Finite State Methods

Kevin Duh

Intro to NLP, Fall 2019
Outline

1. Finite-State Automaton (FSA)
2. Finite-State Transducer (FST)
3. Applications in Morphology
4. Semiring
Finite-State Methods in NLP

• Application of Automata Theory, focusing on
  • properties of **string sets** or **string relations**
  • with a notion of "**bounded dependency**"
    • e.g. phonology and morphology. some syntax
Example: SheepTalk

- Here are some strings produced by a sheep:
  - baa!
  - baaa!
  - baaaaa!

- Some strings not produced by a sheep:
  - baabaa!

- We can model this with a regular expression `/baa*!`/

- We can also model this with a finite-state automaton
Finite-State Automaton (FSA)

- **Definition:**
  - $Q = q_0q_1,...q_{N-1}$: finite set of $N$ states, $q_0$ is start state
  - $\Sigma$: finite input alphabet of symbols
  - $F$: set of final states in $Q$
  - Transition function: given state $q$ and input $i$, returns new state $q'$

- A string is **accepted/recognized** by the FSA if it starts in $q_0$ and reaches a valid final state
Finite-State Automaton
(Computational Device)

Regular Expression
(Descriptive Notation)

Regular Language
(Set of accepted strings)

Note: A regular language (e.g. SheepTalk) can contain infinitely many strings, but modeled by one FSA
Non-Deterministic FSA

• There are ways to check if a string is recognized by a Non-Deterministic FSAs

• Also possible to convert Non-Deterministic FSAs to Deterministic ones

\( \varepsilon \) : epsilon arcs consume no symbols
Operations on Regular Languages

• Suppose $L_1$ and $L_2$ are regular languages
  
  • $L_1$ is sheeptalk— /baa*!/
  
  • $L_2$: 3 strings— ba! baba! bababa!

• **Intersection** $L_1 \cap L_2$ : the language consisting of strings in both languages

• **Difference** $L_1 - L_2$: the language consisting of strings that are in $L_1$ but not in $L_2$

• **Complementation**: $\Sigma^* - L_1$, i.e. set of strings that aren’t in $L_1$
Weighted Finite-State Automata (WFSA)

- Associates some weight (e.g. number) to each arc
- This can be useful!
Can N-grams be represented by a WFSA?

- Yes!
- One state for each (n-1)-gram history
- Each arc is a word & probability $p(\text{word}|\text{history})$

**Bigram with 2 vocab: A, B**

- A $\xrightarrow{\text{A}/p(\text{A}|\text{A})}$ B $\xrightarrow{\text{B}/p(\text{B}|\text{A})}$ A $\xrightarrow{\text{A}/p(\text{A}|\text{B})}$ B $\xrightarrow{\text{B}/p(\text{B}|\text{B})}$ A

**Trigram with 2 vocab**

- A $\xrightarrow{\text{A}/p(\text{A}|\text{A},\text{B})}$ B $\xrightarrow{\text{B}/p(\text{B}|\text{B},\text{B})}$ A $\xrightarrow{\text{A}/p(\text{A}|\text{B},\text{B})}$ B $\xrightarrow{\text{B}/p(\text{B}|\text{B},\text{A})}$ A $\xrightarrow{\text{A}/p(\text{A}|\text{B},\text{A})}$ B $\xrightarrow{\text{B}/p(\text{B}|\text{A},\text{B})}$ A $\xrightarrow{\text{A}/p(\text{A}|\text{A},\text{A})}$ B $\xrightarrow{\text{B}/p(\text{B}|\text{A},\text{A})}$ A

Outline

1. Finite-State Automaton (FSA)
2. Finite-State Transducer (FST)
3. Applications in Morphology
4. Semiring
Finite-State Transducer (FST)

- Maps between two sets of symbols and defines relation between sets of strings
- Can view a recognizer of string pairs
  - Is `baa!` with `BAA!` recognized?
- Alternatively, can view as a translator
  - `baa!` => `BAA!`

Note: FSA is a special case of FST where input=output
Operations on FSTs

• Inversion: Flips the input and output symbols on each arc

• Composition:
  • $T_1$ is transducer from $I_1$ to $O_1$
  • $T_2$ is transducer from $O_1$ to $O_2$
  • $T_1 \circ T_2$ maps $I_1$ to $O_2$
Composition is very useful!

