Using Affinity Perturbations to Detect Web Traffic Anomalies

Yaniv Shmueli
School of
Computer Science
Tel Aviv University
yaniv.shmueli@cs.tau.ac.il

Tuomo Sipola
Department of
Mathematical Information Technology
University of Jyväskylä
tuomo.sipola@jyu.fi

Gil Shabat
School of
Electrical Engineering
Tel Aviv University
gil@eng.tau.ac.il

Amir Averbuch
School of
Computer Science
Tel Aviv University
amir@math.tau.ac.il

***Abstract*—The initial training phase of machine learning algorithms is usually computationally expensive as it involves the processing of huge matrices. Evolving datasets are challenging from this point of view because changing behavior requires updating the training. We propose a method for updating the training profile efficiently and a sliding window algorithm for online processing of the data in smaller fractions. This assumes the data is modeled by a kernel method that includes spectral decomposition. We demonstrate the algorithm with a web server request log where an actual intrusion attack is known to happen. Updating the kernel dynamically using a sliding window technique, prevents the problem of single initial training and can process evolving datasets more efficiently.**

***Index Terms*—perturbation theory, eigenvalue problem, diffusion maps, dimensionality reduction, anomaly detection, web traffic**

I. Introduction

Evolving data that requires frequent updates to the training is a challenging target when extracting constructive information. The computational complexity of the training phase increases with such datasets because an earlier profile may not accurately represent the behavior of current data. Therefore, the extracted profile has to be updated frequently. A straightforward approach for updating the training profile is to repeat the entire computational process that generated the original profile. This paper summarizes a method for efficiently updating the evolving profile.

A common practice in kernel methods is to extract features from a high dimensional dataset, and to form a similarity graph between the features in the dataset. In this research we apply the Diffusion Maps (DM) methodology [1] to a web traffic log. DM finds the embedded coordinates for a low-dimensional representation of the data. This embedding is accomplished by eigenvectors computation of the graph affinity matrix. Changes in the affinity matrix will result in changes in the eigenvectors, and thus will force us to compute them frequently. We use a solution based on the Recursive Power Iteration algorithm combined with the first-order approximation of the perturbed eigenvectors and eigenvalues (eigenpairs) [2]. This enables us to update the dataset profile by considering only the changes in the original dataset, which also requires less computational effort.

Since network data is dynamic and evolving, the embedded low-dimensional space has to be updated as the training data does not adequately represent the incoming data that did not participate in the initial training phase. Even if most of the network log lines in our interest window are unchanged, we will still need to perform the entire computation since we cannot determine the effect of such a change on the embedded space. Therefore, the goal of the paper is to provide an efficient method for updating the embedding coordinates without the need to re-compute the entire SVD again and again over time. We treat the log line feature changes as perturbations from the original network log profile of the feature affinity matrix. By applying a sliding window technique to the incoming network data, we are able to process the data online, and keep embedding it in the low-dimensional space. We test this method on real web traffic data and compare our results to the true classification.

II. Related Work

Traditional computational methods such as the power iteration, inverse iteration and Lanczos methods operate in the same way and compute the eigenpairs of each update of the perturbed matrix. Here, the computation is performed with a random guess as the initial input without taking the unperturbed matrix and its eigenpairs into consideration.

Incremental versions of low-dimensional embedding algorithms have been tailored specifically to fit local linear embeddings (LLE) [3] and ISOMAP [4]. These algorithms use modified manifold learning methods to process the data iteratively. When a new data point arrives, these algorithms add it to the embedding and then efficiently update all the existing data points in the low-dimensional space.

Network security has been one focus among the machine learning community. Kruegel and Vigna studied the parameters of HTTP queries using a training step with unlabeled data with various methods. Their character distribution analysis uses similar feature extraction as our current study [5]. Hubballi et al. described an n-gram approach to detect intrusions from network packets [6]. Ringberg et al. studied IP packets using principal component analysis-based dimensionality reduction [7]. Callegari et al. analyzed similar low-level packet data [8].

Diffusion maps have been also used for network security problems. David studied the use of diffusion map methodology for detecting intrusions in network traffic [9]. Network server logs have also been studied with diffusion maps with an offline approach using n-gram features and spectral clustering [10]. In these works, data analysis was performed in a batch fashion, processing all recordings as a single, offline dataset.

