Detecting Network Anomalies With Kernel Principal Component Analysis

Kenneth Heafield

May 19, 2006

Abstract

Computer network traffic often exhibits a pattern that results from protocols, applications, and use. Deviation from this pattern may indicate intrusion attempts or faults in the network. A computer monitoring the network should be able to learn traffic patterns and detect deviation for reporting to an administrator. Network packets can be treated as data points in a regression model. Kernel Principal Component Analysis, a type of non-linear regression, is used to learn structure from the data points. The algorithm is extended to compute uncertainties of learned values. Statistics useful in testing for outliers and for significant changes in structure are developed.

1 Introduction

Principal Component Analysis (PCA) has proved to be a popular and useful tool for deducing linear structure of high-dimensional data. Extensions such as kernel PCA allow deduction of non-linear structures. Kernel PCA's strength and weakness is that much of the theory uses very large or infinite dimensional vectors. In practice, a kernel function computes dot products between these vectors. As a result, all computations involving the vectors must be expressed in terms of dot products.

The present work extends kernel PCA in two ways compatible with dot products. First, data points can be very high dimensional and never actually computed. This allows each address to have its own dimension. Second, an error model finds uncertainties in the model. Using these uncertainties, one can statistically test for outliers and differences in structure between samples. These extensions are useful for analyzing traffic on computer networks.

In this paper, we apply kernel PCA to computer network packets. Computer network packets have protocol headers and payloads. Headers usually have the same fields: source and destination address, port number, window size, etc. Addresses are grouped into subnets which often have related computers. The payload is application specific, changes meaning with each packet, and could be encrypted or compressed. Additional information, such as the interface on which a packet arrived and timing, can also be used. This paper uses only the header and additional information, avoiding the interesting problem of interpreting payload.

Network packets are translated into data points by parsing their headers. Some fields, such as window size, make sense to represent as a single variable. Other fields are less easily represented. For example, hosts with consecutive addresses need not be related at all. It may, however, be meaningful that two hosts are in the same subnet. Our approach is to give each address and subnet its own dimension.

Kernel PCA is used to analyze batches of packets. As new packets come in, they are translated and tested against models derived from previous batches. This test compares the packet's disagreement with the model to the model's uncertainty near the packet. When enough packets are available, a new batch is processed. Batches are tested against each other to identify large-scale structural changes. By expiring old batches, the algorithm is able to incorporate slow changes in network use while still identifying attacks or faults.

2 Learning Algorithm

In essence PCA [1] is generalized linear regression. It is multivariate and lacks the concept of a dependent or independent variable. Instead, all variables are treated equally. The predictor is a model's best match for a given data point. Conventional regression compares a data point and its predictor using only the dependent variable to compute a residual. Principal Component Analysis uses all of the variables to compute a vector residual. Both minimize the sum of squared residual lengths. Figure 1 compares principal component analysis with standard linear regression in two dimensions.

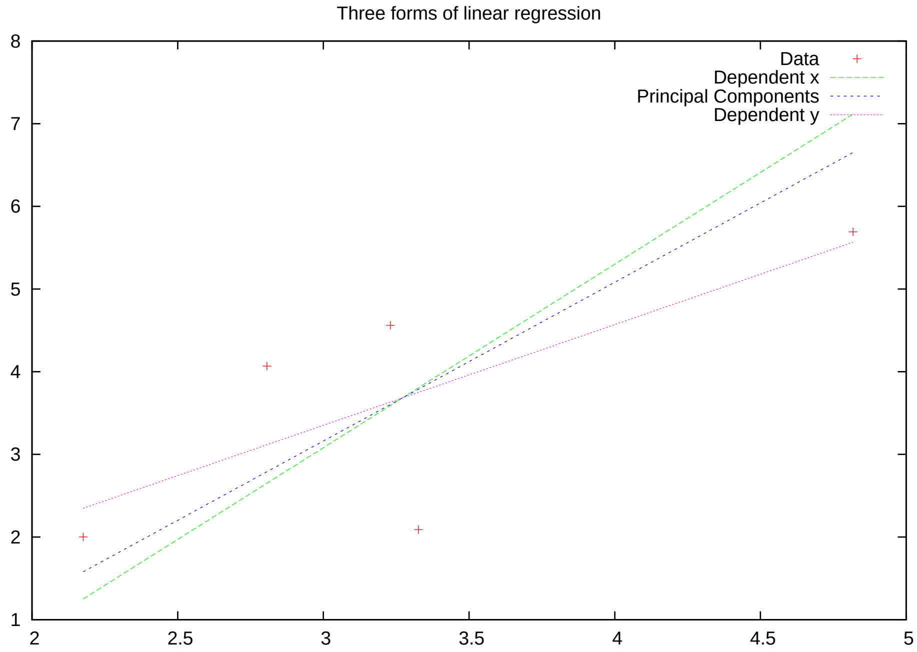

Figure 1: Three forms of linear regression in two dimensions: Principal Components, standard linear regression with the usual dependent y, and standard linear regression with dependent x. Note how PCA lies between the other two. The data are points $(0,0),(1,1),(2,2),(3,3)$, and $(4,4)$ plus normal error with standard deviation 2 in both coordinates. This rather extreme example is intended to illustrate the separation between the three fits.

