Concept Drift と Scale-free の性質を持つデータセットに対する検知の自動化手法

愛甲健二[†1]

概要：マルウェアやスパムメッセージに代表されるセキュリティシーンのデータセットは，時間軸によってその性質が常に変化し続けるため，恒久的に対策を実施することが難しい．本稿では Concept Drift の性質を持つデータセットにおいて，その Scale-free 性に着目し，データ分布の変化に自動的に追随しながら検知を行う自動化手法を提案する．

キーワード：Automation, Concept-Drift, Scale-free, Malware

Automation: Detection method for data sets with Concept Drift and Scale-free properties

KENJI AIKO[†1]

Abstract: Data sets of security scenes typified by malware and spam messages are constantly changing their nature according to time axis, so it is difficult to implement countermeasures permanently. In this paper, we focus on the Scale-free property of a data set with the property of Concept Drift and propose a method to detect while following automatically the change of data distribution.

Keywords: Automation, Concept-Drift, Scale-free, Malware

1. はじめに

現時点において恒久的にマルウェアを検知できる絶対的なアルゴリズムは存在しない．マルウェアは時代によって流行やその性質が変わっていくため，検知アルゴリズムもまた改善し続ける必要がある．変化し続けるデータセット，いわゆる Concept-Drift の性質を持つデータセットに対して，検知アルゴリズムはその変化に対して常に追随しなければならない．本稿では，セキュリティシーンにおいてよく観測される「変化するデータセット」に対して，その検知アルゴリズムを自動化する手法を提案する．

Concept-Drift は様々な分野で観測される[1]．例えば，天気予報における天気，気温，湿度といったデータは季節や時期によって変化する Concept-Drift の性質を持つデータセットである．またショッピングにおける各商品の売上は顧客の趣味趣向の変化に影響するため Concept-Drift のデータセットといえる．どちらも Concept-Drift の例ではあるが，天気は季節という「周期性」があり，その「変化」そのものを予測可能な点が異なる．半年後に売れる商品を予測することは困難だが，半年後の気温を予測することはさほど難しくない．一般に人間が関与しているものには周期性を発見することが難しく，自然現象には周期性を確認できる場合が多い．

セキュリティシーンにおいては，マルウェア検知，スパムメッセージ対策，またオンラインゲームにおける不正ユーザー検知も変化するデータセットを対象とする．これらはいずれも対策を行うとそれを迂回する方法が考えられ，さらにその対策をすればまた迂回されるといういわゆる「いたちごっこ」の性質が内在しており，これが Concept-Drift の原因になっている．また対策と迂回は人の手によって行われており，そこから周期性を見つけ出すことは困難である．このようなセキュリティシーンにおけるデータセットは少なくない．

またセキュリティに関するデータセットには Concept-Drift 以外にもうひとつ特徴がある．それはデータ分布の偏りである．例えば，ある期間において収集されたマルウェアをクラスタリングすると一部のクラスタに大量のマルウェアが属することになり，一方でほとんどマルウェアが属していないとても小さいクラスタもまた大量に生まれるという，いわゆる Scale-free 性[2]が確認できる．これはマルウェアの類似性に言及した既存研究においても確かな事実である[3].

本稿では，この「データ分布の偏り」という特徴を利用して「変化するデータセット」の検知を自動化する手法を提案する．

2. 前提

まずは議論を明確に進めるために，いくつかの記号と条

†1 LINE 株式会社
LINE Corporation

件を定義したい.

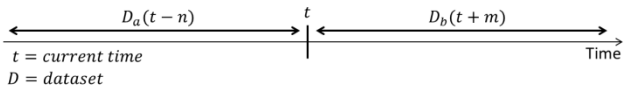

図 1　時間とデータセットの定義

図 1 において，tは時間，$D_x(t)$は時間tにおけるデータセット，$D_a(t-n)$は過去のデータセット，$D_b(t+m)$は未来のデータセットとする．D_a，D_bは良性と悪性を持ち，ラベル付けはされていない．また Concept-Drift の性質を持つ．さらにD_aは十分に多くのユーザーから「通報」されたデータの集合とし，Scale-free の性質も持つ.

