Project 3

Due: 3 May 2011

For this project you will build a babble-bot that can generate text in the style of any author whose work you feed it; then, you'll turn it around and use it to evaluate the work of an unknown author to try to decide whose work it might be. Again, keep a lab notebook with notes about what you tried and what did and didn't work.

1 The text

You've already written a program to read in text, preprocess it, and store some basic probability information about it. Use that.

For the text itself, you can start with *The Vicar of Wakefield* but this will be more fun if you have other authors (and more data). Project Gutenberg is great for this sort of thing, but you can draw from other sources as well. If you find something good, put it in the shared directory (and reserve a portion of it for testing later).

2 The babbler

A babble-bot is a way of using a generative probability model to actually generate output; at any given point you can look at what has already been said (by the bot!), and evaluate the probabilities of all possible next words. Then, pick a word at random from that weighted distribution.

3 The author-evaluator

This part is just a matter of turning the babble-bot around—rather than actually generating text, you look at text that someone else has generated, *imagine* that they are nothing but a babble-bot themselves, and try to decide which author's text they were trained on by evaluating the probabilities of what they wrote according to each model.

To accomplish this, you'll need two separate batches of training text, and you'll need to keep their probability tables separate.

There are two technical difficulties with doing this that you can't avoid: unknown words and underflow. For the former, if in the testing you find a word that this author has never used (in your training), this shouldn't zero out the probability; just assign it some low, constant probability like 10^{-8}.

Underflow is only slightly harder to solve. The problem is that if each word has some low probability of occurring, like 10^{-2} or 10^{-4}, and you're multiplying these together, you get a number smaller than the smallest `double` that you can represent before the end of the first sentence. The solution is to note that multiplying probabilities is just like adding log probabilities:

$$p(\text{this text}) = p(\text{first word}) \cdot p(\text{second word}) \cdot p(\text{third word}) \ldots$$

is equivalent to

$$\log(p(\text{this text})) = \log(p(\text{first word})) + \log(p(\text{second word})) + \log(p(\text{third word})) \ldots$$

So store those instead.

4 Algorithms

Version 0: unigram babbler

To start with, model the language as nothing but pulling words out of a hat, with no context; the only thing affecting the probability of each word is its overall frequency. This takes your probability model from the last homework and just grafts a spinner (weighted random generator) onto it.

Version 1: bigram babbler

More interesting babbling will come from a program that bases its word choice on what it has already uttered. As such, for this version you'll need to build a complete "bigram", i.e. two-word, model—$p(W|P)$, where W is the current word (about to be generated) and P is the previous word (already printed).

Implementation note: you'll have to gather more complicated counts for this; the appropriate data structure (for the counts) will be a `Map<String, Map<String, Integer>>` or your local equivalent.

Version 2: author evaluation

Using precisely the same probabilities as you used in the bigram babble-bot, you can judge the relative likelihood of the text being by a certain author. First, collect the probabilities for two separate authors. Then, if you've just seen the word "not" and need to decide on the word "been", you can look at

$$p(\text{been}|\text{not}, auth_1)$$

and

$$p(\text{been}|\text{not}, auth_2)$$

which you might think of as $p_{a1}(\text{been}|\text{not})$ and $p_{a2}(\text{been}|\text{not})$, because you'll compute them by looking up $p(\text{been}|\text{not})$ first in one lookup table and then in the other.

To compute an "author score" for a text with words $w_1, w_2, w_3, w_4, \ldots$, then, you'd compute

$$p_{a1}(w_1|\text{START}) \cdot p_{a1}(w_2|w_1) \cdot p_{a1}(w_3|w_2) \cdot p_{a1}(w_4|w_3) \cdot \ldots$$

and

$$p_{a2}(w_1|\text{START}) \cdot p_{a2}(w_2|w_1) \cdot p_{a2}(w_3|w_2) \cdot p_{a2}(w_4|w_3) \cdot \ldots$$

and see which has the higher probability. (Theory note: this is not the same as a naïve Bayes classifier, although it's a distant relative; this is actually a Markov model. We'll cover the background theory in more detail later this week.)

Version 3+: more

Like any other 300-level project, this is fundamentally open-ended (with the open ends stretching up into research-land eventually). Do something cool with this! Grammar-based solutions or Hidden Markov models are too much, but converting to a trigram model (basing the probability on the *two* previous words) or improving the unknown word handling should be manageable. Talk to me if you have or are looking for ideas.

5 Handing in

Hand in as `proj3`. As before, *don't* tar or zip the directory, just hand it in. Make sure to include your lab notebook (notes on what did/didn't work) in the handin.