Discovering Outlier Filtering Rules from Unlabeled Data

–Combining a Supervised Learner with an Unsupervised Learner–

Kenji Yamanishi
NEC Corporation
4-1-1,Miyazaki,Miyamae,
Kawasaki,Kanagawa 216-8555,JAPAN
k-yamanishi@cw.jp.nec.com

Jun-ichi Takeuchi
NEC Corporation
4-1-1,Miyazaki,Miyamae,
Kawasaki,Kanagawa 216-8555,JAPAN
tak@ap.jp.nec.com

ABSTRACT

This paper is concerned with the problem of detecting outliers from unlabeled data. In prior work we have developed SmartSifter, which is an on-line outlier detection algorithm based on unsupervised learning from data. On the basis of SmartSifter this paper yields a new framework for outlier filtering using both supervised and unsupervised learning techniques iteratively in order to make the detection process more effective and more understandable. The outline of the framework is as follows: In the first round, for an initial dataset, we run SmartSifter to give each data a score, with a high score indicating a high possibility of being an outlier. Next, giving positive labels to a number of higher scored data and negative labels to a number of lower scored data, we create labeled examples. Then we construct an outlier filtering rule by supervised learning from them. Here the rule is generated based on the principle of minimizing extended stochastic complexity. In the second round, for a new dataset, we filter the data using the constructed rule, then among the filtered data, we run SmartSifter again to evaluate the data in order to update the filtering rule. Applying of our framework to the network intrusion detection, we demonstrate that 1) it can significantly improve the accuracy of SmartSifter, and 2) outlier filtering rules can help the user to discover a general pattern of an outlier group.

1. INTRODUCTION

1.1 Contribution of This Paper

This paper is concerned with the outlier/anomaly detection problem. This is closely related to fraud detection, network intrusion detection, etc., since criminal or suspicious activities may often induce outliers. It is also related to unexpected pattern discovery or rare event discovery.

In prior work [21] we have developed SmartSifter, which is an on-line outlier detection algorithm based on unsupervised learning from data. It takes a data sequence as input in an on-line way and gives each datum a score, with a high score indicating a high possibility of being an outlier. SmartSifter uses a Gaussian mixture model as a statistical representation of normal behaviors. The key idea of SmartSifter is to learn the model with on-line discounting learning algorithms and to calculate a score for a datum on the basis of the change of the model before and after learning from it. The novel features of SmartSifter are: a) a score has a clear statistical/information-theoretic meaning, b) it is adaptive to changes of normal behaviors of new patterns, c) it is computationally inexpensive, d) it attains high detection accuracy. In fact, it was demonstrated using KDD Cup 1999 [6] that network intrusion data were detected with high accuracy among the data that SmartSifter gave higher scores. The drawback of SmartSifter is, however, that it cannot explain why the identified outliers are exceptional.

The contribution of this paper is to provide a new framework for *detecting and explaining outliers* by combining SmartSifter with a supervised learning technique. Based on the scores that SmartSifter outputs, we create labeled data by giving positive labels to higher scored data and negative labels to lower scored data. By supervised learning from both the positive and negative data, we build an outlier filtering rule that discriminates positive data from negative data. This filtering rule is used as a preprocessing of SmartSifter for a new dataset. In this framework individual outliers are identified in the unsupervised learning process, while a general pattern of the identified outliers is discovered in the supervised learning process.

The purpose of building this framework is two-folds: 1) To improve the power of SmartSifter by combining supervised learning with an unsupervised one. 2) To clearly explain the meaning of the outliers through the rule acquired in the supervised learning process. This leads to outlier pattern discovery. We require that 1) and 2) be realized at once.

In the supervised learning process, we use a stochastic decision list as a representation of a classification rule and employ a learning algorithm DL-ESC(DL-SC). The key idea of DL-ESC is to select rules on the basis of the principle of minimizing extended stochastic complexity (ESC [20]) (or stochastic complexity (SC [14]), see Section 4). It is theoretically [20] and empirically [11] demonstrated in the scenario of information theory or statistical decision theory that this principle leads to a strategy of learning classification rules. A version of DL-ESC(DL-SC) which can handle discrete attributes only was introduced in the scenario of text classification [11]. This paper introduces a complete form which can handle both continuous and discrete attributes.

Applying our framework to the network intrusion detection problem, we demonstrate that 1) this framework can significantly increase the accuracy of SmartSifter, and 2)

it can help the user to discover a pattern of some specific groups of intrusions.