III. Finding a Low-Dimensional Embedded Space

A. Diffusion Maps

Finding a low-dimensional embedded space is an important step in understanding high-dimensional data more profoundly. To better understand the proposed algorithm, we review the DM methodology [1] that performs non-linear dimensionality reduction. Given our web log feature matrix X, we define a weighted graph over the log lines, where the weight between lines i and j is given by the kernel $k(i,j) \triangleq e^{-\frac{\|x_i - x_j\|}{\varepsilon}}$. The degree of a log line (vertex) i in this graph is $d(i) \triangleq \sum_j k(i,j)$. Normalizing the kernel with this degree produces an $n \times n$ row stochastic transition matrix whose cells are $[P]_{ij} = p(i,j) = k(i,j)/d(i)$ for log lines i and j. This defines a Markov process over the network log features.

The dimensionality reduction achieved by this diffusion process is a result of the spectral analysis of the kernel. Thus, it is preferable to work with a symmetric conjugate to P that we denote by A and its cells are denoted by

$$[A]_{ij} = a(i,j) = \frac{k(i,j)}{\sqrt{d(i)}\sqrt{d(j)}} = \sqrt{d(i)}p(i,j)\frac{1}{\sqrt{d(j)}}. \quad (1)$$

The eigenvalues $1 = \lambda_1 \geq \lambda_2 \geq \ldots$ of P and their corresponding eigenvectors v_k $(k = 1, 2, \ldots)$ are derived from the eigenvectors u_k of A. The v_k are used to obtain the desired dimensionality reduction by mapping each i onto the data point $\Psi(i) = (\lambda_2 v_2(i), \lambda_3 v_3(i), ..., \lambda_\delta v_\delta(i))$ for a sufficiently small δ, which depends on the decay of the spectrum of A [1].

In matrix notation, the operator A is defined as $A = D^{-\frac{1}{2}} K D^{-\frac{1}{2}} = D^{\frac{1}{2}} P D^{-\frac{1}{2}}$ where D is the diagonal matrix containing the $d(i)$ value in cell D_{ii}. To retrieve the eigenvectors in columns V of P from the eigenvactors of A, we use the transformation $V = D^{-\frac{1}{2}} U$ where U is the eigenvector column matrix of A. The eigenvectors V obtained for P are scaled by dividing each one by the first value of the first eigenvector.

B. Updating the Embedding

Once we have the DM embedding of the initial matrix A, we need to keep updating the embedding for the next arriving samples. By replacing the oldest samples with the newly arriving ones, we can model such a change as a perturbation matrix $\tilde{A}$ of the matrix A. We assume that the perturbations are sufficiently small, that is, $\|\tilde{A} - A\| < \varepsilon$ for some small ε. Note that $\tilde{A}$ is symmetric since it represents the operator defined in 1. We wish to update the eigenpairs of $\tilde{A}$ based on A and its eigenpairs. We now present the problem in mathematical terms.

Given a symmetric $n \times n$ matrix A where its k dominant eigenvalues are $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_k$ and its eigenvectors are $\phi_1, \phi_2, ..., \phi_k$, respectively, and a perturbed matrix $\tilde{A}$ such that $\|\tilde{A} - A\| < \varepsilon$, find the perturbed eigenvalues $\tilde{\lambda}_1 \geq \tilde{\lambda}_2 \geq ... \geq \tilde{\lambda}_k$ and its eigenvectors $\tilde{\phi}_1, \tilde{\phi}_2, ..., \tilde{\phi}_k$ of $\tilde{A}$ in the most efficient way [2].

In the next section, we explain how such processing can be done using the recursive power iteration (RPI) algorithm.

IV. The Recursive Power Iteration (RPI) Algorithm

A. Eigenpair First-Order Approximation

To efficiently update each eigenpair of the perturbed matrix $\tilde{A}$, we first compute the first-order approximation of each eigenpair. Later, it will be used in our algorithm as the initial guess for the RPI algorithm.

Given an eigenpair (ϕ_i, λ_i) of a symmetric matrix A where $A\phi_i = \lambda_i \phi_i$, we compute the first-order approximation of the eigenpair of the perturbed matrix $\tilde{A} = A + \Delta A$. We assume that the change ΔA is sufficiently small, which results in a small perturbation in ϕ_i and λ_i. We look for $\Delta\lambda_i$ and $\Delta\phi_i$ that satisfy the equation

$$(A + \Delta A)(\phi_i + \Delta\phi_i) = (\lambda_i + \Delta\lambda_i)(\phi_i + \Delta\phi_i). \quad (2)$$

