Efficient Online Sequence Prediction with Side Information

Han Xiao, Claudia Eckert

Technische Universität München
Technical University of Munich, Germany

{xiaoh, claudia.eckert}@in.tum.de

ICDM 13 Dallas, TX
December 10, 2013
# System Call Sequence
## Arguments and return values

<table>
<thead>
<tr>
<th>Call</th>
<th>Argument</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>(&quot;/lib/librt.so&quot;, O_RDONLY)</td>
<td>= 3</td>
</tr>
<tr>
<td>read</td>
<td>(3, &quot;\177ELF\2\1\1&quot;)</td>
<td>= 832</td>
</tr>
<tr>
<td>fstat</td>
<td>(3, {st_mode=S_IFREG, st_size=317})</td>
<td>= 0</td>
</tr>
<tr>
<td>mmap</td>
<td>(NULL, 4096, PROT_READ</td>
<td>PROT_WRITE)</td>
</tr>
<tr>
<td>mmap</td>
<td>(NULL, 2129016, PROT_READ)</td>
<td>= 0x7f2fc</td>
</tr>
<tr>
<td>mprotect</td>
<td>(0x7f2f7, 2093056, PROT_NONE)</td>
<td>= 0</td>
</tr>
<tr>
<td>mmap</td>
<td>(0x7f2fa, 8192, PROT_READ)</td>
<td>= 0x7f2fb</td>
</tr>
<tr>
<td>close</td>
<td>(3)</td>
<td>= 0</td>
</tr>
</tbody>
</table>
System Call Sequence
Local and long-range dependency
Observations

Online learning

- Long-range dependency.
  A model can memorize a long history.

- Arguments and return values can be indicative in predicting the next system call.
  A model can harness side information for prediction.

- A process can exhibit different behaviors at various points during its lifetime, depending on user's input and the status of the system.
  A model can handle non-stationarity.
Observations

**Online learning**

- Long-range dependency.
  
  *A model can memorize a long history.*

- Arguments and return values can be indicative in predicting the next system call.
  
  *A model can harness side information for prediction.*

- A process can exhibit different behaviors at various points during its lifetime, depending on user’s input and the status of the system.
  
  *A model can handle non-stationarity.*
Online learning

- Long-range dependency.
  A model can memorize a long history.

- Arguments and return values can be indicative in predicting the next system call.
  A model can harness side information for prediction.

- A process can exhibit different behaviors at various points during its lifetime, depending on user’s input and the status of the system.
  A model can handle non-stationarity.
Observations

**Online learning**

- Long-range dependency.
  
  A model can memorize a long history.

- Arguments and return values can be indicative in predicting the next system call.
  
  A model can harness side information for prediction.

- A process can exhibit different behaviors at various points during its lifetime, depending on user’s input and the status of the system.
  
  A model can handle non-stationarity.
Goal

Problem

Predicting the next system call given an observed sequence.

Applications

- Anomaly detection [warrender1999, eskin2001];
- buffer cache management in operating system [fricke2011];
- power management in smartphones [pathak2011];
- sandbox systems [oyama2005].
Goal

Problem
Predicting the next system call given an observed sequence.

Applications
- Anomaly detection [warrender1999, eskin2001];
- buffer cache management in operating system [fricke2011];
- power management in smartphones [pathak2011];
- sandbox systems [oyama2005].
Motivation

Problem Formulation

Proposed Method

Experimental Results

Conclusions

Notations

Learning is performed in rounds

Observed symbols $\Sigma := \{1, \ldots, K\}$ (all system calls)

$t^{th}$ symbol $x[t] \in \Sigma$ (a system call)

context of $x[t]$ $x[1:t−1]$ (previous seen system calls)

Online Learning

On round $t$, observe $x[1:t−1]$

1. predict $\hat{x}[t] \in \Sigma$ according to the current model;
2. the true symbol $x[t]$ is revealed;
3. suffer a loss reflecting the degree to which its prediction was wrong;
4. modify the prediction rule.