Kernel PCA [2] is an extension of PCA allowing non-linear regression. Conceptually, it performs a non-linear transformation on the data points. Then PCA is applied to the transformed data. In practice, the PCA algorithm is expressed in terms of dot products of transformed data so that the transformation is never computed. A function called the kernel simultaneously transforms and computes the dot product between to data points. Interestingly, we know where data points lie on the model without ever learning the model itself. In extending kernel PCA, one must bear this fact in mind.

2.1 Principal Component Analysis

Principal Component Analysis [1] can be used to learn the structure of input data. Given n data points X_i in linear space F with dimensionality f, PCA finds a linear subspace that maximally preserves their variance. It does so by computing the sample covariance matrix

$$C = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X})(X_i - \bar{X})^{\mathrm{T}} \tag{1}$$

where $\bar{X}_i$ is the sample mean. The orthonormal [1] eigenvectors

$$\{C_k\}_{k=1}^{n} \tag{2}$$

of C, listed in decreasing order of eigenvalue, are called the principal components. The matrix C is positive definite so all the eigenvalues are non-negative. Using this fact, we define two new spaces and recall one:

- F is the space in which the data points lie. In kernel PCA it is called the feature space.
- P is the subspace of F spanned by the data points. Equivalently, it is spanned by the p eigenvectors with non-zero eigenvalue. It has at most n dimensions.
- M is the subspace of P spanned by the first m eigenvectors where $m \leq p$. This subspace is our variance-preserving fit to the data. For an m-dimensional subspace, it has least sum of squares distance to the data points. Several methods exist for choosing m as discussed in [1].

For each of these spaces, the lower case variable refers to its dimension.

2.2 Kernel Principal Component Analysis

Kernel Principal Component Analysis [2] extends PCA by adding non-linearity. Let n data points x_i lie in $D = \mathbb{R}^d$. Counter intuitively, the first conceptual step expands the dimensionality of the problem. One chooses a feature space F and linearizing function

$$\Phi : D \mapsto F \tag{3}$$

which is used to map data points by

$$X_i = \Phi\left(x_i\right). \tag{4}$$

If the data are a non-linear function of some underlying variables, then Φ attempts to invert that relationship. The idea is that Φ makes many simultaneous attempts at doing so. Unsuccessful attempts will be manifest as noisy variables. Successful attempts will result in large variation in a particular direction. By definition, PCA selects the directions with largest variation. Therefore we perform PCA using the f-dimensional X_i as data points. However, f can be very large or even infinite.

In practice kernel PCA reduces the problem of working in F to working in the subspace P. It does so by formulating PCA in terms of dot products

$$X_i \cdot X_j = \Phi\left(x_i\right) \cdot \Phi\left(x_j\right). \tag{5}$$

The kernel function

$$k\left(x_i, x_j\right) = \Phi\left(x_i\right) \cdot \Phi\left(x_j\right) \tag{6}$$

finds these dot products without computing X_i or X_j. For example,

$$k\left(x_i, x_j\right) = \left(x_i \cdot x_j + 1\right)^r \tag{7}$$

is a kernel for polynomials of degree r. Using the kernel one computes the symmetric positive definite matrix

$$K_{ij} = k(x_i, x_j) \tag{8}$$

and center it. The eigenvalues λ_k and eigenvectors α_k of K allow one to find principal components. Note that $\lambda_k \geq 0$ as K is positive definite. Further if $\lambda_k = 0$ then the variance is zero and the component is uninteresting. The remaining eigenvectors are normalized according to

$$1 = \lambda_k\left(\alpha_k \cdot \alpha_k\right). \tag{9}$$

The principal components are

$$C_k = \sum_{i=1}^{n} \alpha_{ki} X_i \tag{10}$$

where α_{ki} is the ith entry in α_k. However, C_k is f-dimensional and never actually computed. Only the principal components of a point x

$$C_k \cdot \Phi\left(x\right) = \sum_{i=1}^{n} \alpha_{ki}\left(\Phi(x_i) \cdot \Phi(x)\right) \tag{11}$$

are computed using the kernel function k to find the dot product. For a complete derivation, see [2] from which this description was summarized. The point of this practical discussion is that some variables are accessible to enhancements only through dot products.