Using the methods described by Shmueli et al. [2], we can obtain the following first-order approximations for the eigenvalues and eigenvectors of $\tilde{A}$

$$\tilde{\lambda}_i = \lambda_i + \phi_i^T [\Delta A] \phi_i \quad (3)$$

and

$$\tilde{\phi}_i = \phi_i + \sum_{j \neq i} \frac{\phi_j^T [\Delta A] \phi_i}{\lambda_i - \lambda_j} \phi_j. \quad (4)$$

B. The Recursive Power Iteration Method

The power iteration method has proved to be effective when calculating the principal eigenvector of a matrix [11]. However, this method cannot find the other eigenvectors of the matrix. In general, an initial guess of the eigenvector is also important to guarantee fast convergence of the algorithm. In Algorithm IV.1, which we call recursive power iteration (RPI), the first-order approximations of the perturbed eigenvectors of $\tilde{A}$ are the initial guess for each power iteration. Once the eigenvector $\tilde{\phi}_i$ is obtained in step i, we transform $\tilde{A}$ into a matrix that has $\tilde{\phi}_{i+1}$ as its principal eigenvector. We iterate this step until we recover the k dominant eigenvectors of $\tilde{A}$.

The correctness of the RPI algorithm and its complexity analysis are given in the original article [2].

The justification for this approach is that the first-order approximation of the perturbed eigenvector is inexpensive, and each RPI step guarantees that this approximation converges to the actual eigenvector of $\tilde{A}$. The first-order approximation should be close to the actual solution we seek and therefore requires fewer iteration steps to converge.

Algorithm IV.1: Recursive Power Iteration Algorithm

Input: Perturbed symmetric matrix $\tilde{A}_{n\times n}$, number of eigenvectors to calculate k, initial eigenvectors guesses $\{v_i\}_{i=1}^{k}$, admissible error *err*

Output: Approximated eigenvectors $\left\{\tilde{\phi}_i\right\}_{i=1}^{k}$, approximated eigenvalues $\left\{\tilde{\lambda}_i\right\}_{i=1}^{k}$

for $i = 1 \rightarrow k$ **do**
  $\phi \leftarrow v_i$
  repeat
    $\phi_{next} \leftarrow \frac{\tilde{A}\phi}{\|\tilde{A}\phi\|}$
    $err_\phi \leftarrow \|\phi - \phi_{next}\|$
    $\phi \leftarrow \phi_{next}$
  until $err_\phi \leq err$
  $\tilde{\phi}_i \leftarrow \phi$
  $\tilde{\lambda}_i \leftarrow \frac{\tilde{\phi}_i^T \tilde{A} \tilde{\phi}_i}{\tilde{\phi}_i^T \tilde{\phi}_i}$
  $\tilde{A} \leftarrow \tilde{A} - \tilde{\phi}_i \tilde{\lambda}_i \tilde{\phi}_i^T$
end for

V. Sliding Window Diffusion Map

Using DM to embed high volumes of data can be computationally intensive. It is even more challenging when the data is generated online and needs to be processed continuously. Therefore, we try to process the incoming data with iterative methodology by using the sliding window model. A sliding window X takes into account the n latest measurements. In practice, it is an $n \times m$ matrix with features on the columns and samples on the rows. The samples are high-dimensional, so the dimensionality of the sliding window is reduced from m to d using DM. This $n \times d$ matrix X_r now contains the low-dimensional representation of the data. This reduction is done each time a new sample appears and the window moves. However, the consecutive update of the DM is a time-consuming process that requires singular value decomposition during each window.

When updating the window, we can replace the oldest measurement with a new one in the matrix X, therefore changing a single row in X. This means that one line and one column of the K matrix in the DM algorithm change. This change can be interpreted as a perturbation to the matrix K, and furthermore to the matrix A, which is defined using the K matrix. The RPI algorithm with first-order approximation solves the eigenvectors for perturbed matrices. This leads us to use the RPI algorithm instead of time-consuming SVD.

Algorithm V.1 outlines the sliding window DM method. First, it solves the eigenvectors for the initial window using SVD. Then the algorithm iteratively process the following windows until no new samples are available.

There are, some practical problems with this approach. First, the RPI algorithm might not be able to solve the eigenvectors for some low-rank matrices. It is possible to prevent this with standard SVD when a low-rank (or otherwise unsuitable) matrix is encountered. Second, the window size itself has to be decided. The changing scales of the data over time introduce a challenge to the sliding window algorithm. The initial window still determines the profile and scale for the beginning of the analysis. Big windows cover a larger representation of the data and thus include a more varied overview of the normal behavior. With smaller windows, the percentage of anomalies within the data might get too big, and detecting the normal state becomes more difficult. Small windows, however, require less computational time since they induce smaller matrices. Optimal window size would therefore be the smallest possible that contains a small enough percentage of anomalies within the data, enabling it to capture the normal samples correctly.