The explicit goal is to improve the accuracy of model’s predictions for the rounds to come.
Notations
Learning is performed in rounds

Observed symbols \( \Sigma := \{1, \ldots, K\} \) (all system calls)

\( t^{th} \) symbol \( x[t] \in \Sigma \) (a system call)

context of \( x[t] \) \( x[1:t-1] \) (previous seen system calls)

Online Learning

On round \( t \), observe \( x[1:t-1] \)

1. predict \( \hat{x}[t] \in \Sigma \) according to the current model;
2. the true symbol \( x[t] \) is revealed;
3. suffer a loss reflecting the degree to which its prediction was wrong;
4. modify the prediction rule.

The explicit goal is to improve the accuracy of model’s predictions for the rounds to come.
Basic Idea

Formulate the sequence prediction problem as linear separation problem.

- Predictive function is mapped to a hyperplane $\mu$ in Hilbert space.
- The context $x^{[1:t-1]}$ is mapped to a vector $\psi$ in Hilbert space.
- Prediction is made by $\mu_k^t \cdot \psi^t$.

1. Representing $x^{[1:t-1]}$ as a vector $\psi$
   - Growing a context tree.
   - Bounding the size of the tree (i.e. depth and nodes).
   - Memory-efficient update.

2. Finding the hyperplane $\mu$.
   - Based on confidence-weighted classifiers [crammer2008, etc.].
   - Conservative and partial update.
Basic Idea

Formulate the sequence prediction problem as linear separation problem.

- Predictive function is mapped to a hyperplane $\mu$ in Hilbert space.
- The context $x^{[1:t-1]}$ is mapped to a vector $\psi$ in Hilbert space.
- Prediction is made by $\mu^{[t]}_k \cdot \psi^{[t]}$.

1. Representing $x^{[1:t-1]}$ as a vector $\psi$
   - Growing a context tree.
   - Bounding the size of the tree (i.e. depth and nodes).
   - Memory-efficient update.

2. Finding the hyperplane $\mu$.
   - Based on confidence-weighted classifiers [crammer2008, etc.].
   - Conservative and partial update.
Basic Idea

Formulate the sequence prediction problem as linear separation problem.

- Predictive function is mapped to a hyperplane $\mu$ in Hilbert space.
- The context $x^{[1:t-1]}$ is mapped to a vector $\psi$ in Hilbert space.
- Prediction is made by $\mu_k^{[t]} \cdot \psi^{[t]}$.

1. Representing $x^{[1:t-1]}$ as a vector $\psi$
   - Growing a context tree.
   - Bounding the size of the tree (i.e. depth and nodes).
   - Memory-efficient update.

2. Finding the hyperplane $\mu$.
   - Based on confidence-weighted classifiers [crammer2008, etc.].
   - Conservative and partial update.
Basic Idea

Formulate the sequence prediction problem as linear separation problem.

- Predictive function is mapped to a hyperplane $\mu$ in Hilbert space.
- The context $x^{[1:t-1]}$ is mapped to a vector $\psi$ in Hilbert space.
- Prediction is made by $\mu^k \cdot \psi^t$.

1. Representing $x^{[1:t-1]}$ as a vector $\psi$
   - Growing a context tree.
   - Bounding the size of the tree (i.e. depth and nodes).
   - Memory-efficient update.

2. Finding the hyperplane $\mu$.
   - Based on confidence-weighted classifiers [crammer2008, etc.].
   - Conservative and partial update.
Representing Context

On round $t$, the context $x^{[1:t-1]}$ is observed. Map this sequence to the function $\psi \in \mathcal{H}$ as follows

$$\psi(s^{[1:i]}) := \begin{cases} 
1 & \text{if } s^{[1:i]} = \epsilon \\
e^{-\rho i} & \text{if } s^{[1:i]} \in \text{suf}(x^{[1:t-1]}) \\
0 & \text{otherwise}
\end{cases}$$

- $\text{suf}(x^{[1:t-1]})$: all suffixes of $x^{[1:t-1]}$;
- $\rho > 0$: a predefined hyperparameter to mitigate the effect of long contexts.
Multi-class Confidence Weighted Algorithm

$K$ system calls, i.e. $K$ classes.

Parameters $\{\mu_k, \Lambda_k\}_{k=1}^K$.