3 Translating Data Points

Our first extension to kernel PCA is the introduction of data translation function. Conceptually, data points are translated and the kernel PCA algorithm is run on these translated points. In practice, the transformed data points are intractable to compute or store. Instead, the same trick is performed: computing only the dot product between translated points.

Let n data points y_i lie in $E = \mathbb{R}^e$. The data transformation is a function

$$\Psi : E \mapsto D \tag{12}$$

with a tractable analogue of the kernel function

$$l\left(y_i, y_j\right) = \Psi\left(y_i\right) \cdot \Psi\left(y_j\right). \tag{13}$$

The kernel PCA algorithm is concerned only with the value of the kernel function

$$k\left(\Psi\left(y_i\right), \Psi\left(y_j\right)\right). \tag{14}$$

In many cases, the kernel k can be expressed as

$$k\left(\Psi\left(y_i\right), \Psi\left(y_j\right)\right) = k'\left(\Psi\left(y_i\right) \cdot \Psi\left(y_j\right)\right). \tag{15}$$

See (7) for an example of such a kernel. Now we may substitute the function l to obtain

$$k\left(\Psi\left(y_i\right), \Psi\left(y_j\right)\right) = k'\left(l\left(y_i, y_j\right)\right), \tag{16}$$

a tractable function to implement data transformation for kernel PCA.

4 Error Model

We want to estimate error in the models produced by the kernel PCA. The model is defined by a set of principal components. Roughly speaking, these are the directions of greatest variance. Each principal component is assigned an unknown error estimate. The error is assumed to be symmetric about the component. Propagating these uncertainties produces an uncertainty in each predictor point. On average, this should equal the length of the associated residual. Setting them equal, we obtain a linear regression problem with which the component uncertainties are found.

To insure that uncertainty is positive and reflects possible principal components, some constraints are placed on the regression problem. This makes it an inequality constrained linear regression problem. Algorithms to solve such problems are readily available [3]. The outcome is an estimate of each component's uncertainty. All of this is done in such a way that it applies to kernel PCA.

Using uncertainty in each component, two types of tests can be applied. First, we now have an estimate for the uncertainty of each predictor point. Outlier points are identified by comparing their residual with the uncertainty of the predictor. Second, if two separate samples are analyzed then we can test their components against each other. This allows comparison of structure in two data sets.

4.1 Assumptions

When the only information available is the data points, no formal analysis of error is possible [1]. Therefore we must make some assumptions about the problem, starting with the components. Assume that there exist actual principal components A_k and that each random sample of n data points will produce estimates of these components. Matching components with their estimates will be discussed later. Components are constrained to the space P [2] and we assume that they obey a distribution in that space. Let R_k denote a random vector sampled from this distribution. Negation of components does not impact the fit allowing the constraint

$$R_k \cdot A_k \geq 0. \tag{17}$$

The distribution should be biased towards A_k so we assume

$$E\left(R_k - A_k\right) \cdot A_j = 0 \quad \forall j \neq k. \tag{18}$$

Since the A_j are orthogonal, $A_k \cdot A_j = 0 \quad \forall j \neq k$ and an equivalent assumption is

$$ER_k \cdot A_j = 0 \quad \forall j \neq k. \tag{19}$$

In other words, R_k is on average equal to A_k in every direction perpendicular to A_k. The distribution is assumed to be symmetric about the R_k. In particular, the different orthogonal dimensions are independent and have variance

$$E\left(R_k \cdot A_j\right)^2 = \frac{\sigma_k^2}{p-1} \quad \forall j \neq k. \tag{20}$$

A new parameter σ_k^2 was introduced: it is the variance associated with the kth component and spread evenly among the $p-1$ dimensions perpendicular to A_k. This will be of much use later. The variance parallel to A_k has been neglected. We recall R_k is normalized and so

$$\sum_{j=1}^{l} \left(R_k \cdot A_j\right)^2 = 1 \tag{21}$$

which immediately implies

$$E\left(R_k \cdot A_k\right)^2 = 1 - \sigma_k^2. \tag{22}$$

In order to satisfy (17), σ_k must obey

$$0 \leq \sigma_k^2 \leq 1. \tag{23}$$

4.2 Residuals

Consider the data point X_i. It has a predictor point N_i under the static C_k model. If instead the randomized component model based on R_k is employed, the predictor point becomes a random variable Q_i. Here we prove a few results about these predictor points and their residuals.

