CS 4650/7650, Lecture 7
Finite-State Automata

Jacob Eisenstein

September 10, 2013

Finite state machines are a formalism that can be used in many different parts of NLP, but their application to morphology — the internal structure of words — is especially well-developed.

1 Review of Finite State Acceptors

Basics :

- An alphabet Σ is a set of symbols
- A string ω is a sequence of symbols.
  The empty string ϵ contains zero symbols.
- A language $L \subseteq \Sigma^*$ is a set of strings.

An automaton is an abstract model of a computer which reads an input string and either accepts or rejects.

Chomsky Hierarchy Every automaton defines a language. Different automata define different classes of languages. The Chomsky Hierarchy:

- Finite-state automata define **regular** languages
- Pushdown automata define **context-free** languages
- Turing machines define **recursively-enumerable** languages

Finite-state automata A finite-state automaton $M = \langle Q, \Sigma, q_0, F, \delta \rangle$ consists of:

- A finite set of states $Q = \{q_0, q_1, \ldots, q_n\}$
- A finite alphabet Σ of input symbols
- A start state $q_0 \in Q$
- A set of final states $F \subseteq Q$
- A transition function δ

Determinism

- In a deterministic (D)FSA, $\delta : Q \times \Sigma \rightarrow Q$.
- In a nondeterministic (N)FSA, $\delta : Q \times \Sigma \rightarrow 2^Q$
- We can determinize any NFSA using the powerset construction, but the number of states in the resulting DFSA may be 2^n.
- Any **regular expression** can be converted into an NFSA, and thus into a DFSA.

The English Dictionary as an FSA We can build a simple "chain" FSA which accepts any single word. So, we can define the English dictionary with an FSA.

- Take the **union** of all of the chain FSAs by defining epsilon transitions from the start state to chain FSAs for each word (5303 states / 5302 arcs using a 850 word dictionary of "basic English")
- Eliminate the epsilon transitions by pushing the first letter to the front (4454 states / 4453 arcs)
- **Determinize** (2609 / 2608)
- **Minimize** (744 / 1535). The cost of minimizing an acyclic FSA is $O(E)$. (this data structure is called a trie)

Operations We've talked about three operations: union, determinization and minimization. Other important operations are **intersection** (only accept strings in both FSAs), **negation** (only accept strings not in FSA M), and **concatenation**. FSAs are closed under these operations, meaning that the output is still an FSA.

2 FSAs for Morphology

Back to morphology! Suppose that we want to write a program that accepts all **possible** English words, but none of the impossible ones:

- *grace, graceful, gracefully*
- *disgrace, disgraceful, disgracefully, ...*
- *Google,Googler,Googleology,...*
- **gracelyful*, **disungracefully*, ...

We could just make a list, and then take the union of the list using ϵ-transitions.

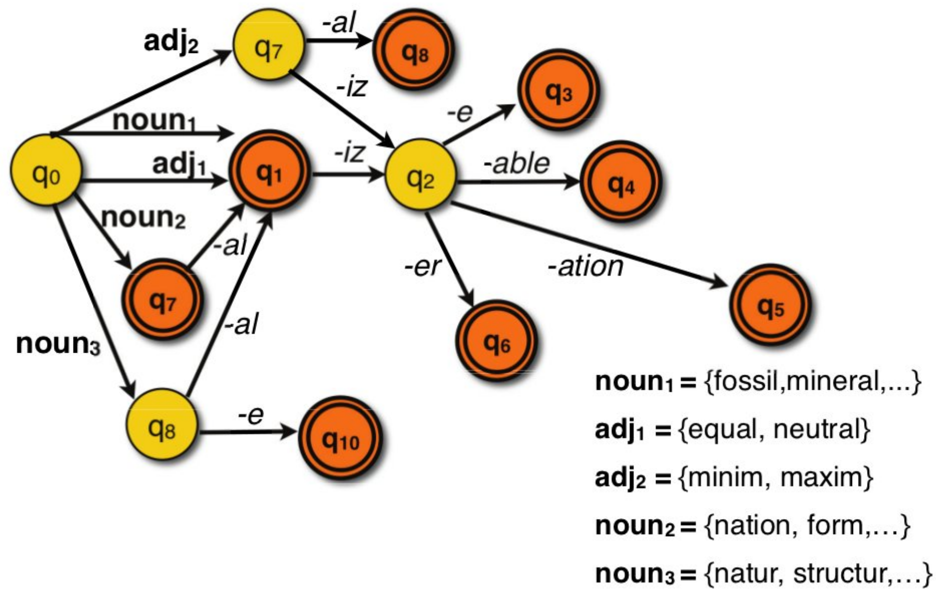
The list would get very long, and it would not account for productivity (our ability to make new words like *antiwordificationist*). So let's try to use finite state machines instead. Our FSA will have to encode rules about morpheme ordering, called *morphotactics*.[1]

