Computer Security Symposium 2017
23 - 25 October 2017

# 機械学習による直感バイナリ分類・可視化（目grep）の実装

大坪 雄平[1,3]　三村 守[2,3]　榊　剛史[4]　後藤 厚宏[3]

**概要**：近年，サイバーセキュリティ人材，特に「高度人材」の育成は重要な課題の 1 つとなっている．しかしながら，高度人材の知識・技能のマニュアル化が困難であり，その能力の伝承に課題がある．そこで本論文では，高度人材の能力のうち特にバイナリ解析分野の「目 grep」に焦点を当て，この能力のツール化を目的とする．これにより，高度人材と同様の分析が可能な人材の増加や，高度な分析の自動化が期待できる．本ツールは，機械学習を使用しており，適切なデータセットを準備することで，高度人材と同様の能力を獲得する．さらに，学習済モデルの入出力に対応することで，データセットそのものを共有せずに能力の共有が可能となる．

**キーワード**：静的解析，機械学習

# Intuitive Binary Classification and Visualization with Machine Learning

Yuhei Otsubo[1,3]　Mamoru Mimura[2,3]　Takeshi Sakaki[4]　Atsuhiro Goto[3]

***Abstract:*** In the area of cyber security, to develop human resources especially with “high-level experts” is one of the urgent issues. However, such advanced knowledge and skill in cyber security depend on their experience and intuition, that makes skill transfer even more difficult. In order to successful handling of this situation, in this paper, we focus on binary analysis in static analysis and describe how to implement cyber security expertise into our tool. This tool will contribute to foster analysts who are able to doing static analysis equivalent of “high-level experts” do and we can also expect automation of high level static analysis. The concept employs machine learning algorithm and by preparing appropriate research dataset for it, analysts can perform “high-level experts”. Moreover, sharing trained models is possible to realize “sharing of expertise”.

***Keywords:*** Static Analysis, Machine Learning

## 1. はじめに

近年，サイバー空間における脅威が深刻化している．2016 年に警察がサイバーインテリジェンス脅威ネットワークを通じて把握した標的型メール攻撃は 4,046 件で，3 年連続で増加しており，サイバー空間における探索行為についても増加傾向にある [1]．一方，経済産業省によると，情報セキュリティ人材は，2016 年の時点で約 13 万人が不足しており，2020 年には，その不足が約 19 万人に拡大するとされている [2]．このような情勢の中，政府のサイバーセキュリティ戦略本部は，2017 年 4 月に「サイバーセキュリティ人材育成プログラム」を策定するなど [3]，サイバーセキュリティ人材の育成は重要な課題となっている．特に，「高度人材」の育成は，高度化・巧妙化するサイバー攻撃に対応するためにも重要である．

上記プログラムによると，高度人材の育成は，学校や企業などの組織が用意した人材育成のカリキュラムだけでは不十分であり，むしろ，突出した能力を持ちうる人材を発掘することができる “場” づくりの推進が重要になるとさ

[1] 警察庁
National Police Agency
[2] 防衛大学校
National Deffense Academy
[3] 情報セキュリティ大学院大学
Institute of information security
[4] 東京大学
University of Tokyo

れている．既存のカリキュラムによる人材育成が困難である理由の一つとして，高度人材の知識・技能がマニュアル化できないということが考えられる．高度人材の知識・技能は経験に基づく膨大な知識から導き出されており，知識・技能の前提条件の知識すべてを文書化するリソースを割くことはできない．

そこで，本研究の目的は，マニュアル化することが難しい高度人材の知識・技能をツール化することとする．知識・技能をツール化することで，高度人材と同様な分析を行える人材の増加や，高度な分析の自動化が期待できる．さらに，高度人材はツール化された能力とは別の能力の獲得にリソースを割くことができるため，高度人材の更なる高度化も期待できる．高度人材の知識・技能は様々なものが考えられるが，本研究では，高度人材の知識・技能のうち特にバイナリエディタを使ったバイナリ解析分野の「目 grep」に焦点を当ててツール化を行う．

この論文の成果を以下に述べる．本論文では，NN（ニューラルネットワーク）や SVM（Support Vector Machine）を用い，256 Byte の生データを入力データとし，クラス分類が可能か 2 つの実験を行なった．1 つ目の実験では，人が容易に区別可能と思われるエンコード文字列を機械学習により区別可能か実験し，99.79%の正解率で区別できることを示した．2 つ目の実験では，入力されたプログラムコードが対象とする CPU アーキテクチャを 4 種類の中から選択するようなクラス分類では，99.63%という正解率を示すことを明らかにした．この分類器を活用し，ファイル全体分の分類結果を俯瞰できるように可視化をするツールを作成し，正常な文書ファイルとプログラムコードを教師データにすることで文書型のマルウェアに埋め込まれたシェルコードの大まかな位置が推定できることを示した．

