Group members: Steve Riley
Katherine Overman

In our group project, we aim to find the best topology of hidden Markov model (HMM) to generate new tweets that are representative of a large tweet set. HMM's are currently used for text generation bots on Twitter (e.g., https://twitter.com/markovchainme) and Reddit (e.g., https://www.reddit.com/r/SubredditSimulator/). We will use the tweets of president Donald J. Trump as our data set.

Architecture

HMMs generate sequences from a known finite alphabet or lexicon. Thus, the modeling of tweets is a natural use for HMMs. An HMM contains a number of hidden nodes; each node has a transition probability matrix (to the set of all nodes in the HMM) and an output probability matrix (to the set of all possible atomic outputs). The HMM starts iteration at a specified node and runs through a number of steps. Each step consists of generating an item from the alphabet or lexicon (using the output probability matrix), which is concatenated onto the output sequence, and then transitioning to another node (using the transition probability matrix). The sequence terminates when iteration reaches a specified "end" node, or alternatively when a specified "end" character is output.

Words vs. Characters

There are two natural choices for the set of atomic outputs: 1) characters and 2) words. Each choice has pros and cons. When using words as outputs, the number of possible atomic outputs is much higher; however, the structure of complete outputs is much simpler, and therefore the number of underlying nodes needed would be fewer. One problem that arises when using words as outputs is that words with spelling errors cannot be associated naturally with the correctly spelled version. This is a serious problem in tweets, since spelling errors and shorthand are common. When using characters as tweets, the structure of the model becomes more complicated, but the output matrices are simpler. We choose to generate our tweets character by character rather than word by word. The main reason is because viewing our dataset as collections of words would drop the size of the set to well below what is expected for the class.

HHMMs

An extension of the HMM architecture is the hierarchical hidden Markov model (HHMM). HHMMs have tree structures. The leaf nodes function as normal HMM nodes. Control in higher level nodes passes recursively downward when a node is entered, and when a sequence is generated from the child nodes, control passes back up. The hierarchical structure of language lends itself well to an algorithm that naturally mimics this property. However, every HHMM can be represented as a HMM, so we feel it is not necessary to

explore the complicated topologies and training algorithms of the HHMM.

Data

The set of tweets by Donald Trump measures 34,550 as of 3/2/17. He has been tweeting since 2009 and currently writes about 5 per day. The average number of characters in a tweet is somewhere around 65 to 70, putting the number of total characters in his tweets (i.e., samples times features) above 2 million.

Training the Model

The canonical training algorithm for HMMs is the Baum-Welch algorithm, which is implemented in the Python package ghmm. Given a set of training data, this algorithm converges on a set of output and transition probabilities that are a local likelihood maximum in the parameter space. The global maximum cannot be found in a tractable manner.

Testing Different Architectures

Given a trained HMM, the probability of an observed sequence can be calculated using a forward algorithm. Thus, if we break the data set into training and test sets, the likelihood of the test set given the HMM built around the training set can be calculated. This means that cross-validation or bootstrapping can be used to test a given architecture of hidden nodes. We will generate a mean and a standard error for the likelihood of the test set given the training set.

An untrained HMM architecture is fully specified by the number of hidden nodes in the model. So, we will start with a small number of hidden nodes and increase. For each architecture we will bootstrap the data, train and test the model, and get an estimate of likelihood mean and standard error. We will continue this search until there is a clear maximum likelihood found. This maximum likelihood will correspond to the optimal model. We will possibly use the 1-SE rule.