Outline

• Motivation
  – Brief History of Grammatical Inference
  – Application of Grammatical Inference
  – Potential Application in Malware Detection

• Automata and Languages
  – Background
  – Chomsky Hierarchy

• GI Algorithms
  – Learning DFA
    • RPNI
    • EDSM
    • Membership Query
  – Learning k-testable Languages

• Open Problems
Basic Stringology

- If $x = uv$ is a string, then $u$ is a *prefix* of the string $x$.
- If $x = uv$ is a string, then $v$ is a *suffix* of the string $x$.
- Given a string $x$, $u$ is a *substring* (or a *factor*) of $x$ if there are two strings $l$ and $r$ such that $x = lur$. In that case we will also say that $x$ is a *superstring* of $u$.
- $u$ is a subsequence of $x$ if it can be obtained from $x$ by erasing letters from $x$.
  - $xy$ is a subsequence of $xay$. 
Basic Stringology

• Example: Consider $x = abbababa$; then $abb$ is a prefix of $x$, $aba$ is a suffix of $x$. Both are substrings of $x$ and so is $bab$. $bbbb$ is a subsequence of $x$, but not a substring.

• Note:
  – Finding the longest common subsequence between $u$ and $v$ is in $O(|u| \cdot |v|)$ time;
  – Finding the longest common subsequence of a set of strings is NP-hard;
  – Checking if string $u$ is a subsequence (or substring) of string $x$ requires $O(|u| + |v|)$ time.
Basic Stringology

- A **prefix set** $\text{Pref}(L)$ of the language $L$ is defined as:
  \[ \text{Pref}(L) = \{ u \in \Sigma : uv \in L \} \]
- Example: $L = \{aa, ab, aaa, aba\}$
  \[ \text{Pref}(L) = \{a, ab, aa, aaa, aba\} \]
Basic Stringology: Ordering Strings

- Suppose we have a total order relation over the letters of an alphabet $\Sigma$.
- Different orders can be defined over $\Sigma^*$.
  - **alphabetical order** $\leq_{\text{alpha}}$
  - **prefix order:** $x \leq_{\text{pref}} y$ if $\exists w \in \Sigma^*: y = wx$
  - **lexicographic order:**
    $$x \leq_{\text{lex}} y \text{ if } x \leq_{\text{pref}} y \lor [x = uaw : y = ubz \land a \leq_{\text{alpha}} b]$$
  - **subsequence order:**
    $$x \leq_{\text{sub-sec}} y \text{ if } x \text{ is a subsequence of } y$$
Basic Stringology: Ordering Strings

• A more interesting order is the **length-lexicographic order** (also sometimes called the *hierarchical* or *length-lex* order).

\[ x \leq_{\text{lex-length}} y \text{ if } |x| < |y| \lor (|x| = |y| \land x \leq_{\text{lex}} y) \]
Exercises:

• Prove that if $L$ is a finite set, then so is $\text{Pref}(L)$.

• Order the following strings $aabab, bbabacaa, cc, acabaab, baaa$ in the order of:
  – lex-length
  – prefix order
  – lexicographic order
  – subsequence order
Prefix Tree Acceptor:

• The prefix tree acceptor of $S_+$ is the DFA $= (\Sigma, Q, q_\lambda, F, \delta)$ denoted $PTA(S_+)$ such that:

  $-Q = \{q_u : u \in Pref(S_+)\}$

  $\forall u a \in Pref(S_+) : \delta(q_u, a) = q_{ua}$

  $F = \{q_u : u \in S_+\}$
Prefix Tree Acceptor:

- Example: \( S_+ = \{aa, ab\} \)
- \( \text{Pref}(S_+) = \{a, aa, ab\} \)
- \( Q = \{\lambda, a, aa, ab\} = \{1, 2, 3, 4\} \)
- \( F = \{3, 4\} \)
RPNI

- Regular Positive and Negative Grammatical Inference
- *Identifying regular languages in polynomial time*
- Jose Oncina & Pedro García 1992
RPNI

• It is a state merging algorithm;
• It identifies any regular language in the limit;
• It works in polynomial time;
• It admits polynomial characteristic sets.
The Classic RPNI

