Zac Gross
Jonathan Gemmell
CSC 380
3/15/15

I did the extra implementation of the n-order markov model.

YouTube Link

https://youtu.be/sBwoXMiYK-4

Observations and Conclusions

- I started by implementing a 1st order model with an array. I found it very hard conceptually to translate that into an n-order model. It may have been easier to just start by trying to implement the n-order
- My implementation works easily for up to 3rd order. After that the training takes a long time and starts to run into memory issues. For fourth order, before the training happens, the process uses about 3 mb of ram. Once it has the training data it takes up 1 gb.
- There are a couple solutions I see to this. First, I store the frequency data and normalized data separately. I could change it to overwrite the frequency data with the normalized data. This should cut the memory usage in about half. Second, I'm created an N level deep nested dictionary for all possible letter combinations. I only need data on combinations that occur. For example, I don't need data on the probability of "qqqq" if that sequence never occurs in the training data. Only storing data as it occurs would cut out a lot of empty space in the data. I did a quick analysis of the data and for my 3rd order model, 99.5% of the data is empty. I expect this empty data percentage to increase as the order increases.
- I could also reprogram this in something other than Python isn't designed to work well for lots of recursion which is what my implementation is based on.
- The results of the 1st order model are pretty bad (e.g. "Irlinnnephaily"). The results get a lot better as the order increases. I found 3rd order to be a good balance between run time and result quality. I was actually surprised when it generated some names and I had to double check that they weren't in the training data.
- The training is by far the most time and memory intensive part of this model. I initially had it retrain every time the user wanted more names. Once I split it up to only regenerate off the previously created model, the name generation was nearly instantaneous. It'd be nice to save off the model for a given dataset and order and store it on the disk for later use. that way you'd only have to generate it once and then just read it in again. This would have large time savings.

- I’d like to eventually make this more generic to take in any file with a list of names. I run tabletop roleplaying games and it’d be really nice to generate names for npcs.
- A possible improvement would be the option to add a generated name to the training data. This would help for starting with a small data set.
- Another possible improvement would be to add a mutation amount. This would add some random data to the model to add in sequences that otherwise not be generated. This coupled with adding generated names to the training data could make for a very interesting system that worked like a genetic algorithm.