A Geometric Framework for
Unsupervised Anomaly Detection:
Detecting Intrusions in Unlabeled Data

Eleazar Eskin, Andrew Arnold,
Michael Prerau, Leonid Portnoy, Sal Stolfo
Department of Computer Science, Columbia University
A chapter in “Applications of Data Mining
in Computer Security”

Agenda

• Introduction
• Feature spaces and kernels
• Detection algorithms
• Experiments
Introduction (1)

• Intrusion detection systems (IDS)
  – detects malicious activities such as DoS attacks, port scans, etc.
    by monitoring network traffic and/or system activities
• Most deployed systems use signature-based detection
  – compares feature values to attack signatures provided by experts
  – need a new signature for every new attack
• Supervised machine learning approach
  – train a classifier using data of normal usage and known attacks
  – can retrain to include more known attacks
  – can detect unknown attacks if perfect model of normality is available

Introduction (2)

• Unsupervised machine learning approach
  – Use unlabeled data as input
  – Attempt to separate the anomalous instances from the normal instances
  – Separate the two classes of instances, then train a traditional anomaly detection algorithm on the clean data
• Geometric framework for unsupervised anomaly detection
  – Map the data to a \( R^d \) feature space
  – Label points in the sparse regions of the feature space as anomalies
Unsupervised Anomaly Detection

• Detect intrusions in unlabeled data
• Can be used to semi-automate manual inspection of data in forensic analysis
• Two main assumptions:
  1) The number of normal instances vastly outnumbers the number of anomalies
  2) The anomalies are qualitatively different from the normal instances

Feature Space (1)

• Instances $x_1, x_2, \ldots, x_n$ in input space $X$
  – for example, X can be the space of all possible network connection records, event logs, system call traces, etc.
• Define a mapping
  $$\varphi : X \rightarrow Y$$
  where $Y$ is $\mathbb{R}^d$, or more generally a Hilbert space
• Define the distance between $x_1$ and $x_2$:
  $$d_{\varphi}(x_1, x_2)$$
  $$= \| \varphi(x_1) - \varphi(x_2) \|$$
  $$= \sqrt{\langle \varphi(x_1), \varphi(x_1) \rangle - 2 \langle \varphi(x_1), \varphi(x_2) \rangle + \langle \varphi(x_2), \varphi(x_2) \rangle}$$
Feature Space (2)

• Some mappings correspond to a kernel function $K$ where

$$ K_{\phi}(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle $$

• Some kernel functions correspond to an infinite dimension mapping!

• Example (radial basis kernel):

$$ K_{rb}(x_1, x_2) = e^{-\frac{||x_1 - x_2||^2}{\sigma^2}} $$

  - This kernel corresponds to an infinite dimension mapping

Detection Algorithms (1)

Algorithm 1: Cluster-based Estimation

  - computes how many points are “near” each point in the feature space, i.e., $d(x1,x2) \leq w$

• Straightforward computation takes $O(n^2)$

• Fixed width clustering algorithm (approximation)

  - First point is center of the first cluster
  - For each subsequent point, if it is within $w$ of some cluster’s center, it is added to that cluster. Otherwise, it is a center of a new cluster
  - Some points may be added to multiple clusters!
  - Complexity is $O(cn)$ where $c$ is the number of clusters
Detection Algorithms (2)

- For points in dense regions, the estimate is inaccurate
  - not an issue, as long as we classify it as a normal point
- For points in sparse regions, the estimate is (more) accurate

Detection Algorithms (3)

Algorithm 2: K-nearest neighbors

- computes the sum of the distance to the $k$-nearest neighbors of each point in the feature space
- Refer to this sum as k-NN score

- To be useful, $k$ needs to be larger than the number of instances of any one attack
- Straightforward computation: $O(n^2)$
Detection Algorithms (4)

- Approximation algorithm
  - Use the fixed-width cluster algorithm from algorithm 1, with a variation that each element is only placed in one cluster
- Let $c(x)$ denote the center of the cluster that contains $x$
- If $x_1$ and $x_2$ are in the same cluster:
  $$d_\phi(x_1, x_2) \leq 2w$$
- In all cases:
  $$d_\phi(x_1, x_2) \leq d_\phi(x_1, c(x_2)) + w$$
  $$d_\phi(x_1, x_2) \geq d_\phi(x_1, c(x_2)) - w$$

Detection Algorithms (5)

- The algorithm uses these three inequalities to find the $k$-nearest neighbors
- Note that choice of $w$ does not affect the k-NN score; it only affects the efficiency of computing the score
Detection Algorithms (6)