What happens when you compose these two transducers?
Composition allows us to declaratively describe transform on a set of strings:
e.g. Speech Recognition: $H \circ C \circ L \circ G$
$G = $ Language Model FSA
$L = $ Pronunciation Lexicon, FST mapping (context-independent) phones to words
$C = $ FST mapping context-dependent phone sequence to context-independent ones
$H = $ FST mapping acoustic model states to context-dependent phones

How to compose FSTs?

- For every state in $s \in T_1$ and $t \in T_2$, create $(s, t)$
- Create arc from $(s_1, t_1)$ to $(s_2, t_2)$ if
  - There’s arc from $s_1$ to $s_2$ with output label $i$
  - And there’s arc from $t_1$ to $t_2$ with input label $i$
- A little bit more complicated for epsilon arcs

Outline

1. Finite-State Automaton (FSA)
2. Finite-State Transducer (FST)
3. Applications in Morphology
4. Semiring
## Morphological Parsing

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>cats</td>
<td>cat +N +PL</td>
</tr>
<tr>
<td>cat</td>
<td>cat +N +SG</td>
</tr>
<tr>
<td>cities</td>
<td>city +N +PL</td>
</tr>
<tr>
<td>geese</td>
<td>goose +N +PL</td>
</tr>
<tr>
<td>caught</td>
<td>(catch +V +PAST-PART)</td>
</tr>
<tr>
<td></td>
<td>or (catch +V +PAST)</td>
</tr>
<tr>
<td>merging</td>
<td>merge +V +PRES-PART</td>
</tr>
<tr>
<td>(word in surface form)</td>
<td>(lemma and morphological features)</td>
</tr>
</tbody>
</table>
Morphological Parsing with FSTs

Lexical: $\{f \ o \ x \ +N \ +PL\}$

Intermediate: $\{f \ o \ x \ ^\wedge \ s\}$

Surface: $\{f \ o \ x \ e \ s\}$

Figure: Jurafsky & Martin (2009) Speech & Language Processing, 2nd ed., Pearson
Figure: Jurafsky & Martin (2009) Speech & Language Processing, 2nd ed., Pearson

reg-noun-stem: fox
reg-noun-stem: cat
irreg-sg-noun-form: sheep
irreg-pl-noun-form: sheep
irreg-sg-noun-form: mouse
irreg-pl-noun-form: mice
Orthography Rules: “foxs” is not correct

\[ \varepsilon \rightarrow e / \{ x, s, z \} \]

Lexical

<table>
<thead>
<tr>
<th>f</th>
<th>o</th>
<th>x</th>
<th>+N</th>
<th>+PL</th>
</tr>
</thead>
</table>

Intermediate

| f | o | x | ^ | s | # |

Surface

| f | o | x | e | s |

Figure: Jurafsky & Martin (2009) Speech & Language Processing, 2nd ed., Pearson
Example in Turkish

[Oflazer (1993), Two-Level Description of Turkish Morphology]

Input
Morpheme Structure     Gloss
                             English meaning

ca\l{\i}s\=man\=\i n
ca\l{\i}s+mA+Hn+nHn      [ V(\l{\i}s)+VtoN(ma)+2PS-POSS+GEN ]
of your work(\=i\i ng)
ca\l{\i}s+mA+nHn            [ V(\l{\i}s)+VtoN(ma)+GEN ]
of the work(\=i\i ng)

Rules for Phonology/Orthography, e.g.