1.2 Related Work

In most of existing techniques for outlier/fraud detection, one first learns the mechanism of generating data and then evaluates a given datum relative to the learned mechanism. We may classify the approaches of outlier detection into two types: the supervised-learning based approach and the unsupervised-learning based one. The former requires that in the learning process, any example be labeled with regard to whether it is exceptional/fraudulent or not (see, e.g., [2],[4],[5],[10],[17],[12]), while the latter does not. The latter is more important in practice since labeled examples are not necessarily available in real situations.

Unsupervised-learning based methods have been explored on the basis of the theory of statistical hypothesis testing (see e.g., [1]). Most of them are univariate in nature and suffers from computational difficulty in dealing with multi-variate data. Burge and Shawe-Taylor [3] developed an algorithm for on-line unsupervised outlier detection. It utilizes current user profile and user profile history to calculate a score for each datum in an on-line process (see [13] [9] for similar profiling methods), similarly with SmartSifter. It cannot, however, handle categorical variables while SmartSifter can. It was also shown in [18] that for the network intrusion detection problem using the dataset KDD Cup 1999 SmartSifter outperformed Burge and Shawe-Taylors' algorithm both in terms of detection accuracy and computational efficiency.

Knorr and Ng [7] developed novel unsupervised algorithms for finding *distance-based*outliers, which are defined in terms of a geometric distance without any statistical meaning. We consider *statistical outliers*, which are defined in terms of deviation from an underlying statistical model.

It is important to give a clear explanation to the identified outliers, because it can help the user to understand why they are interesting and how valid they are. There are, however, only a few work on identifying and explaining outliers at once. Knorr and Ng [8] developed algorithms for finding intensional knowledge of *distance-based outliers*. This paper provides a different approach from theirs: we explain statistical outliers rather than distance-based ones. Although the previous work [8] can only indicate that each identified outlier is deviated from the pattern that most of the other data have, our approach helps the user to *discover a pattern* that outliers in a specific group may commonly have.

To the best of our knowledge, in the area of outlier/fraud detection, there is no previous work which combines a supervised learning technique with an unsupervised one, as with our framework.

2. MAIN FRAMEWORK

Below we describe the framework for outlier filtering.

We prepare a *filtering rule* $\mathcal{R}$ that determines any datum to be "positive" or "negative" where a positive datum is thought to be an outlier with high probability. The filtering rule is initially set to be a default rule "negative." For a given dataset S, we filter S through the rule to extract the dataset P that are determined to be positive by the rule.

Then we delete such data from S to get a filtered dataset $S - P$. We then run SmartSifter for $S - P$. It calculates a score for each datum in $S - P$ and outputs a dataset Q consisting of data with higher scores. Based on the scores, we give positive labels to the top $\alpha\%$ data of highest scores in $S - P$ and let a set of the positive labeled examples be P'. On the other hand, we make a random sampling of $\beta\%$ data in $S - P$ from the pool of remaining data $S - P - P'$ and give negative labels to them. We let a set of the negative labeled examples be N'. Here α and β are pre-determined positive numbers.

Next we run a supervised learning algorithm DL-ESC/DL-SC, which takes P' and N' and outputs a classifier L that discriminates the positive examples from negative ones. Among local rules that appeared in the acquired classifier L, we select a number of rules that would be useful for outlier filtering. Criteria for selecting rules are here: 1) their accuracy of identifying positive examples is as high as possible, 2) the number of examples covered by the rule is not too large, and 3) the rule itself is intuitively meaningful. We add the selected rules to $\mathcal{R}$ to update the filtering rule. This process is repeated. The flow is described in Figure 1.

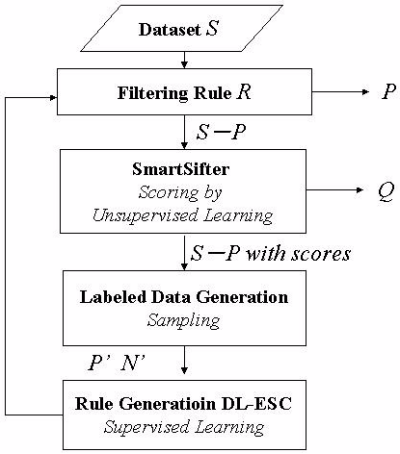

Figure 1: The flow of outlier detection system

The novel features of this framework are as follows:

1) *The filtering rule helps the user to understand outliers.* The filtering rule uses "if–then–else" form to represent a general feature of a specific group of outliers. Hence it is useful for understanding and explaining why the identified outliers are exceptional and how interesting they are. Note that the attributes in data used for filtering are not necessarily the same as those for SmartSifter. Hence the filtering rule is able to characterize outliers in terms of attributes that are not used for SmartSifter.

