CMPSCI 240
Reasoning Under Uncertainty
Homework 7

Prof. Hanna Wallach

Assigned: April 13, 2012
Due: April 20, 2012

Overview

The goal of this project is to implement a Markov chain text generator. The code prompts the user for three inputs: (1) the number of tokens (either words or letters) associated with each state, (2) the kind of tokens associated with each state (words or letters), (3) a file containing training data. The code will construct a Markov chain over either words or letters (as specified) using the training data and then generate some new text. You will be writing the code to construct the Markov chain (i.e., states, transition probabilities, etc.) as well as some other methods to help you do this.

Files

The Java files needed for this assignment are on EdLab in the `/courses/cs200/cs240/cs240/hw7/skeleton/` directory.

- `MarkovChain.java`: this is the starting point for the program. This file is the only file you will need to edit for this homework.
- `MCState.java`: An `MCState` object represents an individual state in a Markov chain. It contains a list of tokens (words or letters). You

should familiarize yourself with this code because you'll need to call some of these methods, but you don't need to change anything here.

- `TransitionCounts.java`: This is part of the data structure used to store how many transitions we've seen between pairs of states.
- `TextFileTokenizers.java`: Don't worry about this file.

The code

`MarkovChain.java` will run straight out of the box (but won't do anything intelligent). You need to complete the following methods:

- `train(...)`: trains a text generator from a stream of tokens.
- `addTrainingExample(...)`: updates the generator.
- `getSmoothedProbability(state1, state2)`: returns the probability that `state1` will transition to `state2`. Use "smoothing".
- `generateText(...)` generate random text using the Markov chain.

Data structures

The only not-completely-intuitive data structure in this program is the one we use to store the transitions for the Markov chain. You will not store the transition probabilities directly; you will only keep track of the frequencies of the transitions observed while examining the token stream. Any time you need a transition probability, you will compute it from these stored frequencies (this is what `getSmoothedProbability(...)` will do).

For every state you need in the Markov chain, you must create a corresponding `TransitionCounts` object. All of these `TransitionCounts` objects should be stored in a hash table called `transitions` (this vari-

able is already defined in the code). If you need to do anything relating to manipulating transitions (such as adding a new transition, counting transitions, etc.) it all starts with this `transitions` hash table.

A `TransitionCounts` object is also a hash table. For any state S, the `TransitionCounts` hash table for S stores the frequencies for any transition from S to state T. The `TransitionCounts` table does this by storing a mapping from T to how many times we've seen the transition $S \to T$.

In other words, we have a hash table which stores other hash tables. As an example, to retrieve the number of times we've seen the transition $S \to T$, we would need to retrieve S's `TransitionCounts` object from `transitions`, and then retrieve the integer corresponding to T from that table.

Testing your code

There are various text files in the `data` directory that you can use to test your program. We will post sample output for some of these text files.

There is also a line in the `main()` method in `MarkovChain.java` that will print the transition probability matrix. It is commented out (because the matrices can get really big) but you may uncomment it to see the matrix.

Turning in the program

You should upload your program to your account on the EdLab: create a subdirectory called `hw7` inside your `cs240` directory and upload your `MarkovChain.java` file into it; that's the only file necessary.