Prediction:

$$\hat{x}[t] := \arg \max_{k \in \Sigma} \mu_k[t] \cdot \psi[t].$$

Update policy:

$$\left(\mu_k^{[t+1]}, \Lambda_k^{[t+1]}\right) = \arg \min_{\mu, \Lambda} D_{KL} \left( \mathcal{N}(\mu, \Lambda) \parallel \mathcal{N}(\mu_k^{[t]}, \Lambda_k^{[t]}) \right)$$

s.t. $\Pr_{w \sim \mathcal{N}(\mu, \Lambda)} \left[ w_r \cdot \psi[t] \geq w \cdot \psi[t] \right] \geq \eta$.

On each round, only update the true class and the highest ranked wrong class.
Multi-class Confidence Weighted Algorithm

$K$ system calls, i.e. $K$ classes.

Parameters $\{\mu_k, \Lambda_k\}_{k=1}^K$.

Prediction:

$$\hat{x}[t] := \arg \max_{k \in \Sigma} \mu_k^{[t]} \cdot \psi[t].$$

Update policy:

$$\left(\mu_k^{[t+1]}, \Lambda_k^{[t+1]}\right) = \arg \min_{\mu, \Lambda} \text{D}_{KL} \left(\mathcal{N} (\mu, \Lambda) \parallel \mathcal{N} (\mu_k^{[t]}, \Lambda_k^{[t]})\right)$$

s.t. \[\Pr_{\mathbf{w} \sim \mathcal{N}(\mu, \Lambda)} \left[ \mathbf{w}_r \cdot \psi[t] \geq \mathbf{w} \cdot \psi[t] \right] \geq \eta.\]

On each round, only update the true class and the highest ranked wrong class.
Incorporation of Side Information

Side information: augments, return values, etc.

\( b^{[t]} \in \mathbb{R}^B \): side information on round \( t \).

Incorporate it into the prediction via a linear combination:

\[
\hat{x}^{[t]} := \arg \max_{k \in \Sigma} \mu_k^{[t]} \cdot \psi^{[t]} + \gamma_k^{[t]} \cdot b^{[t]}.
\]

This equivalent to replacing \( \psi^{[t]} \) as a \((Q + B)\)-dimensional vector \([\psi^{[t]}, b^{[t]}]\).
## Side Information

<table>
<thead>
<tr>
<th>Feature set</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File descriptor</td>
<td>2</td>
<td>The number of opened files and the number of closed files, respectively.</td>
</tr>
<tr>
<td>File type</td>
<td>9</td>
<td>Each element represents the number of opened files of a particular type, such as RDONLY, WRONLY, APPEND, etc.</td>
</tr>
<tr>
<td>Functional group</td>
<td>9</td>
<td>Each element represents the number of occurrences of system calls associated with a group given a context. The groups were built in advance by categorizing similar system calls together, resulting 9 groups in total. For instance, the “file” group includes creat, open, close, read, etc. The “process” group includes fork, wait, exec, etc. The “signal” group includes signal, kill, alarm, etc.</td>
</tr>
<tr>
<td>Access location</td>
<td>12</td>
<td>Each element represents the number of accesses to a particular directory, such as /usr/bin, /usr/lib, /usr/tmp, etc.</td>
</tr>
<tr>
<td>Error code</td>
<td>124</td>
<td>Each element represents the number of caught errors of each code, such as ENOENT, EAGAIN, EBGDF, etc.</td>
</tr>
<tr>
<td>POSIX signal</td>
<td>28</td>
<td>Each element represents the number of sent signals of each type, such as SIGSEGV, SIGABRT, SIGBUS etc.</td>
</tr>
<tr>
<td>String character</td>
<td>256</td>
<td>Each element represents the frequency value of a string character. A char is considered as an 8-bit value, resulting 256 possible characters. We only count characters in the string that is not file path.</td>
</tr>
</tbody>
</table>
Other Techniques

- Context tree pruning;
- Feature selection ;
- Extremely conservative update;
- Covariance matrix reset for nonstationary data;
- Efficient implementation.

See our paper.
Experiment Setup

Three sets of data

- BSM (Basic Security Module) data portion of 1998 DARPA intrusion detection;
- System call data set from University of New Mexico;
- Home made: all executable files on Linux.