Algorithm 3: One Class SVM (Support Vector Machine)
- Standard SVM algorithm is a supervised learning algorithm
  - tries to maximally separate two classes of data in feature space by a hyperplane
- Unsupervised (one class) SVM tries to separate the entire set of training data from the origin
- Objective function:

\[
\min_{w \in Y, \zeta_i \in \mathbb{R}, \rho \in \mathbb{R}} \frac{1}{2} |w|^2 + \frac{1}{\nu} \sum_{i=1}^{l} \zeta_i - \rho \\
\text{subject to: } (w \cdot \phi(x_i)) \geq \rho - \zeta_i, \zeta_i \geq 0
\]

\(\nu\): parameter that controls tradeoff between maximizing the distance from origin and containing most of the data (essentially the ratio of expected anomalies in the dataset)

\(l\): the number of data points

\(w\): the hyperplane’s normal vector in the feature space

\(\rho\): the origin

\(\zeta\): slack variables

Experiments (1)

- Two datasets:
  - KDD Cup 1999
    - Wide variety of simulated intrusions in a military network environment
    - 4,900,000 instances
    - The features are extracted from connection records
      - examples are duration, protocol type, number of bytes transferred, normal or error status of the connection
    - Four categories of attacks (total 24 attack types):
      1) Denial of Service (e.g. syn flood)
      2) Unauthorized access from a remote machine (e.g. password guessing)
      3) Unauthorized access to superuser functions (e.g. buffer overflow attacks)
      4) Probing (e.g. port scanning)
    - Filter out attacks so that dataset consists of 1-1.5% attacks
Experiments (2)

- System call data from BSM (Basic Security Module) portion of 1999 DARPA Intrusion Detection Evaluation data
  - consists of 5 weeks of BSM data of all processes on a Solaris machine
  - consider two programs: eject and ps
  - An attack can correspond to multiple processes because a malicious process can spawn other processes. Consider an attack detected if one of the processes corresponding to the attack is detected

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Total # of Attacks</th>
<th># Intrusion Traces</th>
<th># Intrusion System Calls</th>
<th># Normal Traces</th>
<th># Normal System Calls</th>
<th>% Intrusion Traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>ps</td>
<td>3</td>
<td>21</td>
<td>996</td>
<td>208</td>
<td>35092</td>
<td>2.7%</td>
</tr>
<tr>
<td>eject</td>
<td>3</td>
<td>6</td>
<td>726</td>
<td>7</td>
<td>1278</td>
<td>36.3%</td>
</tr>
</tbody>
</table>

Experiments (3)

- The datasets are divided into two parts: one for training and one for testing
- Parameters:
  - cluster-based algorithm:
    - For network connection data, the width is 40
    - For eject system call traces, the width is 5
    - For ps traces, the width is 10
  - k-nearest neighbor algorithm:
    - For network connection data, \( k = 10,000 \)
    - For eject system call traces, \( k = 2 \)
    - For ps traces, \( k = 15 \)
  - SVM-based algorithm:
    - For network connection data, \( \nu = 0.01 \) and \( \sigma^2 = 12 \)
    - For system call traces, \( \nu = 0.05 \) and \( \sigma^2 = 1 \)
Experiments (4)

- Result for KDD cup data (network connection data)

```
\begin{figure}
\centering
\includegraphics[width=\textwidth]{roc_curves}
\caption{ROC Curves for KDD Cup 1999 Data Set}
\end{figure}
```

Experiments (5)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Detection rate</th>
<th>False positive rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td>93%</td>
<td>10%</td>
</tr>
<tr>
<td>Cluster</td>
<td>66%</td>
<td>2%</td>
</tr>
<tr>
<td>Cluster</td>
<td>47%</td>
<td>1%</td>
</tr>
<tr>
<td>Cluster</td>
<td>28%</td>
<td>.5%</td>
</tr>
<tr>
<td>K-NN</td>
<td>91%</td>
<td>8%</td>
</tr>
<tr>
<td>K-NN</td>
<td>23%</td>
<td>6%</td>
</tr>
<tr>
<td>K-NN</td>
<td>11%</td>
<td>4%</td>
</tr>
<tr>
<td>K-NN</td>
<td>5%</td>
<td>2%</td>
</tr>
<tr>
<td>SVM</td>
<td>98%</td>
<td>10%</td>
</tr>
<tr>
<td>SVM</td>
<td>91%</td>
<td>6%</td>
</tr>
<tr>
<td>SVM</td>
<td>67%</td>
<td>4%</td>
</tr>
<tr>
<td>SVM</td>
<td>5%</td>
<td>3%</td>
</tr>
</tbody>
</table>

Table 2: Selected points from the ROC curves of the performance of each algorithm over the KDD Cup 1999 Data.
Experiment (6)

- For system call traces, **all three** algorithms perform perfectly (100% accuracy)!
- Features used: sub-sequences of system calls
  - Malicious process traces have many sub-sequences that do not occur in normal processes
  - Normal processes have similar sub-sequences of system calls