D_aとD_bの具体的な例を示す．例えばマルウェア検知においてはユーザーからのレピュテーション情報がD_aとなり，スキャンすべき全ファイルがD_bとなる．よって，過去のレピュテーション情報$D_a(t-n)$を用いてスキャン対象$D_b(t+m)$に含まれるマルウェアを検知することが目的となる．同様にスパムメール対策においてはユーザーにより報告されたスパムメールがD_aとなり，送受信される全てのメールがD_bとなる.

以上より，$D_a(t-n)$を用いて，自動的に$D_b(t+m)$を識別する手法を提案する.

2.1 Scale-free

データセット$D_a(t)$の分布について触れる．多くのユーザーは悪性を悪性として正確に通報しようとするが，すべてのユーザーがミスなく悪性を判断できるわけではない．よって多くの良性が$D_a(t)$に含まれてしまう．つまり$D_a(t)$は，ただユーザーが悪性だと判断しただけの”信頼できない”データセットである．このようなデータセットを便宜上 untrusted data-set と呼ぶ.

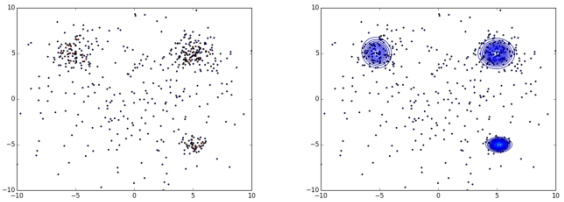

図 2　$D_a(t)$の分布例

$D_a(t-n)$を用いて自動的に$D_b(t+m)$を識別するためには，第一に$D_a(t)$のような untrusted data-set を入力として，以下の 2 つを出力するアルゴリズムA_aが必要となる.

1. $D_a(t)$を正確にラベル付けする
2. $D_a(t)$の悪性に共通する特徴を算出する

データセット$D_a(t)$は経験的に Scale-free に従う．また$D_a(t)$は任意のnヶ所の中心点O_nに密集する悪性と，全体に平均的に散らばる良性で構成される．よって各データに対して，O_nからの距離を求めることで悪性と良性の判断ができる．つまりA_aは，図 2 において左図にある二次元空間にマップされた任意の数の点が与えられたとき，右図にあるnヶ所の円の中心点O_nを算出するアルゴリズムと言い換えることができる.

アルゴリズムA_aは対象とするデータセットによって異なるが、図 2 を例としよう．二次元空間にマップされた任意の数の点が与えられたとき，すべての点同士の距離を計算すれば密集したデータの中心点を探し出せる．特別な工夫をせず，愚直に計算しても計算量は$O(n^2)$以下となる．計算量を減らしたい場合は適当なサイズをサンプリングしても良い．ある分布から十分な量サンプリングされたデータの分布は，サンプリング数が十分であれば似た分布となる．そのため計算可能な量をサンプリングしてもほぼ同様の結果が得られる.

1. 計算可能な量をサンプリングする（必要ならば）
2. 各データ間の距離を計算する
3. 密集した箇所の数をカウントする
4. 散らばっているデータを削除する
5. 混合ガウス分布をフィッティングする

上記アルゴリズムで密集点を探し，悪性としてラベル付けするデータを青い円で図示したものが図 2 の右図である．このデータセットサンプルと画像一式，ソースコードは GitHub に公開している[4].

ちなみに，特にこのアルゴリズムでなければならないという決まりはない．このアルゴリズムは手法のひとつであり，nヶ所の円の中心点O_nを現実的な時間で算出できれば何でも良い.

2.2 Concept-Drift

任意の時間nからmまでの区間のデータセット$D_x(n)$, $D_x(n+1), D_x(n+2), \dots, D_x(m)$の集合を$[D_x[n,m]]$とする．$[D_x[n,m]]$はnからmへ時間が進むごとに変化する．nからm$-1$の範囲にある任意の値 a をとる．このとき$D_x(a)$と$D_x(a+1)$が十分に類似することを前提とする.

