 01
Final Project Paper

Generating Dating Profiles

Introduction

Sometimes, you can't get enough of a character and sometimes your friend and you can't settle on who should get the girl/guy. Wouldn't it be nice if Edward Cullen had an OkCupid account, or if you could swipe right on Black Widow? You could find out what movie characters do in their free time, and read about them in first person text, written the way they speak! What started out as a late-night conversation grew into an idea for a project-- maybe, with a little help from natural language processing techniques, this could become reality.

Background

For the background of our project, we've looked into both software and research papers for inspiration in creating our project. One piece of software we looked at was an IntelliJ plugin that generates texts based on categories such as culinary inspiration, esoteric wisdom and science fiction. For the most part though, research papers served as better sources of inspiration.

A paper we focused on for inspiration to the solution of text generation was Trainable Methods for Surface Natural Language Generation. The paper described methods for surface natural language generation - one that generates text based on most frequent phrases, and two that generate individual words in a phrase with calculations for word order and choice. The author describes a technique to build a dependency tree for determining word order and calculating probabilities for each child in the tree.

The other work that was a motivation for us to pursue our project in the movie processing direction was Learning Latent Personas of Film Characters . The paper takes plot summaries with implicit descriptions of the characters and parses the text to determine linguistic features for each character in order to determine the character's "latent personality", or their role in a film. For example, a villain may perform a lot of "assault"-type actions, like strangling or hitting or kidnapping. These words then tie in the character closer to the villain persona cluster. As long as there are words describing actions in a persona, the persona's identity can vary-- instead of boring categories like "villain" or "hero", the authors were also able to partially produce an accurate clustering of characters on TV Trope roles like “The Jerk Jock” or “The Klutz”.

Process

We decided early on that we wanted to construct a text generator for creating dating profiles, but we weren’t sure where to pull data-- Twitter has a limit for amount of data we can pull every 15 minutes, Facebook security would greatly limit what text we could get off of it, and we weren’t very familiar with any other APIs that could potentially help with the project. We briefly also thought about using Project Gutenberg, as there was a large amount of free fictional works written in first person on there. However, the process of getting a text file to parse and clean would be in itself a challenge. Thus, we ended up deciding to use Cornell’s movie dialog corpus because it is clean, organized, and fulfills the need for first-person, in-character texts quite well.

We began by playing around with NLTK’s part of speech classification tool by writing a parser that went movie by movie, character by character, and line by line, categorizing each movie character’s words Cornell’s movie screenplay corpus. The runtime of the parser was quite long, and we had the printouts saved to a json file so that the procedure only had to be run once.

After discussing whether generating profiles for all of the characters in all of the movies in the database was truly necessary (our parsing took more than a week to run, and the hmm would then take longer), we decided to focus on one movie instead. Because we had both seen 10 Things I Hate About You, we picked that as our corpus.
With the dictionary of word frequencies categorized by part of speech, we were able to “randomly” select a word from a part of speech with probability weighted towards words that showed up more frequently in that part of speech. The results were fill-in-the-blank answers that made sense in terms of the movie, but were not quite grammatically correct. With some success in terms of single-word selection for characters, we decided to move on to sentence generation.

As Professor Reddy warned, sentence generation was not an easy task. While n-gram models could form sentences that were roughly correct, there would be much less variation to the sentences, and words would be selected in a much less random fashion. For more flexibility, we decided to use a Hidden Markov Model (hmm) to “learn” the speech patterns of each character. To do so, we parsed through the lines, collecting them in a dictionary of characters and their lines,

broken down into tuples of the word and its part of speech. From there, we built transition and emit files by going through the lines and keeping track of word emissions and transitions and normalizing the counts.
One difficulty in using the Hidden Markov Model with movie characters that we did not anticipate, though, was the low number of lines some characters had. Because the character only spoke for the duration of the movie (and some side characters had barely any lines), the training data was not the most desirable to work with. Many of the parts of speech in Penn Treebank's part of speech tag set had no transitions to each other, and the tag set was much more involved than the previous parts of speech tag set that we worked with.

To counter the zero counts in transitions, used add-sigma smoothing. To start off with, we added one to every value in the transitions file before normalizing the data. The resulting generated sentences were mostly nonsensical, and we had no way of telling which character said what. After some reflection, we realized that one was too big of a number to add, since our corpus for each character was not very large to begin with. Thus, we ran the hmm builder again, with .1 as the smoothing factor. The results looked slightly better, with less random punctuation all around.

To use the program, run the command "python parseCorpus.py". Afterward, the dating profiles are located in data/datingprofiles.

Analysis/ Future Work
The current state of the project is clearly not as developed as it could be. For the most part, the sentences don't make sense, especially not strung together. The sentences also have no context and aren't specifically about what the character is into in terms of dating. In the future, we could work to make the sentences make more sense from an English standpoint. There are various other smoothing techniques that we could try which could possibly make the sentences more grammatical in English. Also, NLTK tags some words like "i" to be an adjective but in English, it is rarely used as an adjective if at all. So trying to resolve some of those issues would be helpful to getting more sensible sentences.

Also, it would be interesting to have characters be able to answer questions-- have the program be able to interpret the question and then formulate the response. We had originally thought about doing this, but the scope of the project would have been too large for the amount of time we had.

Another idea we had was to create a pleasing UI and maybe create a fake dating site with all of the profiles formatted to look like an actual platform that the characters are using. This would be less NLP-heavy and more focused around web programming.

Finally, it would be interesting to take this concept and actually implement it with fictional characters. If we run the program on cleaned corpi of Project Gutenberg texts, we would have a larger vocabulary to work with in building emissions and transitions files for the HMM. Hopefully, this would provide a better variety of sentences.