**Input:** A sample $S = S^+ \cup S^-

**Output:** A DFA compatible with $S$

begin

// Initialization
$\pi = \pi_0 = \{\{0\}, \{1\}, \ldots, \{N - 1\}\}$
$M_\pi = PTA(S^+)$
// Perform state merging
for $i = 1$ to $N - 1$
  for $j = 0$ to $i - 1$
    // Merge the block of $\pi$ containing state $i$ with the block containing state $j$
    $\tilde{\pi} = \pi \setminus \{B(i, \pi), B(j, \pi)\} \cup \{B(i, \pi) \cup B(j, \pi)\}$
    // Obtain the quotient automaton $M_{\tilde{\pi}}$
    $M_{\tilde{\pi}} = derive(M, \tilde{\pi})$

//... continued
The Classic RPNI

// Determine the quotient automaton (if necessary) by state merging
\hat{\pi} = \text{deterministic-merge}(M_\pi)
// Does M_{\hat{\pi}} reject all strings in S^-?
\text{if consistent}(M_{\hat{\pi}}, S^-)
\text{then}
  // Treat M_{\hat{\pi}} as the current hypothesis
  M_\pi = M_{\hat{\pi}}
  \pi = \hat{\pi}
  \text{break}
\text{end if}
\text{end for}
\text{end for}
return \ M_\pi
end
The Classic RPNI

- Example: $S_+ = \{aa, ab\}, S_- = \{aaa, aab\}$
The Classic RPNI

1,2,3,4

1,2,4

1,2,3,4

1,2,3

1,2,4

1,2,3,4

1,2,3

1,2,4

1,2,3

1,2,4

1,2,3

1,2,4

1,2,3

1,2,4

1,2,3

1,2,4

1,2,3

1,2,4

1,2,3

1,2,4

1,2,3

1,2,4

1,2,3

1,2,4

1,2,3

1,2,4

1,2,3

1,2,4

1,2,3

1,2,4

1,2,3

1,2,4

1,2,3
RPNI

• Algorithm RPNI starts by building $PTA(S_+)$ from the positive data
• Then iteratively chooses possible merges,
• Checks if a given merge is correct and is made between two compatible states
• Makes the merge if admissible
• Promotes the state if no merge is possible
**RPNI Promote**

Input: a DFA $A = (\Sigma, Q, q_0, F_A, F_R, \delta)$, sets $\text{Red}, \text{Blue} \subseteq Q$, $q_u \in \text{Blue}$

Output: $A$, $\text{Red}$, $\text{Blue}$ updated

$\text{Red} \leftarrow \text{Red} \cup \{q_u\}$;
$\text{Blue} \leftarrow \text{Blue} \cup \{\delta(q_u, a), a \in \Sigma\}$;
return $A$, $\text{Red}$, $\text{Blue}$

```plaintext
defunction [dfr, RED, BLUE] = RPNI_PROMOTE(blue, dfa)
dfa.RED = union(dfa.RED, blue);
RED = dfa.RED;
for i = 1:length(dfa.Alphabets)
    if(GetTransitionState(dfa, blue, dfa.Alphabets(i))~=0)
        dfa.BLUE = union(dfa.BLUE, GetTransitionState(dfa, blue, dfa.Alphabets(i)));
    end
end
BLUE = dfa.BLUE;
end
```
RPNI Compatible

Input: $A, S$
Output: a Boolean, indicating if $A$ is consistent with $S$
for $w \in S$ do
    if $\delta_A(q_\lambda, w) \cap F_A \neq \emptyset$ then return false
end
return true

function flag = RPNI_COMPATIBLE(dfa, S_negative)

S_negative = strrep(S_negative, ' ', '');

for i = 1:length(S_negative)
    if (IsStringAccepted(char(S_negative(i)), dfa) == 1)
        flag = 0;
        return;
    end
end

flag = 1;
end
**RPNI Merge**

**Input:** a DFA $\mathcal{A}$, states $q \in \text{RED}$, $q' \in \text{BLUE}$

**Output:** $\mathcal{A}$ updated

Let $(q_f, a)$ be such that $\delta_\mathcal{A}(q_f, a) \leftarrow q'$;