表 1　時間軸と各クラスタに所属するデータ数の関係

	クラスタ A	クラスタ B	クラスタ C	…
n	50	30	10	…
n + 1	10	60	5	…
n + 2	0	10	80	…
…	…	…	…	…
m	0	0	10	…

Scale-free と Concept-Drift，そして$D_x(a)$と$D_x(a+1)$が十

分に類似することを確認するためには時間軸とクラスタを要素とした表を作成すれば良い（表 1）．縦軸を時間，横軸を各クラスタ，そして各ブロックに入る値がその期間の各クラスタに属するデータ数とする．

クラスタを固定して時間軸を縦に進めていくとそのクラスタに属するデータ数が増減していく．これが特定クラスタにおける Concept Drift である．逆に時間を固定して全クラスタを横に観察していくとその期間のデータの分布，つまり偏り，Scale-free を確認できる．

実際のマルウェアのデータセットを用いて時間軸と各クラスタに所属するデータ数の関係をグラフ化した例を図 3 に示す．横軸が時間，縦軸が各クラスタに所属するデータ数である．

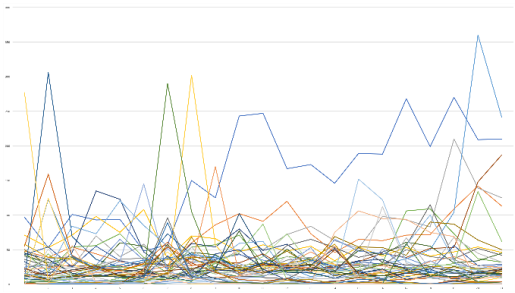

図 3　時系列データにおけるクラスタの変化

特定のマルウェアが時間軸の変化によって増減することが分かる．Concept-Drift が発生する理由は単純だろう．攻撃者が悪性 A を作り，ユーザーから通報され，悪性 A がブロックされる．それを認識した攻撃者が新たに悪性 B を作る．これが繰り返されることでデータセットが変化していく．いわゆる「いたちごっこ」である．

さて，アルゴリズムA_aにより$D_a(t)$のラベル付けと悪性に共通する特徴の算出を行った上で，そのデータを用いて$D_b(t+1)$から類似のデータを検知するアルゴリズムをA_bとする．A_bは一般的な教師あり学習のアルゴリズムでも良いし，単純な類似度を算出するものでも良い．ただし，False Positive は 0.01%以下を目標とする．

2.3　False Positive

誤検知率の許容範囲は環境によって異なるが，セキュリティシーンにおける False Positive は 0.1%以上を許容できない場合が多い．一般的なパターン認識とは異なり True Positive を犠牲にして False Positive を一定ライン以下に落とすことが求められる．

データセットが Scale-free であり密集しているデータを悪性，散らばっているデータを良性と仮定する．この散らばっている良性というのは密集している悪性の場所にも少なからず散らばっている．よって密集しているデータの中心点O_nを得ても，その近くにわずかな良性が存在してしまう．もしその中心点の密集度が少なくそれほど検知率に寄与しない場合はその中心点は使用しない，という選択ができる．少しの True Positive を犠牲に False Positive を落とす．では密集度が高い中心点に対して近くに良性が見つかった場合はどうするか．その良性とその周辺をホワイトリストとする．

セキュリティシーンでは，対象が未来のデータセットとなるため False Positive を 0 にすることは非常に難しい．よって True Positive を犠牲にしても False Positive を 0 に近づけやすいこと，そして False Positive が発生したときに対処しやすいことを優先したアルゴリズムを選ぶ．

3.　提案手法

FFRI Dataset 2018 には 2017 年に報告されたマルウェア 29 万件と正常系 21 万件の表層解析ログが収められている[9]．D'_aを FFRI Dataset 2018 内にあるマルウェア 29 万件のデータセット，tを 2017 年の月，D_Nを正常系 21 万件のデータセットとする．そして 1 月のマルウェアデータセット$D'_a(1)$を用いて 2 月のマルウェアデータセット$D'_a(2)$を検知する手法を考える．

FFRI Dataset 2018 には「ssdeep による Fuzzy Hash の結果」が含まれている．以後この Fuzzy Hash 値を単に Hash 値と記述する．$D'_a(1)$に含まれる 28514 検体を Hash 値でソートすると 3072:ZIDNcIFN3tw4...となる検体が 263+1 見つかる．そして$D'_a(2)$の中でこれと同一の Hash 値となる検体は 62 存在する．ちなみに$D'_a(3)$では 68 検体，$D'_a(4)$は 62 検体，$D'_a(5)$は 58 検体，$D'_a(6)$で 17 検体，そして$D'_a(7)$，6 ヶ月後でやっと 7 検体まで落ちる．つまり 1 月の時点でこの Hash 値となる検体をマルウェアと判断しておけば，2 月以降においてその検体を止めることができる．