Theorem 1. *Let X_i be a data point, N_i its predictor in the static model, and Q_i its predictor in the random model. These points have the following properties:*

i. The predictor point N_i of X_i is

$$N_i = \sum_{k=1}^{m} \left(X_i \cdot C_k\right) C_k. \tag{24}$$

ii. The squared magnitude of the ith residual is

$$||X_i - N_i||^2 = X_i \cdot X_i - \sum_{k=1}^{m} (X_i \cdot C_k)^2. \tag{25}$$

iii. In the randomized model, the predictor point Q_i of X_i is

$$Q_i = \sum_{k=1}^{m} (X_i \cdot R_k)\, R_k. \tag{26}$$

iv. The expected squared distance between Q_i and N_i is

$$E\,||Q_i - N_i||^2 = \frac{1}{p-1} \sum_{k=1}^{m} \sigma_k^2 \left(X_i \cdot X_i - 2X_i \cdot N_i + p\,(X_i \cdot C_k)^2 \right) \tag{27}$$

In proving this theorem we will make use of a technical lemma.

Lemma 1. *If $Y, Z \in P$ then*

$$E\,(Y \cdot R_k)\,(Z \cdot R_k) = \frac{\sigma_k^2}{p-1}\,(Y \cdot Z) + \left(1 - \frac{p\sigma_k^2}{p-1}\right)(Y \cdot C_k)\,(Z \cdot C_k) \tag{28}$$

The proof of Lemma 1 is found in the appendix. Now we prove Theorem 1.

Proof of Theorem 1. The point nearest X_i in M is found by projection to be

$$N_i = \sum_{k=1}^{m} (X_i \cdot C_k)\, C_k, \tag{29}$$

establishing *i*. Observing that the residual $X_i - N_i$ is perpendicular to N_i and expanding norms, *ii* is obtained

$$||X_i - N_i||^2 = X_i \cdot X_i - \sum_{k=1}^{m} (X_i \cdot C_k)^2. \tag{30}$$

If instead of C_k the components are R_k then the predictor is

$$Q_i = \sum_{k=1}^{m} (X_i \cdot R_k)\, R_k, \tag{31}$$

yielding *iii*. The expected squared distance between the predictors is

$$E\,||Q_i - N_i||^2 = E\,||Q_i||^2 - 2E\,(Q_i \cdot N_i) + E\,||N_i||^2\,. \tag{32}$$

Since the R_k are orthonormal,

$$E\,||Q_i||^2 = \sum_{k=1}^{m} E\,(X_i \cdot R_k)^2. \tag{33}$$

Lemma 1 applies as $X_i \in P$ and therefore

$$E\left|\left|Q_i\right|\right|^2 = \sum_{k=1}^{m} \frac{\sigma_k^2}{p-1}\left(X_i \cdot X_i\right) + \left(1 - \frac{p\sigma_k^2}{p-1}\right)\left(X_i \cdot C_k\right)^2. \tag{34}$$

Expanding Q_i in the second term of (32) yields

$$E\left(Q_i \cdot N_i\right) = \sum_{k=1}^{m} E\left(X_i \cdot R_k\right)\left(R_k \cdot N_i\right) \tag{35}$$

where $X_i, N_i \in P$ so by Lemma 1,

$$E\left(Q_i \cdot N_i\right) = \sum_{k=1}^{m} \frac{\sigma_k^2}{p-1}\left(X_i \cdot N_i\right) + \left(1 - \frac{p\sigma_k^2}{p-1}\right)\left(X_i \cdot C_k\right)\left(N_i \cdot C_k\right). \tag{36}$$

Equation (29) implies that since $k \leq m$, $N_i \cdot C_k = X_i \cdot C_k$ which allows the simplification

$$E\left(Q_i \cdot N_i\right) = \sum_{k=1}^{m} \frac{\sigma_k^2}{p-1}\left(X_i \cdot N_i\right) + \left(1 - \frac{p\sigma_k^2}{p-1}\right)\left(X_i \cdot C_k\right)^2. \tag{37}$$

Finally, the third term is constant:

$$E\left|\left|N_i\right|\right|^2 = \sum_{k=1}^{m} \left(X_i \cdot C_k\right)^2. \tag{38}$$

Substituting these terms into (32) and simplifying we find

$$E\left|\left|Q_i - N_i\right|\right|^2 = \frac{1}{p-1}\sum_{k=1}^{m} \sigma_k^2\left(X_i \cdot X_i - 2X_i \cdot N_i + p\left(X_i \cdot C_k\right)^2\right), \tag{39}$$

which is statement *iv.* □

It should be noted that p is sometimes unknown but generally very large. Approximating for large p, equation *iv* simplifies as

$$E\left|\left|Q_i - N_i\right|\right|^2 \approx \sum_{k=1}^{m} \left(X_i \cdot C_k\right)^2 \sigma_k^2. \tag{40}$$

This form will be used for the remainder of the paper.