2) *Combining a filtering rule with SmartSifter improves the power of SmartSifter.* It is expected that using a filtering rule as a preprocessing for SmartSifter greatly improves its efficiency. This is actually demonstrated in the network intrusion detection scenario in Section 5. Further, our framework is expected to be effective in the discovery of a new pattern of outliers. Once the data is filtered by the rule and SmartSifter is applied to the remaining data in the next stage, we are able to discover patterns of outliers that didn't appear in the earlier stages.

3. OUTLIER DETECTOR: SmartSifter

We briefly overview SmartSifter according to [21]. The approach of SmartSifter is given as follows:

I) SmartSifter uses a probabilistic model as a representa-

tion of an underlying mechanism of data-generation. The model takes the following hierarchical structure: Let $(\boldsymbol{x}, \boldsymbol{y})$ denote a datum where $\boldsymbol{x}$ denotes a vector of categorical variables and $\boldsymbol{y}$ denotes a vector of continuous variables. We write the joint distribution of $(\mathbf{x}, \mathbf{y})$ as $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x})p(\mathbf{y}|\mathbf{x})$. We represent $p(\mathbf{x})$ by using a *histogram density* with a finite number of disjoint cells, and for each cell, for all $\mathbf{x}$s that fall into it, we represent $p(\mathbf{y}|\mathbf{x})$ by using a *Gaussian mixture model*. Hence we prepare as many Gaussian mixture models as cells in the histogram density.

II) Every time a datum is input, SmartSifter employs an on-line discounting learning algorithm to update the model. Consider the situation where a sequence of data is given: $(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2) \cdots$ in an on-line process. Given the tth input datum $(\mathbf{x}_t, \mathbf{y}_t)$, identify the cell that $\mathbf{x}_t$ falls into and update the histogram density using the *SDLE* (*Sequentially Discounting Laplace Estimation*) algorithm to obtain $p^{(t)}(\boldsymbol{x})$. Then, for that cell, update the Gaussian mixture model using the *SDEM* (*Sequentially Discounting Expectation and Maximizing*) algorithm to obtain $p^{(t)}(\mathbf{y}|\mathbf{x})$. For other cells, set $p^{(t)}(\mathbf{y}|\mathbf{x}) = p^{(t-1)}(\mathbf{y}|\mathbf{x})$. The most important feature of SDLE and SDEM algorithms is that they gradually *discount* the effect of past examples in the on-line process. This makes the outlier detector *adaptive* to non-stationary sources, of which the mechanism for generating data may change over time. The SDLE algorithm was developed in [21] as an on-line discounting variant of the Laplace law based estimation algorithm, while the SDEM algorithm was developed in [21] as an on-line discounting variant of the incremental EM algorithm (see [15]).

III) SmartSifter gives a score to each datum on the basis of the learned model. Here we calculate the *Hellinger score* defined as follows: Let $p^{(t)}(\mathbf{x}, \mathbf{y})$ be the probability distribution learned after obtaining the tth datum. Then the Hellinger score at the tth datum is given by

$$S_H(\mathbf{x}_t, \mathbf{y}_t) = \sum_{\mathbf{x}} \int \left(\sqrt{p^{(t)}(\mathbf{x}, \mathbf{y})} - \sqrt{p^{(t-1)}(\mathbf{x}, \mathbf{y})} \right)^2 d\mathbf{y}.$$

Intuitively, this score measures how large the distribution $p^{(t)}$ has moved from $p^{(t-1)}$ after learning from $(\boldsymbol{x}_t, \boldsymbol{y}_t)$. Thus a highly scored data indicates a high possibility that the datum is an outlier in the sense that it greatly contributes to changing a statistical model.

The computation time for SmartSifter is $O(d^3 m)$ where d is the data dimension and m is sample size.

For a given dataset, we may run SmartSifter to sequentially score the data then sort them according to their scores. Then the sorted dataset is a final output of SmartSifter.

4. RULE GENERATOR: DL-ESC/DL-SC

As for the rule generation, we use a stochastic decision list as a representation of a classifier and employ the principle of minimizing extended stochastic complexity or stochastic complexity as a criterion for rule selection.