さて，$D'_a(1)$にはすでにマルウェアというラベルがついているが，本来は untrusted data-set であるため$D'_a(1)$にランダムな良性を追加したデータセットを改めて$D_a(1)$としよう．ここで$D'_a(1)$と$D_a(1)$の違いを考える．

$D'_a(1)$をソートするともっとも多い Hash 値となるのは 3072:ZIDNcIFN3tw4...で 263 検体，続いて 172 検体，3 番目に 95 検体であるが，ここにランダムな良性を加えたとして，この上位 3 つの順位は変わるだろうか．答えは変わらない，”全体に平均的に散らばった”良性は Fuzzy Hash におけるソートの上位に入ることはない．つまり Scale-free の上位数パーセントは極めて高い確率でマルウェアと判断でき，またそれを学習しておけば直近の未来においても有効に働くのである．

3.1　Suffix Tree

データセットが量的データならばN次元空間において点が密集しているnヶ所とその中心点O_nを算出するアルゴリズムを考えれば良い．しかし FFRI Dataset 2018 のような質的データの場合は，”可能な限り多くのデータに類似する特徴”を見つける必要がある．FFRI Dataset 2018 におけるアル

ゴリズムA_a，A_bについて考える．

$D'_a(1)$を Hash 値でソートすると，同一ではないがかなり似ている Hash 値が散見される．表 2 においては 5 位以降の Hash 値は最後の 1 文字を除いて同一である．Hash 値は異なるが，おそらくこれらは同じマルウェアもしくは亜種だと推察されるため同じものと判断したい．

表 2　Fuzzy Hash 値によるソートの結果

順位	数	Fuzzy Hash 値
1	263	3072:ZIDNcIFN3tw4QfwmAOMe6UJbVM/vkA9OQzY6e
2	172	1536:ahUDofByDJWbMGcEFLPEPKOJUsy1+VMA:aIof
3	95	48:aF2k+IYYTAxB5EC7BWnVxAiidlxav2trgtQvh2l
4	89	96:8ufPG/bXeyV0UYDv92+uA8JvU3j4JOYtAUnIgQa
5	25	6144:jhBoLWUUsUYdMq2FUzG14mxbn/ZNLo77g...9
6	25	6144:jhBoLWUUsUYdMq2FUzG14mxbn/ZNLo77g...l
7	24	6144:jhBoLWUUsUYdMq2FUzG14mxbn/ZNLo77g...n
8	22	6144:jhBoLWUUsUYdMq2FUzG14mxbn/ZNLo77g...Q
…	…	…

表 3　Suffix Tree の例

ABC	+BCD	+BCB	+ABD
_: 1 A: 1 B: 1 C: 1 _: 1 **B: 1** **C: 1** _: 1 C: 1 _: 1	_: 2 A: 1 B: 1 C: 1 _: 1 **B: 2** **C: 2** _: 1 D: 1 _: 1 C: 2 _: 1 D: 1 _: 1 D: 1 _: 1	_: 3 A: 1 B: 1 C: 1 _: 1 **B: 3** _: 1 **C: 3** _: 1 B: 1 _: 1 D: 1 _: 1 C: 3 _: 1 B: 1 _: 1 D: 1 _: 1 D: 1 _: 1	_: 4 A: 2 B: 2 C: 1 _: 1 D: 1 _: 1 **B: 4** _: 1 **C: 3** _: 1 B: 1 _: 1 D: 1 _: 1 D: 1 _: 1 C: 3 _: 1 B: 1 _: 1 D: 1 _: 1 D: 2 _: 2

任意のサイズの 1 次元配列をn個含む集合を$S_1(n)$とする．$S_1(n)$から出現頻度の高い共通部分を探索するアルゴリズムをA_aとする．1 次元配列の代表的なものは文字列である．n個の文字列から共通部分文字列を探す問題は Longest Common Substring problem（LCS）である．

FFRI Dataset 2018 において，すべての Hash 値から出現頻度の高い共通部分文字列を探し出せれば，それが$D_a(t)$に含まれるマルウェアの特徴である．LCS は Suffix Tree を用いれば線形時間で解けることが知られている[5].