The preceding theorem solved for squared lengths of two values: the residual from a static model and the uncertainty in the predictor from a random model. A predictor is as certain as its ability to fit the data, reflected by the residual. Hence these values should, on average, equal each other. Formally, we have regression equations

$$\left|\left|X_i - N_i\right|\right|^2 = E\left|\left|Q_i - N_i\right|\right|^2. \tag{41}$$

By equation (40),

$$||X_i - N_i||^2 \approx \sum_{k=1}^{m} (X_i \cdot C_k)^2 \, \sigma_k^2. \tag{42}$$

The left hand side is given by Theorem 1. We recall the constraint from equation (23),

$$0 \leq \sigma_k^2 \leq 1. \tag{43}$$

This is an inequality constrained linear regression problem in σ_k^2 with n points and m variables. Several algorithms such as [3] can be applied to this problem.

4.3 Statistical Testing

A natural application is identifying outlier data points. When n is large, both included and new points can be tested. Suppose we are have a data point X_i and want the probability that it was not sampled from the same model. Proceeding similarly to before, we find the predictor

$$N_i = \sum_{k=1}^{m} (X_i \cdot C_k) \, C_k \tag{44}$$

and squared residual length

$$||X_i - N_i||^2 . \tag{45}$$

From (42) the squared residual length is predicted by

$$\sum_{k=1}^{m} (X_i \cdot C_k)^2 \, \sigma_k^2 \tag{46}$$

yielding a test statistic

$$t_i = \frac{||X_i - N_i||^2}{\sum_{k=1}^{m} (X_i \cdot C_k)^2 \, \sigma_k^2}. \tag{47}$$

Values of t_i smaller than 1 simply indicate point i matches the model well. When t_i is much larger than 1, the ith data point deviates from the model and may be of interest. An underlying assumption of the linear regression model (42) is normally distributed error. Therefore a right-tailed χ^2 test is appropriate for testing the t_i to identify interesting points.

Another test compares the principal components produced by two separate samples. Prior to comparing principal components, we must first match them. Ordering by eigenvalue does not suffice because nearby values may interchange between samples. Therefore, we match by distance, noting that components are the same under negation. Suppose two samples have principal components B_k and C_k. Then $(B_j \cdot C_k)^2$ reflects the degree to which they are in the same direction. Overall, we select a permutation ρ of the first m natural numbers to maximize

$$\sum_{k=1}^{m} \left(B_k \cdot C_{\rho(k)}\right)^2. \tag{48}$$

This is a weighted matching problem, solved by the Hungarian algorithm [4]. The last few components may not match well because their corresponding entries are cutoff by selecting the first m components. This can be resolved by identifying a few more than m components. After matching the components, each squared distance

$$\left|\left|B_k - C_{\rho(k)}\right|\right|^2 \tag{49}$$

is compared with the expected sum of distances

$$E\left|\left|R_k - B_k\right|\right|^2 + E\left|\left|S_{\rho(k)} - C_{\rho(k)}\right|\right|^2 \tag{50}$$

where S is the second sample's analogue of random variable R. Expanding one of the terms,

$$E\left|\left|R_k - B_k\right|\right|^2 = E\left(R_k \cdot R_k\right) - 2ER_k \cdot B_k + B_k \cdot B_k \tag{51}$$

which using (20) and normalization simplifies to

$$2 - 2\left(1 - \sigma_k^2\right) = 2\sigma_k^2 \tag{52}$$

and similarly for the second sample. We now have a test statistic. Although confidence intervals and the like require knowing more about the distributions, experimentation will yield a good cutoff score. In this analysis we ignored the fact that the data sets likely span different spaces P. By design, the equations here omit any mention of p, the dimension of P. We may therefore work in the direct sum of these spaces. Finally this test did not compare variance along principal components. Performing an ANOVA test may yield additional insight.

5 Network Packet Data

A computer monitoring a network has access to packet headers and payloads. It can also record timing information and the interface on which the packet arrived. Packet headers can take many different forms depending on the protocols employed. Dealing with multiple protocols simultaneously is left to future work. Here we assume that the protocol and therefore the packet header structure is fixed. Payloads are ignored as they vary in structure and size each packet. Packet headers and additional information have a fixed structure and will be used as inputs to the algorithm.

Parsing the protocol headers and adding additional information yields various fields. The types of fields discussed here apply to any protocol stack. However in the examples, we assume standard TCP/IP over Ethernet. Each field can be taken as a dimension of $E = \mathbb{R}^n$ and each packet as a point in $y_i \in E$. Kernel PCA cannot be applied at this point because it assigns meaning to the ordering of fields such as port numbers and addresses.