本論文の構成を以下に述べる．第 2 章では，本論文における目 grep を定義する．第 3 章では，本論文の提案手法について述べる．第 4 章では，提案手法の分類器の性能を評価するため，2 つの分類問題について実験を行う．第 5 章では提案手法の分類器を使い解析対象ファイル全体を目 grep した結果を可視化等するツールの機能について述べる．第 6 章では，作成したツールを使用し，目 grep の可視化を試みる．第 7 章では，提案手法の効果等について考察し，最後にまとめる．

## 2. 目 grep とは

本論文における目 grep とは，あるパターンに一致するものを特別なツールに頼らずに人の目で探索することを指す．目 grep は，様々な場面で利用され，マルウェア解析でも利用される．マルウェアを解析する手法は，実際にプログラム等を動作させてその挙動を観察する動的解析と，プログラム等を動作させることなくプログラムコード等を読み解くことによりその挙動を推測する静的解析の大きく

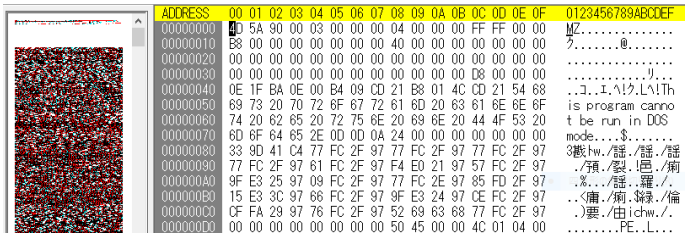



図 1　バイナリエディタの表示の例
**Fig. 1**　Sample of a Hex-Editor.

表 1　Bitmap View における各オフセット値と色の対応
**Table 1**　Correspondence between each offset value and color in Bitmap View

| Value | Color | comment |
|---|---|---|
| 0x00 | 白色 | null |
| 0x01-0x1F | 水色 | 制御文字 |
| 0x20-0x7F | 赤色 | ASCII 文字 |
| 0x80-0xFF | 黒色 | |

2 種類に分けられる．静的解析の中でも，ファイル名，拡張子，プロパティ，ハッシュ値など，解析対象から比較的容易に得られる情報を元に行う解析を表層解析と言う．目 grep は表層解析に該当し，重点的に解析を行う部分の絞り込みなどに活用されている．目 grep は，特別なツールを必要としないため，誰にでも目 grep そのものは可能である．しかしながら，それにより目的を達成できるか否かや解析速度は実施者の解析経験・実力に大きく依存する．

バイナリ解析における目 grep では，**図 1** のようなバイナリエディタを使用する．図の中央部分は，入力したファイルを 1 Byte ごとに 16 進数表記したものが表示される．この機能は，Hex View とも呼ばれ，どのバイナリエディタにも実装されている基本的な機能である．今回の場合，1 行に 16(0x10) Byte 表示されている．図の右側は，中央部のデータに対応するテキストを表示したものである．バイナリエディタによっては複数の文字コード表記に対応しており，今回の場合，シフト JIS で表示されている．図の左側は，1 Byte を 1 つの点で可視化したもので，Bitmap View とも言われる．今回の場合，配色は**表 1** に示すとおりで，1 行あたり，128(0x80) Byte の情報を表示することができる．バイナリエディタの中には，Bitmap View の代わりに，Hex View で表示している部分の逆アセンブル結果を表示する機能を搭載するものもある．今では高解像度が当たり前になってきたが，画面の解像度が XGA（1,024 × 768）が一般的であったとき，1 画面で表示できる情報量は，Hex View で 256(0x100) Byte，Bitmap View で 80 KB 程度であった．

**図 2** は，プログラムコードが埋め込まれた悪性 doc ファイルの Bitmap View である．目 grep の経験を積むと，このような Bitmap View を見て，ファイルの種類を推定したり，埋め込まれているプログラムコードのおおまかな位

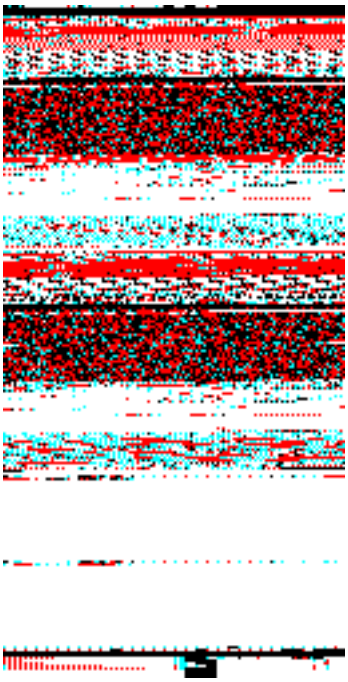


図 2　doc 文書ファイルの Bitmap View
**Fig. 2**　Bitmap View of a doc file.