Let's start with some examples:

- *grace*: $q_0 \rightarrow_{\text{stem}} q_1$
- *dis-grace*: $q_0 \rightarrow_{\text{prefix}} q_1 \rightarrow_{\text{stem}} q_2$
- *grace-ful*: $q_0 \rightarrow_{\text{stem}} q_1 \rightarrow_{\text{suffix}} q_2$
- *dis-grace-ful*: $q_0 \rightarrow_{\text{prefix}} q_1 \rightarrow_{\text{stem}} q_2 \rightarrow_{\text{suffix}} q_3$

Can we generalize these examples?

[1] Chomsky (1957) demonstrated that English is not finite-state language, but finite-state machinery can handle a huge range of morphology.



- This example abstracts away important details, like why *wordificate* is preferred to **wordifycate* (this is an **orthographic** rule). "Two-level morphology" is an approach to handling such transformations in a finite-state framework.

- It also misses a key point: sometimes we have choices, and not all choices are equally good.

  - Google counts:

    * *superfast*: 70M; *ultrafast*: 16M; *hyperfast*: 350K; *megafast*: 87K
    * *suckitude*: 426K; *suckiness*: 378K
    * *nonobvious*: 1.1M; *unobvious*: 826K; *disobvious*: 5K

  - Rather than asking whether a word is grammatically acceptable, we might like to ask how acceptable it is.

  - But finite state acceptors gives us no way to express *preferences* among technically valid choices.

  - We'll need to augment the formalism for this.

3 Weighted Finite State Automata

A weighted finite-state automaton $M = \langle Q, \Sigma, \pi, \xi, \delta \rangle$ consists of:

- A finite set of states $Q = \{q_0, q_1, \ldots, q_n\}$
- A finite alphabet Σ of input symbols
- Initial weight function, $\pi : Q \to \mathbb{R}$
- Final weight function $\xi : Q \to \mathbb{R}$
- A transition function $\delta : Q \times \Sigma \times Q \to \mathbb{R}$

  We have added a weight function that scores every possible transition.

- We can score any path through the WFSA by the sum of the weights.
- Arcs that we don't draw have infinite cost.
- The shortest-path algorithm can find the minimum-cost path for accepting a given string in $O(V \log V + E)$.

3.1 Applications of WFSAs

We can use WFSAs to score derivational morphology as suggested above. But let's start with a simpler example:

Edit distance . We can build an edit distance machine for any word. Here's one way to do this (there are others):

- Charge 0 for "correct" symbols and rightward moves
- Charge 1 for self-transitions (insertions)
- Charge 1 for rightward epsilon transitions (deletions)
- Charge 2 for "incorrect" symbols and rightward moves (substitutions)
- Charge ∞ for everything else

The total edit distance is the *sum* of costs across the best path through machine.

Probabilistic models For probabilistic models, we make the path costs equal to the likelihood:

$$\delta(q_1, s, q_2) = P(s, q_2|q_1) \tag{1}$$

Note that the total score for a path is now the *product* of the transitions. This enables probabilistic models, such as N-gram language models.