5.1 Flags

Protocol headers often have flags to indicate state such as TCP's flags FIN, SYN, RST, etc. These can be treated as separate binary dimensions. The coordinate in the dimension corresponding to the flag is arbitrarily 0 when set and 1 when unset. Let $\beta \in [0, 1]$ be the frequency with which the flag occurs is set in a sample. The centered coordinate is then

$1 - \beta$ if the flag is set and $-\beta_v$ if not. The arbitrary constant 1 difference between these coordinates yields a standard deviation

$$\sqrt{\beta - \beta^2}. \tag{53}$$

Scaling could of course make the standard deviation constant. More on scaling can be found later.

5.2 Port and Address Numbers

Port and address numbers have the property that equality is more significant than distance. For example, finger, HTTP, and SSH traffic are equally likely to be related but use ports 79, 80, and 22, respectively. Similarly, addresses usually represent distinct hosts and traffic patterns. To deal with these fields, we treat each value as its own flag and create a separate binary dimension for each value.

Formally, we define a function $t : E \mapsto \mathbb{R}^b$ where b is the number of possible ports or addresses. For $e \in E$, coordinate e is 1 and all others are 0. We will address centering and scaling issues later. An important property of this function is that

$$t(y_i) \cdot t(y_j) = \begin{cases} 1 & \text{if } y_i \text{ and } y_j \text{ have the same address} \\ 0 & \text{otherwise.} \end{cases} \tag{54}$$

Simply adding a dimension for each port and address ignores potentially useful information. Service ports (those numbered less than 1024) have special meaning in many operating systems. Hosts on a local network likely have different traffic patterns than hosts separated across the Internet. For these reasons, it is useful to add additional dimensions. Port ranges relevant to the operating systems and applications involved can be added as their own dimensions. A binary dimension corresponding to membership in a subnet allows segregation of local networks. An alternative method assigns a binary dimension to each entry in the system's routing tables. In either case, subnets can be thought of as separate fields and translated in the same fashion as addresses.

5.3 Sequence Numbers and Times

Some fields are expected to change while the model is setup to detect significant changes. One approach to these variables, discussed in [1], is a weighted version of PCA. Older values receive lower weights so that packets are compared to those closer in time to it. It is beyond the scope of this paper to implement weights in the kernel PCA algorithm. A second approach includes these variables for purposes of computing the regression model. This allows the algorithm to identify a relationship between time and sequence numbers but makes testing new data points against old model difficult.

Sequence numbers are susceptible to wrapping around. When this is detected, a constant can be added to wrapped values, keeping the packets in order within a batch. However, the monitor is then required to keep a list of connections. Many operating systems provide this functionality.

5.4 Preparation for Kernel PCA

The translations on parsed packets $y_i \in E$ described above are implemented by a function $\Psi : E \mapsto D$. Recall fields such as TCP window size are unchanged by translation while IP addresses all have their own dimension. Since D is very high dimensional, it is intractable to compute the vectors $x_i = \Psi(y_i)$. Instead, the extension of kernel PCA described in Section 3 is employed. This requires a dot product function

$$l\left(y_i, y_j\right) = \Psi\left(y_i\right) \cdot \Psi\left(y_j\right). \tag{55}$$

We now show how to construct l from Ψ. There are only two types of dimensions in D: those unchanged by translation and newly created binary dimensions. Let u_i and v_i denote the projection of x_i onto the dimensions unchanged and created by Ψ, respectively. Without loss of generality,

$$\Psi\left(y_i\right) = \left(u_i, v_i\right) \tag{56}$$

and thus

$$\Psi\left(y_i\right) \cdot \Psi\left(y_j\right) = u_i \cdot u_j + v_i \cdot v_j. \tag{57}$$

One term, $u_i \cdot u_j$, can be computed directly from the inputs. It remains to find $v_i \cdot v_j$. If Ψ employs transformations $\{t_k\}_{k=1}^{\gamma}$ of the form described in the preceding paragraphs then

$$v_i = \left(t_1\left(y_i\right), t_2\left(y_i\right), \ldots, t_\gamma\left(y_i\right)\right) \tag{58}$$

so that

$$v_i \cdot v_j = \sum_{k=1}^{\gamma} t_k\left(y_i\right) \cdot t_k\left(y_i\right). \tag{59}$$

By (54), each term is computed using a simple test for equality. We conclude that $l(y_i, y_j)$ is simply $u_i \cdot u_j$ plus the count of matching flags, port numbers, addresses, ranges, and subnets between packets i and j.