Let Ξ be a multi-dimensional space called a domain and let $\mathcal{M} = \{0, 1\}$ where "1" means a positive datum while "0" means a negative one.

We denote a random variable over Ξ as $\boldsymbol{\xi}$ and that over $\mathcal{M}$ as μ. Each component of $\boldsymbol{\xi}$ is called an attribute. For a given positive integer k, let $\mathcal{T}$ be a set of k-terms on Ξ where a k-term is a conjunction of at most k attribute conditions, e.g., $(\xi_1 \geq 40)$ and $(\xi_3 < 50)$ and $(\xi_5 \geq 60)$ is a 3-term and ξ_j denotes the j-th attribute.

A *stochastic decision list* L is a conditional probability distribution $p(\mu|\boldsymbol{\xi})$ taking a form of $L = (t_1, v_1, p_1), \cdots, (t_s, v_s, p_s)$. Here for any given input $\boldsymbol{\xi}$, L assigns $\mu = v_i$ with probability p_i where the i is the least index that $\boldsymbol{\xi}$ makes t_i true. Hence it has the following meaning:

```
If ξ makes t_1 true, then μ = v_1 with probability p_1
else if ξ makes t_2 true, then μ = v_2 with probability p_2
.....
else μ = v_s with probability p_s.
```

Stochastic decision lists have been developed in a scenario of learning stochastic rules [19] as a probabilistic variant of Rivest's decision lists [16]. We employ an algorithm DL-ESC [11] for learning a stochastic decision list from given positive and negative examples. DL-ESC takes as input a data sequence of pairs of $\boldsymbol{\xi}$ and μ: $(\boldsymbol{\xi}_1, \mu_1) \cdots (\boldsymbol{\xi}_m, \mu_m)$ and outputs a stochastic decision list. The key of DL-ESC is to employ the principle of minimizing *extended stochastic complexity*(ESC) [20] in rule selection. Here the ESC is a kind of extension of Rissanen's stochastic complexity (SC [14]), which is the measure of information quantity included in a data sequence. SC measures the information in terms of the logarithmic loss, equivalently the codelength required for encoding the sequence, while ESC measures it in terms of a general loss function, specifically the 0-1 loss in this case. We may also use an algorithm DL-SC, which is obtained by just replacing ESC with SC in DL-ESC. The details of DL-ESC/DL-SC are given in Appendix.

The computation time for DL-ESC/DL-SC is $O(d^k m)$ where d is the data dimension, k is a positive integer, usually set less than 4, and m is sample size.

5. EXPERIMENTAL RESULTS

5.1 Dataset

We used the dataset KDD Cup 1999 [6] prepared for network intrusion detection. The purpose of our experiment was to detect as many intrusions as possible without making use of the labels concerning intrusions. Although in KDD Cup 1999 the data labels were used in training for supervised intrusion detection, we used them only for the evaluation of algorithms.

Each datum in KDD Cup 1999 is specified by 41 attributes (34 continuous and 7 categorical) and a label describing attack type (22 kinds: normal, back, buffer_overflow, ftp_write, warezmaster, etc.) where all labels except "normal" indicate an attack. In the original attributes, there are many attributes that are obtained by combining several attributes. We deleted such attributes from the original 41 attributes to get 13 attributes: `duration, src_byte, dst_byte, srv_error, wrong frag, numofurgent, numoffail, service, protocol, land root login, guest, flag`. We used the 13 attributes for DL-ESC. Further we selected four attributes: `service, duration, src_bytes, dst_bytes` for the use of SmartSifter. This is because these four were thought of as the most basic attributes. We were interested in seeing how well SmartSifter worked using these attributes only. We were also interested in generating rules using more attributes than SmartSifter. Only 'service' is categorical. Since the continuous attribute values were concentrated around 0, we transformed each value into a value far from 0, by $y = \log(x + 0.1)$

where the base of logarithm is e.

The original dataset contains 4,898,431 data, including 3,925,651 attacks (80.1%). This rate of attacks is too large for statistical outlier detection. Therefore, we removed the data whose attribute *logged in* is negative. The resultant dataset, which we named SF, consists of 976,157 data, including 3,377 attacks (0.35%). Attacks that successfully *logged_in* are called *intrusions*.