Algorithm V.1: Sliding Window Diffusion Map with RPI

Input: Dataset X, window width n, embedded dimension k, admissible error err.

Output: Anomaly score for points in X.

$\epsilon \leftarrow$ estimate kernel parameter for first window of size n.
$[K]_{ij} \leftarrow \exp\left(-\frac{||x_i - x_j||^2}{\epsilon}\right)$, where $i, j = 1 \dots n$
$D \leftarrow \mathrm{diag}(\sum_{i=1}^{n}[K]_{ij})$
$A \leftarrow D^{-\frac{1}{2}} K D^{-\frac{1}{2}}$
$U, \Lambda, U^T \leftarrow \mathrm{SVD}(A)$
while new sample x_t available, where $t > n$ **do**
  $l \leftarrow t \bmod n$
  Replace the row l in X with the new sample x_t.
  Update both row l and column l of the affinity matrix K.
  $D \leftarrow \mathrm{diag}(\sum_{i=1}^{n}[K]_{ij})$
  $\tilde{A} \leftarrow D^{-\frac{1}{2}} K D^{-\frac{1}{2}}$
  $U, \Lambda \leftarrow$ RPI with first-order approximation $(\tilde{A}, A, k, U, \Lambda, err)$
  $V \leftarrow D^{-\frac{1}{2}} U$
  $V \leftarrow \frac{V}{V_{1,1}}$
  $\Psi \leftarrow V\Lambda$
  Find anomalies in Ψ and rate all samples in X.
  $A \leftarrow \tilde{A}$
end while
Return aggregated anomaly scores for each sample in X.

Detecting the anomalies in the low-dimensional representation can be done in various ways. A straightforward approach is to calculate distances between the embedded samples and find the ones that deviate too far from the center of the dataset. This and other spectral clustering methods give good results for datasets that contain clear separation [12], [10]. Similarly, k-means or any other clustering algorithm can find possible normal as well as anomalous behavior in the data. The density of points in the low-dimensional space tells how far they are from the more clustered areas. These methods calculate the distances to neighboring points [9]. All these methods usually need a threshold value for the anomalous region.

In each iteration, we evaluate the anomaly level of the samples within the window. Each sample gets a score if it is classified as an anomaly according to the selected anomaly

detection method. The scores of each sample are added as the window moves. This cumulative anomaly score histogram may be used to determine the anomaly level of a point. Scoring is used because locally inside a window some samples might appear anomalous but globally, considering the whole dataset, they are not. Even if the sample looks like an anomaly in some windows, it still gets only a few scores globally.

VI. Experimental Results

For the experiment, we use a labeled proprietary dataset of queries to a web server, which is known to contain some network attacks. These web queries are in Apache combined log text file. To extract numerical features from this text file, only the changing parameter values are used. The frequencies of 2-grams in these parameters are calculated to a matrix. In this matrix, the rows represent the log lines, and the columns represent the different 2-grams we found. The entries in this matrix count how many times each specific 2-gram appeared in the parameters of a log line. See [10] for more information about this dataset and the feature extraction.

The web log we use has 4292 lines and contains 480 different 2-grams. Thus, the feature matrix has dimensions 4292×480. The experiment simulates the initial state when n samples, or log lines, have arrived. When a new line arrives, it is added to the current window, while the oldest sample is removed from the matrix. This is continued until no new samples are available. The algorithm tracks only the samples within the window so that the dynamically changing nature of the data can be followed. As the size of the window does not change, the eigenpair problem stays reasonably sized.

Anomaly detection with Euclidian distances finds the most deviating samples within a window. This leads to false alarms when using simple normalized anomaly metrics because inside a window a point might look anomalous. Its local abnormality might be evident, but it should not be classified as one since globally it is just a small deviation from the normal state. This fact promotes thresholding the non-normalized but centered low-dimensional representation $d_k = |\Psi_k - \mathrm{mean}(\Psi_k)|$ within one window using statistical threshold $\theta_k = c \cdot \mathrm{std}(d_k)$, where the parameter c has to be adjusted empirically, for each dimension k in the embedded space.