<table>
<thead>
<tr>
<th>Data set</th>
<th># calls</th>
<th># seq.</th>
<th>Min. len.</th>
<th>Max. len.</th>
<th>Avg. len.</th>
</tr>
</thead>
<tbody>
<tr>
<td>darpa</td>
<td>243</td>
<td>200</td>
<td>2</td>
<td>3,074</td>
<td>57</td>
</tr>
<tr>
<td>lpr1</td>
<td>182</td>
<td>2,766</td>
<td>82</td>
<td>59,565</td>
<td>1,080</td>
</tr>
<tr>
<td>lpr2</td>
<td>182</td>
<td>1,232</td>
<td>74</td>
<td>39,306</td>
<td>449</td>
</tr>
<tr>
<td>sendmail1</td>
<td>190</td>
<td>8,000</td>
<td>8</td>
<td>173,664</td>
<td>669</td>
</tr>
<tr>
<td>sendmail2</td>
<td>190</td>
<td>8,000</td>
<td>8</td>
<td>149,616</td>
<td>648</td>
</tr>
<tr>
<td>stide1</td>
<td>164</td>
<td>8,000</td>
<td>225</td>
<td>146,695</td>
<td>1,055</td>
</tr>
<tr>
<td>stide2</td>
<td>164</td>
<td>8,000</td>
<td>108</td>
<td>174,401</td>
<td>1,255</td>
</tr>
<tr>
<td>ubuntu</td>
<td>458</td>
<td>1,218</td>
<td>2</td>
<td>53,247</td>
<td>952</td>
</tr>
</tbody>
</table>
Baselines

- interpolated Kneser-Ney (IKN) \cite{chen1996}
- online prediction suffix tree (PST) \cite{dekel2009}
- sequence memoizer (SM) \cite{wood2011}
- learning experts (LEX) \cite{eban2012}
## Predictive Performance

Online accumulative error

<table>
<thead>
<tr>
<th>Data set</th>
<th>EOSP</th>
<th>EOSP&lt;sub&gt;s&lt;/sub&gt;</th>
<th>IKN</th>
<th>PST</th>
<th>SM</th>
<th>LEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>darpa</td>
<td>50.11</td>
<td>48.17</td>
<td>52.14</td>
<td>49.25</td>
<td>49.75</td>
<td>51.11</td>
</tr>
<tr>
<td>lpr1</td>
<td>41.63</td>
<td>41.53</td>
<td>41.09</td>
<td>46.24</td>
<td>40.88</td>
<td>42.27</td>
</tr>
<tr>
<td>lpr2</td>
<td>47.44</td>
<td>47.03</td>
<td>47.61</td>
<td>48.52</td>
<td>47.24</td>
<td>51.15</td>
</tr>
<tr>
<td>sendmail1</td>
<td>33.47</td>
<td>34.26</td>
<td>35.62</td>
<td>33.65</td>
<td>33.06</td>
<td>36.81</td>
</tr>
<tr>
<td>sendmail2</td>
<td>33.11</td>
<td>33.91</td>
<td>33.52</td>
<td>34.17</td>
<td>32.19</td>
<td>38.96</td>
</tr>
<tr>
<td>stide1</td>
<td>8.34</td>
<td>8.29</td>
<td>8.54</td>
<td>8.59</td>
<td>8.41</td>
<td>9.06</td>
</tr>
<tr>
<td>stide2</td>
<td>7.75</td>
<td>7.75</td>
<td>8.09</td>
<td>7.95</td>
<td>7.78</td>
<td>8.51</td>
</tr>
<tr>
<td>ubuntu</td>
<td>40.90</td>
<td>36.13</td>
<td>38.90</td>
<td>39.23</td>
<td>75.26</td>
<td>52.72</td>
</tr>
</tbody>
</table>
Scalability

Time cost

Figure: Time cost in second (averaged over 10 runs) of different algorithms. Both axes are in logarithmic scale.
Scalability

Memory consumption

Figure: Memory consumption (averaged over 10 runs) of different algorithms. Both axes are in logarithmic scale.
Conclusions

- Efficient online sequence prediction for system calls traces.
- Incorporation of side information.
- Good scalability on large data sets.
- C implementation available on http://home.in.tum.de/~xiaoh.