We further produced from SF five datasets S0,S1,S2,S3,S4 by random sampling, where each of them consists of about 10% of SF. Table 1 shows the data size, the number of intrusions and the details of intrusions for each dataset.

5.2 Illustration by an Example

Let us illustrate how our framework works. First S1 was fed to SmartSifter where the first 8,000 data in S1 were not scored but used only for training because the model would not be well-trained in the early stages. After processing all of the data in S1, we sorted them according to their scores that SmartSifter gave.

We create labeled examples by giving positive labels to the top 1% data of highest scores and negative labels to the 3% data in S1 randomly sampled from the remaining data. (Note that "positive" does not necessarily mean "intrusion," but highly scored data.) Then we input them to DL-ESC, which output the following decision list:

```
If duration ≥ 0.74 & protocol =tcp
  then positive with prob.0.84 (539/639)
elsif src_byte < 10.91 & root login =user
  then positive with prob.0.93 (2572/2774)
else negative with prob.  0.97(151/155)
```

Here (539/639) in the first rule means that the number of the data satisfying the condition "duration $\geq$ 0.742 & protocol =tcp" is 639 while 539 of them were positive. We ignored the first rule because its accuracy (539/639) is not high enough. We picked up only the last rule in the list and use it as a filtering rule:

```
If neither ''duration ≥ 0.742 & protocol =tcp''
nor ''src_byte < 10.907 & root login =user '' then positive.
```

Next we input a new dataset S2 into the rule, and let P be the set of data determined to be positive by the rule. The number of data included in P was 311 and 199 of them were intrusions of type `back` (63.98% accuracy, 63.17% coverage). No other type of intrusions was included. (*Note*: This information about intrusions were not used for learning but only for the evaluation of algorithms.) This implies that the filtering rule successfully generalized a pattern of `back` and led to discovery of a specific group of outliers, which turned out to be a group of `back.`

For the filtered dataset S2−P, we run SmartSifter again. Then we observed that 25 intrusions were included in the top 689(=1000−311) dataset Q of highest scores. Hence we could detect 224(=199+25) intrusions in the 1000 data included in S2. On the other hand, when S2 was fed to SmartSifter only, 110 intrusions were included in the top 1000 data of highest scores. This implies that the coverage of intrusion detection in 1000 data was significantly increased from 34.92% (=110/315) to 71.11%(=224/315) by filtering with the rule plus SmartSifter.

Similarly, we applied DL-ESC again to S2–P to get a rule:

```
If duration ≥ 1.131 & protocol=tcp then positive.
```

A new dataset S3 was filtered by the preceding rule, then the filtered dataset was again filtered by the above rule. We observed that 35 intrusions were included in the extracted positive dataset and 70% of them were of type `warezclient.` This led to discovery of a specific group of outliers, which includes a group of `warezclient.`

5.3 Evaluation

We investigated how well our framework performs in comparison with SmartSifter. Throughout this section, we denote our framework as R&S (=Rule and SmartSifter) and SmartSifter as SS. For example, for the case where S0 was used as a training set to construct a filtering rule, each of S1, S2, S3, and S4 was used for test, i.e., it was first filtered by the rule and SmartSifter was applied to the filtered dataset.

Table 2 shows the coverages achieved by SS and R&S. Here the coverage of SS is the ratio of the number of intrusions in the top n data (n =1,000, 2,000, 3,000) of highest scores produced by SS, to the total number of intrusions in the dataset. The coverage of R&S is the ratio of the number of intrusions in the output of R&S, to that. The output here is the set which consists of P (flitered data by rules) and Q (highest scored data by SS) in Figure 1. For example, consider the case in which S1 was input as test data and the number of outputs was 1,000 (see the column whose first line is S0 in Table 2). Let the number of intrusions included in the positive dataset extracted by the filtering rule be N_1. Let the number of intrusions included in the top $1,000-|\mathrm{P}|$ data of highest scores in the filtered dataset S1−P be N_2. Then the number of intrusions detected by R&S in the 1,000 of S1 is calculated as $N_1 + N_2$. This calculation was done for the cases in which the number of outputs were 1,000, 2,000, and 3,000. Each value in the column whose first line is S0 is calculated as the average of coverages over all test datasets: S1,S2,...,S4. The second column shows the coverage of SS averaged over all training sets: S0,..,S4, while the third column shows those of R&S.

We see from Table 2 that when the size of evaluated data is less than 2000 data (=2%), R&S could detect significantly more intrusions than SS. Remarkably, R&S attained 68.5% coverage in 1000 data on average while SS attained 43.1% for the same sample size.