“ABC”が与えられたとき，終端文字を追加した”ABC¥0”を入力データとして，”ABC¥0”，”BC¥0”，”C¥0”，”¥0”を Suffix Tree に入れる．すると Suffix Tree は表 3 の左端 ABC の列になる．この Tree にさらに”BCD”を追加すると+BCD の列になり，さらに”BCB”，”ABD”を追加すると+BCB, +ABD の列となる．そしてもっとも頻出した 2 文字は，Tree のトップから大きい数値をたどることで得られる．この例においては BC である．この共通部分文字列 BC を未来のデータセットに対して利用する．

Suffix Tree を利用したアルゴリズムをA_a，A_bとし，FFRI Dataset 2018 の$D_a(1)$に対して使用する．$D_a(1)$は，$D'_a(1)$に同じ数の良性をランダムに追加したデータセットとする．$D_a(1)$にアルゴリズムA_aを適用したことにより得られた共通部分文字列（size=30）を表 4 に示す．

表 4　$D_a(1)$の共通部分文字列（size=30）

順位	数	Fuzzy Hash 値
1	1051	6144:jhBoLWUUsUYdMq2FUzG14mxbn
2	849	6144:MJueTkwOwoWOQ3dwaWB28edeP
3	646	384:1YsQzCkQ2gtWAv3815td3ReNI5
4	347	9HFJ9rJxRX1uVVjoaWSoynxdO1FVBa
…	…	…

$D_a(1)$の中で 5 個以上のデータに含まれていた共通部分文字列を抽出し，$D'_a(2)$のデータセットに適用する．$D'_a(2)$のデータセットの中で，抽出した共通部分文字列が含まれていれば悪性，含まれていなければ良性とする．結果，$D'_a(2)$にある 19682 検体のうち 8130，約 41.3%が悪性と判断され，21 万件の正常系D_Nのうち 5072 件，約 2.4%を誤検知した．ここから誤検知した共通部分文字列を削除すると 19682 検体のうち 8091，約 41.1%を悪性と判断し，誤検知は 0 件となる．

3.2 Time Series Analysis

同様のことを$D_a(n)$と$D'_a(n+1)$に対して行った結果を表 5 に示す．

表 5　$D_a(n)$を学習し，$D'_a(n+1)$を検知した結果

n	$n+1$	検知数	FP	検知数(FP0)	総数
1	2	8130 (41.3%)	5072	8091 (41.1%)	19682
2	3	7610 (32.1%)	4607	7540 (31.8%)	23681
3	4	9400 (39.4%)	5009	9351 (39.2%)	23828
4	5	11136 (43.3%)	4597	10996 (42.7%)	25692

5	6	8749 (37.4%)	5061	8639 (36.9%)	23390
6	7	10202 (43.5%)	4413	10092 (43.0%)	23432
7	8	9181 (39.1%)	4243	8913 (38.0%)	23431
8	9	7683 (34.8%)	4910	7646 (34.7%)	22029
9	10	10193 (43.9%)	4660	10181 (43.8%)	23201
10	11	10715 (47.1%)	4409	10563 (46.5%)	22704
11	12	10016 (32.9%)	4944	9976 (32.7%)	30416

検知率はおおよそ 30%から 45%くらいとなる．続いて$m=6$とし，$[D_a[n,m-1]]$を用いて$D'_a(m)$を検知したものを表 6 に示す.

表 6　$[D_a[n,m-1]]$を学習し，$D'_a(m)$を検知した結果

$n,m-1$	m	検知数	FP	検知数(FP0)
5	6	8749 (37.4%)	5061	8639 (36.9%)
4,5	6	9685 (41.4%)	7219	9466 (40.4%)
3,4,5	6	10039 (42.9%)	8811	9778 (41.8%)
2,3,4,5	6	10285 (43.9%)	10268	10006 (42.7%)
1,2,3,4,5	6	10474 (44.7%)	12135	10228 (43.7%)

直近の 1 ヶ月だけではなく，2 ヶ月，3 ヶ月と増やすことで検知率が改善されていく．ただし，検知に対する寄与率は徐々に低下していく．最後に 1 月から 11 月のデータセットを用いて 12 月を検知した結果を表 7 に示す.