$\delta_\mathcal{A}(q_f, a) \leftarrow q$;

return $\text{RPNI-FOLD}(\mathcal{A}, q, q')$

```matlab
function dfa = RPNI_MERGE(dfa, q1, q2)
    % first find a (q_f, a) such that del(q_f, a)<--q2
    for i = 1:length(dfa.Alphabets)
        q_f = GetSourceState(dfa, q2, dfa.Alphabets(i));
        a = dfa.Alphabets(i);
        if(q_f~=0)
            break;
        end
    end
    if(length(q_f)>1)
        display(q_f);
    end
    % Set the transition (q_f, a) as q1
    dfa = SetTransition(dfa, q_f, a, q1);
    dfa = RPNI_FOLD(dfa, q1, q2);
end
```
RPNI Fold

Input: a DFA $\mathcal{A}$, states $q, q' \in Q$, $q'$ being the root of a tree
Output: $\mathcal{A}$ updated, where subtree in $q'$ is folded into $q$
if $q' \in F_\mathcal{A}$ then $F_\mathcal{A} \leftarrow F_\mathcal{A} \cup \{q\}$;
for $a \in \Sigma$ do
  if $\delta_\mathcal{A}(q', a)$ is defined then
    if $\delta_\mathcal{A}(q, a)$ is defined then
      $\mathcal{A} \leftarrow \text{RPNI-Fold}(\mathcal{A}, \delta_\mathcal{A}(q, a), \delta_\mathcal{A}(q', a))$
    else
      $\delta_\mathcal{A}(q, a) \leftarrow \delta_\mathcal{A}(q', a)$;
  else
    $\delta_\mathcal{A}(q, a) \leftarrow \delta_\mathcal{A}(q', a)$;
end
end
return $\mathcal{A}$
RPNI Fold

```matlab
function dfa = RPNI_FOLD(dfa, q1, q2)
    if(find(dfa.FinalAcceptStates == q2))
        dfa.FinalAcceptStates = union(dfa.FinalAcceptStates, q1);
    end

    for i = 1:length(dfa.Alphabets)
        a = dfa.Alphabets(i);
        transition_from_q2 = GetTransitionState(dfa, q2, a);
        transition_from_q1 = GetTransitionState(dfa, q1, a);
        if(transition_from_q2~=0)
            if(transition_from_q1~=0)
                dfa = RPNI_FOLD(dfa, transition_from_q1, transition_from_q2);
            else
                dfa = SetTransition(dfa, q1, a, transition_from_q2);
                dfa = DeleteTransition(dfa, q2, a);
            end
        end
    end
end
```
**RPNI**

**Input:** a sample \( S = \langle S_+, S_- \rangle \), functions \textit{COMPATIBLE}, \textit{CHOOSE}

**Output:** a DFA \( A = \langle \Sigma, Q, q_\lambda, F_A, F_R, \delta \rangle \)

\[ A \leftarrow \text{BUILD-PTA}(S_+); \]
\[ \text{RED} \leftarrow \{ q_\lambda \}; \]
\[ \text{BLUE} \leftarrow \{ q_a : a \in \Sigma \cap \text{PREF}(S_+) \}; \]