We have neglected the issue of centering. The kernel PCA algorithm assumes centered inputs

$$x_i' = x_i - \frac{1}{n}\sum_{i=1}^{n} x_i, \tag{60}$$

$$X_i' = X_i - \frac{1}{n}\sum_{i=1}^{n} X_i. \tag{61}$$

However these centerings are intractable to compute directly. We observe that kernel PCA depends purely on the matrix K defined in (8). So it suffices to find K', the corresponding matrix for centered values. Appendix A of [2] solves this problem. Let 1_N be the nxn matrix whose entries are all $1/n$. We have

$$K' = K - 1_N K - K 1_N + 1_N K 1_N. \tag{62}$$

This will emulate centered X_i. A similar formula allows new data points to be centered in an old model for purposes of comparison. The same tricks can be applied to center the x_i using the kernel l and matrix

$$L_{ij} = l(y_i, y_j). \tag{63}$$

Scaling of input variables significantly impacts the output of PCA and kernel PCA. Further, the error model uses residuals and implicitly assumes that all dimensions can be treated equally. An entire chapter of [1] is dedicated to the subject of scaling. In its simplest form, each dimension is scaled so that its standard deviation is 1. For dimensions unchanged by translation, this is straight forward. Binary dimensions created during translation can either be left alone or scaled according to the inverse of (53). In the latter case, equal values contribute

$$\frac{1}{\sqrt{\beta - \beta^2}} \tag{64}$$

to $v_i \cdot v_j$ where β is the frequency with which that value occurs. This requires keeping keeping frequency statistics for each value but fortunately nearly all are 0.

6 Batch Processing

The basic operation of the network monitor is to sniff network packets then perform three tasks: test against previous against previous models, generate new models, and test models against each other. Testing individual packets against previous models is relatively quick and should be performed on every packet to identify attacks. Generating new models is done in batches of packets that may be sampled. Iterative learning algorithms that are able to incorporate data as they come in are being actively researched. Batches can theoretically be of different sizes but in practice it is easier to compare data sets of equal size. Models learned from individual batches are tested against each other as described in Section 4.3.

Batches are best compared when they have equal centering and scaling. When different centerings and scalings are applied, the significance of input variables changes. Further, long term statistics are anyway less suspectible to noise. But it is important to allow the system to adapt to slow changes in the network. For these reasons, decaying statistics should be used for centering and scaling. When the most recent data have sufficiently small impact on statistics, models derived from batches captured near each other can be compared. Common centering can be accomplished by using matrices covering multiple batches in (62) then extracting a submatrix. Scaling statistics are based on input data and frequencies which easily extend across batches.

7 Future Work

Packets do not always have the same headers. They may for example arrive on different types of interfaces, be streams or datagrams, or have undefined fields. This problem was ignored by selecting only a fixed set of headers and assuming they are always present. The learning algorithm and error model might be modified to accomodate missing data.

Particularly useful to the application of computer networks is an iterative version of the algorithm. Given an existing model and new data point, an iterative version could incorporate the point into the model. This would alleviate many of the problems of batching and allow faster response to changes in structure of traffic.

Other learning algorithms such as Locally Linear Embedding [5] and Support Vector Machines could also be applied to the problem. Many algorithms have the power to classify and cluster data. Given the obvious (particular protocols) and less obvious relations between network packets, it would be interesting to see these applied.

8 Conclusion

A network packet starts by being sniffed and parsed into a point in some space E. From there it is mapped by a function Ψ into a much higher dimensional space. Then it enters the kernel PCA algorithm which starts by mapping it under Φ into an even higher dimensional vector space F. The span of the mapped data points in F is called P. Principal component analysis selects a subspace M of P. Using the residuals we are able to estimate uncertainties in the principal components that form the basis of P. Finally, we compare these uncertainties with the packet's residual and determine if the packet is suspect. All of the computations conveniently boil down via dot products to tractable comparisons between packets.

The techniques described here are applied to network packets. Network security and fault detection will find this application particularly useful. While motivated by this purpose, the hope is that the techniques are sufficiently general to find application in other areas.

9 Acknowledgments

Thanks to Steven Low for support and mentoring of this research. We gratefully acknowledge the support of Richard and Dena Krown and the Summer Undergraduate Research Fellowship program who partially funded this research. Lachlan Andrew helped with technical discussions and proofreading.

A Proof of Lemma 1

Now we will prove the lemma:

Lemma 1. *If $Y, Z \in P$ then*

$$E\left(Y \cdot R_k\right)\left(Z \cdot R_k\right) = \frac{\sigma_k^2}{p-1}\left(Y \cdot Z\right) + \left(1 - \frac{p\sigma_k^2}{p-1}\right)\left(Y \cdot C_k\right)\left(Z \cdot C_k\right). \tag{65}$$

Proof. Rewrite the left hand side as

$$E\left(Y \cdot \left(Z \cdot R_k\right) R_k\right) \tag{66}$$

and expand dot products in P to obtain

$$E \sum_{i=1}^{p} \left(Y \cdot C_i\right) \sum_{j=1}^{p} \left(Z \cdot C_j\right)\left(R_k \cdot C_j\right)\left(R_k \cdot C_i\right). \tag{67}$$