表 7　$[D_a[1,11]]$を学習し，$D'_a(12)$を検知した結果

$n,m-1$	m	検知数	FP	検知数(FP0)
11	12	10016 (32.9%)	4944	9976 (32.7%)
1,2,…,11	12	12826 (42.1%)	15897	12561 (41.2%)

1 月から 11 月までのデータセットを利用すると，11 月のみの場合と比較して 10%近い検知率の上昇が見込める．また False Positive も注目すべき点である．$[D_a[1,11]]$を用いると False Positive が 15897 まで増加した．この値は良性 21 万件の約 7.5%である．そして True Positive を約 1%減らすだけで False Positive を 0 にできる.

ちなみに時系列データに基づくマルウェアの変化は 2014 年にも研究している[6].

3.3 LCS

Suffix Tree は共通部分文字列を抽出できる．しかし連続してない共通文字を探索したい場合は使用できない．共通部分列を求めるのは Longest Common Subsequence problem（LCS）問題となる．どちらも LCS と略されるが，これらは異なる.

2 つの文字列が与えられたとき，それらの共通部分列を求めたい．”abcd”と”aced”が与えられた場合に，どちらの文字列も a, c, d という順番で出現するため” acd”を得たい．順序が同じであれば，文字と文字の間に他の文字が入ったり，または連続していても構わない．Longest Common Substring よりもこちらの方が汎用性は高いが，計算量もまた大きい．n個のデータに対して計算量$O(n^2)$が許容できるならばこちらを用いても良い.

また計算量を無視するなら，2 つのデータに共通する特徴Cを得るアルゴリズムさえ存在すればアルゴリズムA_a，A_bを作れる．n個のデータに対して，愚直に各 2 データのCを取得していくと計算量$O(n^2/2)$でCの集合が得られる．その集合に含まれるもっとも多いCが，もっとも多くのデータに共通する特徴である.

3.4 n次元配列

これまで「ssdeep による Fuzzy Hash の結果」を用いたマルウェアの検知を例に記述してきた．しかし，データセットが前提を満たし，かつ，1 次元配列のデータセットであればこの手法はどのようなものでも適用できる．例えば，FFRI Dataset 2018 には含まれていないが，マルウェアファイルそのものに対して Suffix Tree を用いて，マルウェアファイルの集合から頻出する共通部分データ列を取り出すことも可能である．または，API コールのログやファイルアクセスといった実行結果から得られたデータセットに対してもそれらを 1 次元配列に変換すれば同様に適用できる．ただしアルゴリズムA_a，A_bの計算量は考慮しない.

では 2 次元配列，3 次元配列といったデータセットの場合はどうだろうか．つまり，これまで 1 次元配列をn個含む集合$S_1(n)$を対象としてきたが，任意のサイズの 2 次元配列をn個含む集合を$S_2(n)$とする．この$S_2(n)$から共通する特徴を求める問題である．2 次元配列の代表的なものは画像であるが，画像に対しても共通する部分を見つけられれば 1 次元配列と同様に識別できることは自明である．2 つの画像の共通部分を得るアルゴリズムの研究は SIFT，SURF，A-KAZE など様々なものがあるが，本稿の趣旨から逸脱するため説明はしない[7]．2 つの画像の共通部分が得られるならば，愚直に$D_a(t)$の各 2 データに共通する特徴Cを求めればよい.

3 次元以上の配列，つまりm次元配列$S_m(n)$においてはどうだろうか．こちらも基本的な理論は同じだと考えているが，実用的なデータで試したことはないため，これについては記述しない.

また当たり前の話だが，$S_m(n)$においても特徴を維持したまま$S_1(n)$に変換できるならば，本稿の手法で解決できる．つまり画像を Fuzzy Hash により文字列に変換するなら$S_1(n)$として扱える．しかし，例えばアダルト画像をその「アダルト」の要素を保持したまま Hash 値に変換することは極めて難しいだろう.