Figure 1 illustrates the scores each point gets as the sliding window moves. The number of times the data points are classified anomalous are plotted against time. The window width is set to $n = 1000$. This experiment uses only the second eigenpair, $k = 2$, Ψ_2 for the low-dimensional presentation. In our analysis, we use a value of $c = 10$ for the anomaly threshold calculation. These scores themselves indicate in how many windows each sample is considered anomalous: the data points that are considered attacks are clearly seen from 2500 to 3500. Notice that a sample might be considered anomalous in several windows, but in the global view it is not an anomaly. Therefore, we use another threshold, which is the horizontal red line in the figure. With this setup, we manage to reach an accuracy of 92.5% and a precision of 99.7% after tuning the parameters of the algorithm.

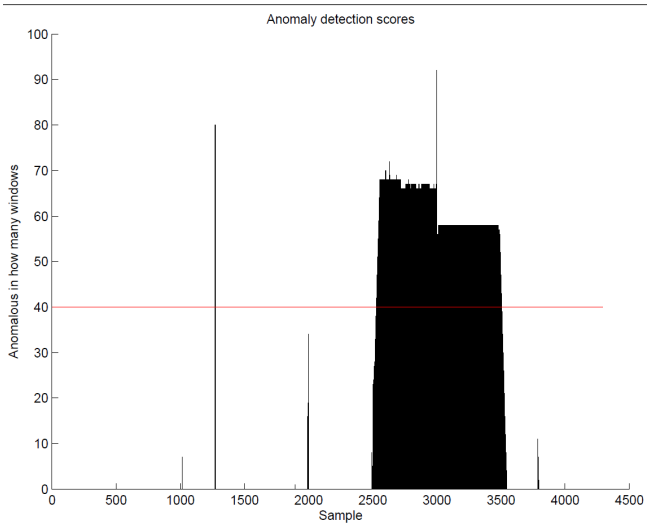

Fig. 1. The scores for each point with window size 1000 using the second eigenvector. The more times the data point is classified anomalous, the higher the score.

Acknowledgments

This research was supported by the Israel Science Foundation (Grant No. 1041/10) and by the Foundation of Nokia Corporation. The authors thank Antti Juvonen for assisting with the data analysis and Tapani Ristaniemi for guidance and support.

References

[1] R. R. Coifman and S. Lafon, "Diffusion maps," *Applied and Computational Harmonic Analysis*, vol. 21, no. 1, pp. 5–30, 2006.

[2] Y. Shmueli, G. Wolf, and A. Averbuch, "Updating kernel methods in spectral decomposition by affinity perturbations," *Linear Algebra and its Applications*, vol. 437, no. 6, pp. 1356–1365, 2012.

[3] O. Kouropteva, O. Okun, and M. Pietikäinen, "Incremental locally linear embedding algorithm," *Image Analysis*, pp. 145–159, 2005.

[4] M. Law and A. Jain, "Incremental nonlinear dimensionality reduction by manifold learning," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 3, pp. 377–391, 2006.

[5] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proceedings of the 10th ACM conference on Computer and communications security*. ACM, 2003, pp. 251–261.

[6] N. Hubballi, S. Biswas, and S. Nandi, "Layered higher order n-grams for hardening payload based anomaly intrusion detection," in *Availability, Reliability, and Security, 2010. ARES'10 International Conference on*. IEEE, 2010, pp. 321–326.

[7] H. Ringberg, A. Soule, J. Rexford, and C. Diot, "Sensitivity of PCA for traffic anomaly detection," *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1, pp. 109–120, 2007.

[8] C. Callegari, L. Gazzarrini, S. Giordano, M. Pagano, and T. Pepe, "A novel PCA-based network anomaly detection," in *Communications (ICC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–5.

[9] G. David, "Anomaly Detection and Classification via Diffusion Processes in Hyper-Networks," Ph.D. dissertation, Tel-Aviv University, 2009.

[10] T. Sipola, A. Juvonen, and J. Lehtonen, "Anomaly detection from network logs using diffusion maps," in *Engineering Applications of Neural Networks*, ser. IFIP Advances in Information and Communication Technology, L. Iliadis and C. Jayne, Eds. Springer Boston, 2011, vol. 363, pp. 172–181.

[11] A. Langville and C. Meyer, "Updating markov chains with an eye on google's pagerank," *SIAM journal on matrix analysis and applications*, vol. 27, no. 4, pp. 968–987, 2006.

[12] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, pp. 395–416, 2007.