Figure 2 shows how the averaged coverage grows as the evaluated sample size increases. The horizontal axis shows the *sample rate*–the ratio of the size of evaluated sample to the total data size, while the vertical axis shows the coverage. We observe that the coverage of R&S grows more rapidly than that of SS for 0-1% sample rates. When the sample rate is larger than 3%, they are not different so much. We are able to say that the outlier filtering contributes to improving the accuracy of SS for low sample rates.

Figure 3 shows the graph of the coverages of R&S and SS for small sample rates. Here the coverages of R&S are plotted for all of different training sets. The coverages were calculated only for positive datasets extracted by the rule. For example, R&S(S0) indicates the coverage of R&S when S0 was used as a training dataset to construct a filtering rule. All of the coverages of R&S are concentrated around 64%–70%, which is 20%-55% higher than those of SS.

	total data size	# of intrusions	ratio of intrusions(%)	details		
				# back	# warezclient	# others
S0	97560	337	0.35	202	120	15
S1	97216	346	0.36	241	93	12
S2	98013	315	0.32	206	92	17
S3	97890	335	0.34	216	100	19
S4	98189	342	0.35	219	106	17

Table 1: Data Sets

evaluated sample size	SS (%) (average)	R&S (%) (average)	training dataset				
			S0(%)	S1(%)	S2(%)	S3(%)	S4(%)
in 1000 (1%)	43.1	68.5	70.8	69.1	64.3	69.4	68.9
in 2000 (2%)	70.5	71.8	73.1	73.0	69.5	71.6	71.6
in 3000 (3%)	75.2	74.3	75.9	74.6	71.6	74.7	74.5

Table 2: Coverage Comparison: SS vs R&S

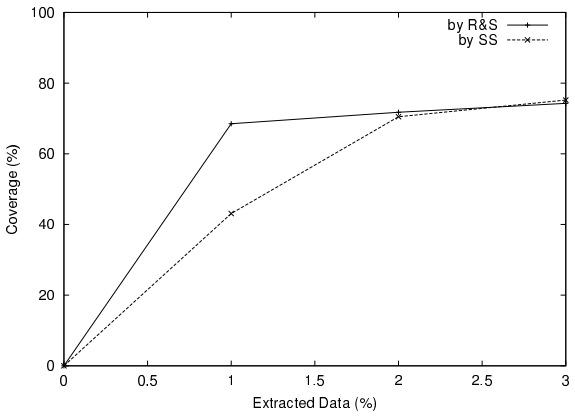

Figure 2: SS vs R&S, Averaged Coverage

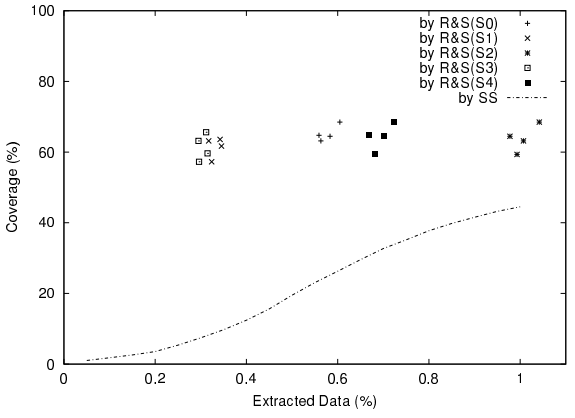

Figure 3: SS vs R&S, Different Training Sets

6. CONCLUDING REMARKS

We have proposed a new framework for outlier detection by combining an unsupervised outlier detector: SmartSifter with a supervised learning algorithm: DL-ESC. An outlier filtering rule is constructed with supervised learning by generating labeled data on the basis of the scores that SmartSifter calculates for unlabeled data. By using the rule as a preprocessing of SmartSifter, we are able to significantly improve the power of SmartSifter. Furthermore it helps the user to discover a general pattern that a specific group of outlier may have. We demonstrated through the network intrusion detection that our framework was effective both in identification and explanation of outliers.

7. REFERENCES

[1] V. Barnett and T. Lewis, *Outliers in Statistical Data*, John Wiley & Sons, 1994.

[2] F. Bonchi, F. Giannotti, G. Mainetto, and D. Pedeschi, A classification-based methodology for planning audit strategies in fraud detection, in *Proc. of KDD-99*, pp:175–184, 1999.