By linearity this equals

$$\sum_{i=1}^{p} \sum_{j=1}^{p} \left(Y \cdot C_i\right)\left(Z \cdot C_j\right) E\left(R_k \cdot C_j\right)\left(R_k \cdot C_i\right). \tag{68}$$

We now evaluate $E\left(R_k \cdot C_j\right)\left(R_k \cdot C_i\right)$ by cases.
Case 1: $i = k$ and $j \neq k$. We assumed that R_k is distributed symmetrically about C_k. By (21), $(R_k \cdot C_k)$ is invariant under reflection of R_k under C_k. Therefore

$$E\left(R_k \cdot C_j\right)\left(R_k \cdot C_k\right) = 0. \tag{69}$$

Case 2: $i \neq k$ and $j = k$. Similarly to Case 1,

$$E\left(R_k \cdot C_k\right)\left(R_k \cdot C_i\right) = 0. \tag{70}$$

Case 3: i, j, k are distinct. Recall that the principal components are an orthonormal basis and therefore C_i, C_j, C_k are orthogonal. By hypothesis, R_k is distributed symmetrically about C_k. Formally, $R_k \cdot C_i$ and $R_k \cdot C_j$ are uncorrelated. Then, by definition,

$$E\left(R_k \cdot C_k\right)\left(R_k \cdot C_i\right) = 0. \tag{71}$$

Case 4: $i = j \neq k$. Use C_i as an estimator for A_i in (20) to obtain

$$E\left(R_k \cdot C_j\right)^2 = \frac{\sigma_k^2}{p-1}. \tag{72}$$

Case 5: $i = j = k$. Use C_k as an estimator for A_k in (22) to obtain

$$E\left(R_k \cdot C_k\right)^2 = 1 - \sigma_k^2. \tag{73}$$

Only the cases with $i = j$ yield a non-zero value and therefore (68) simplifies as

$$\sum_{i=1}^{p} \left(Y \cdot C_i\right)\left(Z \cdot C_i\right) E\left(R_k \cdot C_i\right)^2. \tag{74}$$

Separating the $i = k$ term yields

$$\sum_{k \neq i=1}^{p} \left(Y \cdot C_i\right)\left(Z \cdot C_i\right) E\left(R_k \cdot C_i\right)^2 + \left(Y \cdot C_k\right)\left(Z \cdot C_k\right) E\left(R_k \cdot C_k\right)^2. \tag{75}$$

By cases 4 and 5 we have

$$\sum_{k\neq i=1}^{p} (Y \cdot C_i)\,(Z \cdot C_i)\,\frac{\sigma_k^2}{p-1} + (Y \cdot C_k)\,(Z \cdot C_k)\,\big(1-\sigma_k^2\big) \tag{76}$$

$$=\sum_{i=1}^{p} (Y \cdot C_i)\,(Z \cdot C_i)\,\frac{\sigma_k^2}{p-1} + (Y \cdot C_k)\,(Z \cdot C_k)\left(1-\sigma_k^2-\frac{\sigma_k^2}{p-1}\right) \tag{77}$$

$$=\sum_{i=1}^{p} (Y \cdot C_i)\,(Z \cdot C_i)\,\frac{\sigma_k^2}{p-1} + (Y \cdot C_k)\,(Z \cdot C_k)\left(1-\sigma_k^2-\frac{\sigma_k^2}{p-1}\right) \tag{78}$$

$$=\frac{\sigma_k^2}{p-1}\,(Y \cdot Z) + (Y \cdot C_k)\,(Z \cdot C_k)\left(1-\sigma_k^2-\frac{\sigma_k^2}{p-1}\right) \tag{79}$$

$$=\frac{\sigma_k^2}{p-1}\,(Y \cdot Z) + \left(1-\frac{p\sigma_k^2}{p-1}\right)(Y \cdot C_k)\,(Z \cdot C_k)\,. \tag{80}$$

□

References

[1] I. T. Jolliffe, *Principal Component Analysis*, Springer, New York, 2002.

[2] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller, *Nonlinear Component Analysis as a Kernel Eigenvalue Problem*, Neural Computation **10** (1998), 1299–1319.

[3] John Geweke, *Bayesian Inference for Linear Models Subject to Linear Inequality Constraints*, Journal of Applied Econometrics **1** (1986), 127–141.

[4] Giorgio Carpaneto and Paolo Toth, *Algorithm 548: Solution to the Assignment Problem*, ACM Transactions on Mathematical Software **6** (1980), no. 1, 104–111.

[5] Lawrence Saul and Sam Roweis, *Think Globally, Fit Locally: Unsupervised Learning of Low Dimensional Manifolds*, Journal of Machine Learning Research **4** (June 2003), 119–155.