3.5 まとめ

$D'_a(12)$からランダムに選んだ 1000 検体とD_Nからランダムに選んだ 10000 検体を合わせたものを$D_b(12)$とする．そして，$[D_a[1,11]]$を用いて$D_b(12)$を検知した結果，$D_b(12)$に含まれる 1000 検体のマルウェアのうち 428 を検知した．検知率は 42.8%，False Positive は 0．この結果から，静的解析

による Fuzzy Hash のみを用いた場合，検知率は約 40%前後となった．

4. 考察

マルウェアのデータセットには Scale-free と Concept-Drift の性質が内在している[6]. そしてこの性質を利用すると，自動的に変化に追随する検知アルゴリズムを作ることができる．

本章では提案手法を使い実際に運用するにあたって注意すべき問題点，改善点を上げ，またそれらを含めた考察を行う．

4.1 汚染攻撃

データセットを自動で学習，検知するシステムに対する攻撃について考察する．D_aは十分に多くのユーザーから通報されたデータの集合であるため，攻撃者が多くのユーザーアカウントを手にできれば良性を大量に通報できる．結果，大量に通報された良性は Scale-free の上位に入り，共通する特徴として抽出され，D_bの識別時に大量に誤検知される．これは密集したデータを悪性，散らばっているデータを良性と仮定しているために発生する問題である．対策としては，True Positive の急激な増加を監視することでアラートを出したり，他の悪性クラスタとの差異を調べることで意図して大量に通報された良性を排除できる．ただし，それらもアルゴリズムが知られてしまえば迂回されるため，本質的な対策はない．

1. 大量のユーザーアカウントを入手できる
2. 学習，検知のアルゴリズムを推測する

攻撃者が上記の条件をクリアすることで汚染攻撃が可能となる．つまり汚染攻撃に対しては自動化を実現できておらず，人による監視が必要となる．

4.2 アルゴリズム

本稿でもいくつかのアルゴリズムを用いたが，目的を満たせるならアルゴリズムは何でも良い．また実際に運用する場合は計算量を考慮しなければならないため，あらゆる環境において正しいアルゴリズムというものは存在しない．ただ、セキュリティシーンにおいては False Positive が致命的な問題となるため単純な検知性能だけではなく，現実的な時間で処理が終わるアルゴリズムA_a，A_b，False Positive が見つかった場合に容易に対処できること，True Positive を犠牲にして False Positive を下げられること，といった点を重要視すべきである．一般的なパターン認識とは異なる性質が多くあるため，高度なアルゴリズムをそのまま適用するのは難しいかもしれない．

4.3 Concept-Drift

$D_a(t)$と$D_a(t+1)$との差，つまり Concept-Drift によって発生する変化についても触れよう．$D_a(t)$にはいくつかの悪性クラスタが存在し，良性は全体に散らばっている．この場合，2 パターンの Concept-Drift が考えられる．

1. クラスタに含まれるデータが変化した
2. 全体に散らばっているデータが変化した

仮に$D_a(t)$に 2 つの悪性クラスタ X，Y が存在したとする．さて，時間tで X をブロックすると，当たり前だが$D_a(t+1)$には X クラスタと判断されるデータが減る可能性が高い．つまり，悪性への「対策」を行うことでデータセットの変化，Concept-Drift が発生する．これがパターン 1.である．対して，もし時間tで X をブロックしなかった場合，悪性クラスタ X，Y は$D_a(t+1)$においてもその数が変化しない可能性が高い．しかし，全体に散らばっているデータ，すなわちクラスタが形成されるほど成長していない悪性や間違って通報された良性が時間軸によって変わる可能性は十分にある．これが 2.である．パターン 1.は対策によって引き起こる変化であり，パターン 2.は自然現象的な変化であるともいえるかもしれない．

またパターン 1.は True Positive に影響を与え，パターン 2.は False Positive に影響を与えることが多い．攻撃者が「X がブロックされた」という情報を得て，X ではなく新たな悪性クラスタ Z を作り始めるとブロックされる X の代わりにブロックされない Z が増加し，結果 True Positive が減る．逆に特別なことをしていないにもかかわらず検知率が増え始めたら，それは False Positive が増えている可能性が高い．その場合はパターン 2.の Concept-Drift が起こったと考えられる．

いずれのパターンにおいても再学習が必要となる．分布の変化は更新のタイミングを決める上で極めて重要な要素であるため，その変化を知るアルゴリズムも考えておく必要がある．