[3] P. Burge and J. Shawe-Taylor, Detecting cellular fraud using adaptive prototypes, in *Proc. of AI Approaches to Fraud Detection and Risk Management*, pp:9–13, 1997.

[4] T. Fawcett and F. Provost, Adaptive fraud detection, *Data Mining and Knowledge Discovery*, vol.1, Kluwer Academic Publishers, Boston CA, pp:291-316 (1997).

[5] http://www.hnc.com

[6] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[7] E. M. Knorr and R. T. Ng, Algorithms for mining distance-based outliers in large datasets, in *Proc. of the 24th VLDB Conference*,pp:392–403, 1998.

[8] E. M. Knorr and R. T. Ng, Finding intensional knowledge of distance-based outliers, in *Proc. of the 25th VLDB Conference*, pp:211–222, 1999.

[9] T. Lane and C.E. Brodley, Temporal sequence learning and data reduction for anomaly detection, *ACM Trans. on Information and System Security*, 2,pp:295–331 (1999).

[10] W. Lee, S. J. Stolfo, and K. W. Mok, Mining audit data to build intrusion detection models, in *Proc. of KDD-98*, 1998.

[11] H. Li and K. Yamanishi, Text classification using ESC-based stochastic decision lists, in *Proc. of CIKM'99*, pp:122–130 (1999).

[12] Y. Moreau and J. Vandewalle, Detection of mobile phone fraud using supervised neural networks: a first prototype, Available via: ftp://ftp.esat.kuleuven.ac.jp/pub/SISTA/moreau/reports/icann97_TR97-44.ps.

[13] U. Murad and G. Pinkas, Unsupervised profiling for

identifying superimposed fraud, in *Proc. of PKDD'99*, pp:251-261 (1999).
[14] J. Rissanen, Fisher information and stochastic complexity, *IEEE Trans. Inf. Theory*, IT-42, 1, pp. 40–47 (1996).
[15] R. M. Neal and G. E. Hinton, A view of the EM algorithm that justifies incremental, sparse, and other variants, ftp://ftp.cs.toronto.edu/pub/radford/www/publications.html 1993.
[16] R.L. Rivest, Learning decision lists, *Machine Learning*, 2, pp:229–246, (1987).
[17] S. Rosset, U. Murad, E. Neumann, Y. Idan, and G. Pinkas, Discovery of fraud rules for telecommunications-challenges and solutions, in *Proc. of KDD-99*, pp:409–413, 1999.
[18] J.Takeuchi and K.Yamanishi, Empirical evaluation of an outlier detection engine SmartSifter, in *Proc. of Symposium on Information and Its Applications* (in Japanese), 2000.
[19] K.Yamanishi, A learning criterion for stochastic rules, *Machine Learning*, Vol.9,pp:165-203 (1992).
[20] K. Yamanishi, A decision-theoretic extension of stochastic complexity and its application to learning, *IEEE Trans. on Inf. Theory*, IT-44, pp.1424–1439 (1998).
[21] K. Yamanishi, J.Takeuchi, G.Williams, and P.Milne, On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms, in *Proc. of KDD2000*, ACM Press, pp:250–254, (2000).

APPENDIX: Learning Stochastic Decision Lists

Below we give a brief sketch of DL-SC(DL-ESC)–the algorithm for learning stochastic decision lists. A data sequence is denoted as $D^m = (\boldsymbol{\xi}_1, \mu_1) \cdots (\boldsymbol{\xi}_m, \mu_m)$ where $\boldsymbol{\xi}_t$ is an input, μ_i is a corresponding output, and m is sample size.

First we give a strategy for discretizing a continuous variable. Here the discretizing means determining a threshold τ for each variable ξ_i then change ξ_i into a binary variable by setting $\xi_i \geq \tau$ and $\xi_i < \tau$. Here τ is chosen so that for D^m, $I(\tau|D^m)$ is minimized with respect to τ:

$$I(\tau|D^m) = I(D^m_+) + I(D^m_-),$$

where D^m_+ is a subsequence of D^m such that $\xi_i \geq \tau$ while D^m_- is a subsequence of D^m such that $\xi_i < \tau$. Here $I(D^m)$ is *stochastic complexity* [14] of D^m calculated as follows:

$$I(D^m) = mH\left(\frac{m_1}{m}\right) + \frac{1}{2}\log\frac{m\pi}{2}, \qquad (1)$$

where the base of the logarithm is 2, m_1 is the number of examples in D^m such that $\mu = 1$, and $H(z) = -z\log z - (1-z)\log(1-z)$.