\[ \text{while BLUE} \neq \emptyset \text{ do} \]
  \[ \text{CHOOSE}(q_b \in \text{BLUE}); \]
  \[ \text{BLUE} \leftarrow \text{BLUE} \setminus \{ q_b \}; \]
  \[ \text{if} \ \exists q_r \in \text{RED} \text{ such that} \]
  \[ \text{RPNI-COMPATIBLE}(\text{RPNI-MERGE}(A, q_r, q_b), S_-) \text{ then} \]
  \[ A \leftarrow \text{RPNI-MERGE}(A, q_r, q_b); \]
  \[ \text{BLUE} \leftarrow \text{BLUE} \cup \{ \delta(q, a) : q \in \text{RED} \land a \in \Sigma \land \delta(q, a) \notin \text{RED} \}; \]
  \[ \text{else} \]
  \[ A \leftarrow \text{RPNI-PROMOTE}(q_b, A) \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{for} \ q_r \in \text{RED} \text{ do} \]
  \[ \text{/* mark rejecting states */} \]
  \[ \text{if} \ \lambda \in (L(A_{q_r})^{-1} S_-) \text{ then} \]
  \[ F_R \leftarrow F_R \cup \{ q_r \} \]
\[ \text{end} \]
\[ \text{return} \ A \]
function dfa = RPNI(positive, negative)
    dfa = BUILD_PTA(positive);
    dfa.RED = [dfa.RED, dfa.FiniteSetOfStates(1)];
    for i = 1:length(dfa.Alphabets)
        temp_blue = GetTransitionState(dfa, dfa.FiniteSetOfStates(1), dfa.Alphabets(i));
        if(temp_blue~=0) % 0 means no transition
            dfa.BLUE = [dfa.BLUE, temp_blue];
        end
    end
    while (~isempty(dfa.BLUE))
        dfa.BLUE = sort(dfa.BLUE);
        q_b = dfa.BLUE(1);
        dfa.BLUE = [dfa.BLUE(2:length(dfa.BLUE))];
        promote = 1;
        for i = 1:length(dfa.RED)
            dfa_merged = RPNI_MERGE(dfa, dfa.RED(i), q_b);
            if(RPNI_COMPATIBLE(dfa_merged, negative))
                dfa = dfa_merged;
                dfa = AddNewBlueStates(dfa);
                promote = 0;
                break;
            end
        end
        if (promote==1)
            dfa = RPNI_PROMOTE(q_b, dfa);
            display('promoting:');
            display(q_b);
        end
    end
RPNI

- Example: $S_+ = \{aa, ab\}, S_- = \{aaa, aab\}$

Build the PTA
RPNI

- Example: $S_+ = \{aa, ab\}, S_- = \{aaa, aab\}$

Set the initial RED & BLUE states
**RPNI**

- Example: \( S_+ = \{aa, ab\}, S_- = \{aaa, aab\}\)

Merge 1 and 2
RPNI

• Example: \( S_+ = \{aa, ab\}, S_- = \{aaa, aab\} \)

After folding the merge 1 and 2

Accepts \( aaa \), therefore rejected
**RPNI**

- Example: \( S_+ = \{aa, ab\}, S_0 = \{aaa, aab\} \)

Promote 2
RPNI

• Example: \( S_+ = \{aa, ab\}, S_- = \{aaa, aab\} \)

Merge 1 & 3
After folding....

Accepted!
**RPNI**

- Example: \( S_+ = \{aa, ab\}, S_- = \{aaa, aab\} \)

  Merge 1 & 4
  After folding....

  Accepted!
Exercises

• Run RPNI for the order relations $\leq_{\alpha}$ and $\leq_{\text{lex-length}}$ on
  
  $S_+ = \{a, abb, bab, babbb\}$
  
  $S_- = \{ab, bb, aab, b, aaaa, babb\}$
Noisy DFA

- Classic machine learning problem
- Studied in the literature for well over thirty years
- Interesting for EC, Neural Net, Grammatical Inference and Machine Learning communities
The Problem Setup

• A random DFA (deterministic finite automaton) is constructed
  – By choosing the maximum number of states
  – Randomly selecting entries in the state transitions matrix
  – Randomly selecting state labels (accept or reject)
• This DFA (called the target DFA) is used to label two sets of randomly chosen strings (train + test)
• By learning only from the training set, must try to predict test set labels
• Twist: training set labels are corrupted with 10% noise (i.e. 1 in 10 chance of flipping each label)
• Target DFA ranging from 10 through to 50 states
Heuristic State Merging

• Basic idea: build prefix-tree acceptor
• This fits only the training data
• Then merge states while maintaining training set consistency
• Best methods have emerged from Abbadingo One DFA inference problem
• E.g. EDSM: evidence driven state merging
• Clever algorithms – carefully designed and tuned
Evolutionary Approaches

- Typically much simpler
- Encode DFA
- Evolve it!!!
- Heuristic methods work very well on noise-free problems
- But is evolution better able to cope with noise?