また本稿では過去のデータセットを用いた未来のデータセットの検知を扱ったが，未来で引き起こされる変化のタイミングをあらかじめ予測し，対処する研究もある[8]. False Positive の問題があるため，現実的な環境で実用化するのは難しいと思うが，このような研究もあることを付け加えておく．

4.4 Scale-free

最後に Concept-Drift によって変化する Scale-free について触れる．

FFRI Dataset 2018 には良性ファイル 21 万件の解析データがある．そのデータセットD_Nを分析すると傾きの低い Scale-free になる．つまり，仮に世の中にマルウェアが存在しなくても，ソフトウェアは何かしらの Scale-free 性を持っていると考えられるが，そもそも Scale-free はセキュリティに限った性質ではないため，当たり前といえば当たり前である．

悪性が存在しない世の中で，初めて悪性 X を作り出す攻撃者が生まれたとする．彼はそれを世の中に広め，結果的に悪性 X は多くのユーザーに通報される．すると図 4 の左上のグラフのように Scale-free を超えてより極端な分布となる．すぐに X が対策され徐々に減っていくが，今度は悪性 Y が出現する．そしてその Y も対策される．こういった「いたちごっこ」が続くと，グラフは徐々に図 4 右上，そしてさらに図 4 左下のように変化していく．左上，右上，左下，を同じグラフにしたのが右下である．いたちごっこが続くことで 1 位とそれ以外の差が縮まっていく．

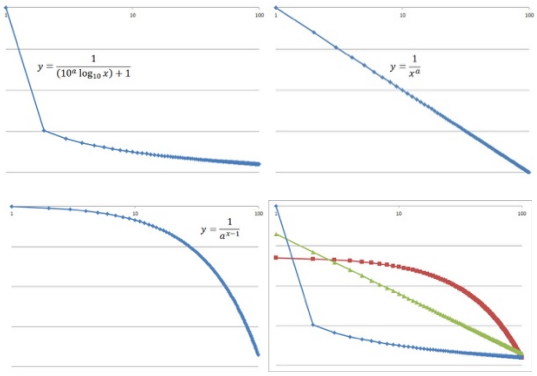

図 4　Scale-free の変化（両対数グラフ）

もし新しいプラットフォームのデータセットを得られた場合は，そのデータセットがどのような分布になっているのかを確認することで，どの程度効果的な対策が施されているかを知ることができるかもしれない．

5. まとめ

本稿では，時系列のデータセットが Scale-free と Concept-Drift を併せ持つと仮定し，データセットの分布を利用して変化するデータセットに対応し，検知率を一定に保つ自動化手法を提案した．またその手法に至る過程を一般化し，特定のデータセットに依らないものとした．提案手法ではマルウェアのデータセットを用いた例を示しているが，マルウェア以外においても同様に成り立つことを確認済みである．またこの手法に対する攻撃の可能性，アルゴリズム，更新タイミングについて考察した．

本稿で利用したソースコードは全て GitHub にて公開している[4].

6. 参考文献

[1] Game, J.; Zliobaite, I.; Bifet, A.; Pechenizkiy, M.; and Bouchachia, A. 2014. A Survey on Concept Drift Adaptation. ACM Computing Surveys (CSUR) 46(4):44.

[2] Barabási, Albert-László and . Albert, Emergence of scaling in random networks, Science, 1999

[3] M. Iwamura, M. Itoh, and Y. Muraoka, Automatic Malware Classification System Based on Similarity of Machine Code Instructions, CSS, 2010

[4] https://github.com/kenjiaiko/secml

[5] P. Weiner, 1973, Linear pattern matching algorithm, 14th Annual IEEE Symposium on Switching and Automata Theory. pp. 1-11

[6] K. Aiko, T. Matsuki，時系列データに基づくマルウェア検知アルゴリズムの評価, CSS, 2014

[7] David G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, 2004

[8] A. Kumagai, T. Iwata, Learning Latest Classifiers without Additional Labeled Data, IJCAI-17

[9] 高田雄太，他: マルウェア対策のための研究用データセット ～ MWS 2018 Datasets ～，情報処理学会, Vol.2018-CSEC-82, No.38, 2018 年 7 月