For a set of attributes: $V = \{\xi_1, \cdots, \xi_n\}$, for a given input data sequence D^m, for each ξ_i, we choose $\tau_i = \arg\min_\tau I(\tau|D^m)$ as above, and change ξ_i into $\tilde{\xi}_i$ s.t. $\tilde{\xi}_i = 1$ if $(\xi_i \geq \tau_i)$ $\tilde{\xi}_i = 0$ if $(\xi_i < \tau_i)$. $\tilde{V} = Dis(V, D^m) = \{\tilde{\xi}_1, \cdots, \tilde{\xi}_n\}$. Here a discrete variable remains the same.

The algorithm DL-SC for learning stochastic decision lists consists of two processes: Growing and pruning. In the growing process a rule minimizing the splitting complexity is sequentially added to the list. Below we show how to calculate the splitting complexity. For $t \in \mathcal{T}$, for a given data sequence D^m, we let D^m_t be a subsequence of D^m such that $\boldsymbol{\xi}$ makes t true, and let $D^m_{\neg t}$ be a subsequence of D^m such that $\boldsymbol{\xi}$ makes t false. We define the *splitting complexity* of t w.r.t. D^m by

$$I(t|D^m) = I(D^m_t) + I(D^m_{\neg t}) \qquad (2)$$

where $I(D^m)$ follows (1). In the process of growing, we choose t so that $I(t|D^m)$ is minimized w.r.t. t.

Given:
A set of attribute variables $V = \{\xi_1, \cdots, \xi_n\}$,
a data sequence D^m,
a positive integer k

Initialization:
$\mathcal{D} := \mathcal{D}^m$,
$\tilde{V} := Dis(V, \mathcal{D})$,
$T :=$ a set of conjunctions of at most k attributes in $\tilde{V}$
$L := \emptyset$

Growing
do while $\mathcal{D} \neq \emptyset$:
For each $t \in T$, calculate the information gain $\Delta I(t|\mathcal{D})$.
$t^* := \arg\min I(t|\mathcal{D})$
if $I(\mathcal{D}) - I(t^*|\mathcal{D}) > 0$
then
$\mathcal{D}^* :=$ a subset of $\mathcal{D}$ that makes t^* true
$v^* :=$ the labels that most frequently appeared in $\mathcal{D}^*$
$p^* := \frac{|\mathcal{D}^*_+|+1/2}{|\mathcal{D}^*|+1}$ ($|\mathcal{D}^*_+|$ is the number of examples s.t. $\mu = v^*$ in $\mathcal{D}^*$)
Add (t^*, v^*, p^*) to L
$\mathcal{D} := \mathcal{D} - \mathcal{D}^*$
$\tilde{V} := Dis(V, \mathcal{D})$
$T := T - \{t^*\}$
else
Go out of the loop
Add the default rule $(true, v^*, p^*)$ to L

Pruning
do while $L :\neq$ only a default rule
Prune the bottom rule other than the default rule from L to obtain L'
if $\mathcal{I}(D^m : L') \geq \mathcal{I}(D^m : L)$
then Go out of the loop
else $L := L'$
Output L

Figure 4: DL-SC(DL-ESC)

We may also calculate $I(D^m)$ by the following formula:

$$I(D^m) = \min\{m_1, m - m_1\} + \lambda\sqrt{m\log m}, \qquad (3)$$

where m_1 is the number of examples in D^m such that $\mu = 1$, and λ is a positive real number. The quantity (3) is called the *extended stochastic complexity* (*ESC*) [20] for the 0-1 loss.

Once we get a decision list, we prune a rule from the bottom in order, so that the total complexity is minimized. Below we show how to calculate the total complexity. For a decision list L, for a conjunction t appearing in the bottom rule in L, we let $\mathcal{D}_t$ be a dataset such that $\boldsymbol{\xi}$ makes t true. We calculate the *total complexity* of D^m w.r.t. L by

$$\sum_t I(\mathcal{D}_t) + \lambda' \sum_t \ell(t), \qquad (4)$$

where the sum is taken over all ts appearing in L, and λ' is a positive real number, which may be different from λ in (3). Here $\ell(t) = \log|T|$ for the total number $|T|$ of conjunctions with at most k attributes over $\tilde{V}$. A list L for which (4) is minimized is finally output.