H:0 \Rightarrow V(\cdot\cdot) + :0 ___
An H vowel is deleted if the last phoneme of the stem it is being affixed to is a vowel. For example:

Lexical: masa+Hm   N(table)+1PS-POSS
Surface: masa00m   masam

X:0 \Leftrightarrow C(\cdot\cdot) + :0 ___ (C) V
This rule deletes the beginning s, n, or a y of a suffix when it gets affixed to a stem ending in a consonant.

Lexical: ev+nHn   N(house)+GEN
Surface: ev00in   evin

Morphotactics FST: order of morphemes
Outline

1. Finite-State Automaton (FSA)
2. Finite-State Transducer (FST)
3. Applications in Morphology
4. Semiring
Recall Viterbi vs Forward Algorithms in HMMs

• The way we compute is almost the same! Just swap max & sum

• Is there a way to generalize this on arbitrary FSA/FST?

• Yes! We’ll do so with semirings
Semirings

- Define weights as a semiring:
  - K: set of values
  - \(\oplus\): associative, commutative, has 0 as identity
  - \(\otimes\): associative, has 1 as identity, has 0 as annihilator, distributes with respect to \(\oplus\)
  - 0 and 1

- Example: Tropical
  - K: All Reals \(\mathbb{R}\), \(\pm\infty\)
  - \(\oplus\): min
  - \(\otimes\): +
  - 0: \(\infty\)
  - 1: 0

- Example: Boolean
  - K: \{0,1\}
  - \(\oplus\): OR
  - \(\otimes\): AND
  - 0: 0
  - 1: 1

*from Abstract Algebra: Semiring is like a ring but does not require additive inverse*
Weighted FSA/FST

- We can use a general algorithm but change semirings on weights

- baa! in probability semiring: \( .2 \times .3 \times .2 \times .1 \)

- baa! in tropical semiring: \( .2 + .3 + .2 + .1 \)

<table>
<thead>
<tr>
<th>SEMIRING</th>
<th>SET</th>
<th>( \oplus )</th>
<th>( \otimes )</th>
<th>( \bar{0} )</th>
<th>( \bar{1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>{0, 1}</td>
<td>( \lor )</td>
<td>( \land )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Probability</td>
<td>( \mathbb{R}_+ )</td>
<td>+</td>
<td>( \times )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Log</td>
<td>( \mathbb{R} \cup {-\infty, +\infty} )</td>
<td>( \oplus_{\log} )</td>
<td>+</td>
<td>+(\infty)</td>
<td>0</td>
</tr>
<tr>
<td>Tropical</td>
<td>( \mathbb{R} \cup {-\infty, +\infty} )</td>
<td>min</td>
<td>+</td>
<td>+(\infty)</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
x \oplus_{\log} y = -\log(e^{-x} + e^{-y})
\]
Weighted FSA/FST

- We define two operators in a weighted graph

- the weight of a path uses $\otimes$ on arcs:
  
  - weight of path “using data is” = $1 \otimes 0.66 \otimes 0.5$

- the weight of a vertex with multiple incoming paths uses $\oplus$:
  
  - weight at state 4 = weight(using data is) $\oplus$ weight(“using data are”) $\oplus$ weight(“using intuition is”)
Algorithms on WFST

- Single-source shortest-path
  - Question: Which semiring is used in Viterbi Algo in HMM? How about Forward Algo?
- N shortest paths
- etc
Optimizing WFST

Determinization:
- Eliminates ambiguity in input paths, so each state won’t have outgoing arc with same labels
- May improve efficiency of shortest-path

Figure from: J. Novak. http://www.gavo.t.u-tokyo.ac.jp/~novakj/wfst-algorithms.pdf
Optimizing WFST

Minimize #states

Push weights to beginning if possible

Figure from: J. Novak. http://www.gavo.t.u-tokyo.ac.jp/~novakj/wfst-algorithms.pdf
Summary

1. Finite-State Automaton (FSA)
2. Finite-State Transducer (FST)
3. Applications in Morphology
4. Semiring