- A unigram language model is just one state, with V edges.
- A bigram language model will have V states, with V^2 edges.
- The text shows how to do an interpolated bigram/unigram language model. (Actually I think there's a better way, with only $V + 3$ states rather than $2V + 4$.)
  - Recall that the model is

    $$\hat{P}(y|x) = \lambda P_2(y|x) + (1 - \lambda)P_1(y), \tag{2}$$

    with $\hat{P}$ indicating the interpolated probability, P_2 indicating the bigram probability, and P_1 indicating the unigram probability.
  - Note that unlike the basic n-gram language models, our interpolated model has non-determinism: do we choose the bigram context or the unigram context?
  - For a sequence *a,b,a*, we want the final path score to be

    $$(\lambda P_2(a|*)+(1-\lambda)P_1(a))(\lambda P_2(b|a)+(1-\lambda)P_1(a))(\lambda P_2(b|a)+(1-\lambda)P(b)) \tag{3}$$

Notice that a lot of the details are different between these three examples:

- Scoring
  - In the derivational morphology FSA, we wanted a boolean "score": is the input a valid word or not?
  - In the edit distance WFSA, we wanted a numerical (integer) score, with lower being better.
  - In the interpolated language model, we wanted a numerical (real) score, with higher being better.

- Nondeterminism
  - In the derivational morphology FSA, we accept if there is any path to a terminating state.
  - In the edit distance WFSA, we want the score of the single best path.
  - In the interpolated language model, we want to sum over non-deterministic choices.
- How can we combine all of these possibilities into a single formalism? The answer is semiring notation.

3.2 Semirings

A semiring is a system $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$

- $\mathbb{K}$ is the set of possible values, e.g. $\{\mathbb{R}_+ \cup \infty\}$, the non-negative reals union with infinity
- $\oplus$ is an addition operator
- $\otimes$ is a multiplication operator
- $\overline{0}$ is the additive identity
- $\overline{1}$ is the multiplicative identity

A semiring must meet the following requirements:

- $(a \oplus b) \oplus c = a \oplus (b \oplus c)$, $(\overline{0} \oplus a) = a$, $a \oplus b = b \oplus a$
- $(a \otimes b) \otimes c = a \otimes (b \otimes c)$, $a \otimes \overline{1} = \overline{1} \otimes a = a$
- $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$, $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
- $a \otimes \overline{0} = 0 \otimes \overline{a} = \overline{0}$

Some semirings of interest:

Name	$\mathbb{K}$	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$	Applications
Boolean	$\{0,1\}$	$\vee$	$\wedge$	0	1	identical to an unweighted FSA
Probability	$\mathbb{R}_+$	$+$	$\times$	0	1	sum of probabilities of all paths
Log-probability	$\mathbb{R} \cup -\infty \cup \infty$	$\oplus_{\log}$	$+$	∞	0	negative log marginal probability
Tropical	$\mathbb{R} \cup -\infty \cup \infty$	min	$+$	∞	0	best single path

where $\oplus_{\log}(a,b)$ is defined as $-\log(e^{-a} + e^{-b})$.

Semirings allow us to compute a more general notion of the "shortest path" for a WFSA.

- Our initial score is $\overline{1}$
- When we take a step, we use $\otimes$ to combine the score for the step with the running total.
- When nondeterminism lets us take multiple possible steps, we combine their scores using $\oplus$.

Example Let's see how this works out for our language model example.

$$\begin{aligned} score(\{a,b,a\}) =& \overline{1} \\ & \otimes (\lambda P_2(a|*) \oplus (1-\lambda) \otimes P_1(a)) \\ & \otimes (\lambda P_2(b|a) \oplus (1-\lambda) \otimes P_1(b)) \\ & \otimes (\lambda P_2(a|b) \oplus (1-\lambda) \otimes P_1(a)) \end{aligned}$$

Now if we plug in the **probability semiring**, we get

$$\begin{aligned} score(\{a,b,a\}) =& 1 \\ & \times (\lambda P_2(a|*) + (1-\lambda) P_1(a)) \\ & \times (\lambda P_2(b|a) + (1-\lambda) P_1(b)) \\ & \times (\lambda P_2(a|b) + (1-\lambda) P_1(a)) \end{aligned}$$

- The score of the input will the **sum** of probabilities across all paths that successfully process the input.

- Note that if we really want to have a score that we can minimize, we should use the log-probability semiring, where the score will be the **negative** log-probability. Minimizing this is equivalent to maximizing the (log) probability.
- What happens if we use the tropical semiring?

Software There are mature software toolkits for working with finite state machines. OpenFST is a C++ package which I have had some experience with; it's fast and relatively well-documented. XFST and Carmel are other options.