置を推定したりすることができるようになる．

つまり，目 grep は，与えられた入力データを元に適切なクラスに分類するという点で，機械学習におけるクラス分類問題と言い換えることができる．分類器の精度や対応する分類問題の種類の豊富さが高度人材と同等の能力を持つか否かの 1 つの指標となる．

## 3. 提案手法

本論文では，教師あり学習モデルの SVM または NN を用いる．そのため，ラベル付きの教師データが必要となる．ただし，マルウェアは教師データとして用いず，インターネット上から誰でも容易に入手できるデータを教師データとして用いる．

提案手法は，ファイルの断片の生データの各オフセット値を 0 から 1 の値を取るように 255 で割って正則化したものを入力データとし，その種類を推定する．入力データサイズは，暫定的に 256 Byte とした．これは，バイナリエディタで一画面上に表示される情報量と概ね一致しており，我々のこれまでの解析経験から，場合によっては推定が可能と考えている大きさである．入力データサイズが小さくなればなるほど推定の難易度は高くなる．しかしながら，入力データサイズを大きくし過ぎると，シェルコードのような小さなプログラムの場合，入力データに占めるプログラムの割合が減少し，別のクラスと判定される可能性がある．最適な入力データサイズは，解析の目的により変動すると思われるが，本論文では 256 Byte 固定としている．以下，本論文で用いる SVM および NN について述べる．

### 3.1 SVM

SVM は，1963 年に発表された教師あり学習モデルである [4]．入力層と出力のみからなる単純パーセプトロンと呼ばれるニューロンモデルとして最も簡単なモデルがあり，これを改良したものが SVM である．SVM は，最適化するべきパラメータが少なく（カーネル関数として RBF(ラジアル基底関数）を選択した場合は 2 つ），そのパラメータの算出も容易という利点がある．しかしながら，学習データが増加すると，計算量が膨大になるという欠点がある．

本論文では，学習データからランダムに 1,000 個のサンプルを抽出し，このサンプルに対し，2 種類のカーネル関数（RBF，線形），9 種類の $C$ 値（0.0001，0.001，0.01，0.1，1，10，100，1,000，10,000）および 9 種類の $\gamma$ 値（0.0001，0.001，0.01，0.1，1，10，100，1,000，10,000）でグリッドサーチすることで，最適なパラメータの近似解を求めている．

### 3.2 NN

本論文では，NN のうち最も基本的かつ最もよく使われているといわれる順伝搬型（ニューラル）ネットワーク（feedforward neural network）を用いる．この手法は、多層パーセプトロン（multi-layer perceptron）とも呼ばれ，層状に並べたユニットが隣接層間でのみ結合した構造を持ち，情報が入力側から出力側に一方向のみ伝播する NN である [5]．

本論文で使用するネットワークの概要を**図 3** に示す．これは，3 層の順伝搬型ネットワークで，入力層のユニット数は 256，中間層のユニット数は 400，出力層のユニット数は分類するクラス数により変動し n となっている．各層は全結合となっており，中間層の活性化関数に ReLU(Rectified Linear Unit)，出力層の活性化関数に Softmax 関数を適用し，損失を交差エントロピーで求める．これにより学習を繰り返すことで，出力層の各ユニットは，クラス分類の推定結果を確率で出力するようになる．また，計算量を削減させるためミニバッチ学習法を取り入れている．バッチサイズは，100 としているが，ネットワークの学習プロセスをより安定化させ高速化するため，2015 年に Ioffe らが提案した BN（Batch Nomalization）[6] を取り入れている．なお，最適化アルゴリズムは後に述べる予備実験で検証する．

#### 3.2.1 Dropout の導入について

Dropout[7] は，NN のニューロンをランダムで取り除いて学習していくことで，過学習の抑制が期待できる手法であるとされている．画像認識のように，概念を学習させるような場合，入力データの一部を使用しない場合でも，推定に影響は少ない可能性がある．しかしながら，例えばプログラムコードのように 1 Byte でも欠けていると成立しないようなものを学習させようとした場合，画像認識のように Dropout が有効に動作するかは明らかでない．仮に有

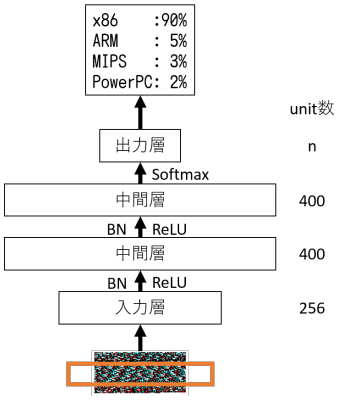



図 3　ネットワーク概要
**Fig. 3**　Overview of the network of the tool.

表 2　エンコードされた文字列の例
**Table 2**　Example of encoded strings

| 16 進数文字列の例 | Base64 文字列の例（抜粋） |
|---|---|
| 1d66792587b71b075ed5372d78c | P0A5xvy8njgyb6bV3jm1XU87 |

効だとしても，最適なドロップ率は，画像認識の場合と比較して低い値となることが予想され，Dropout を導入するメリットが小さくなる．また，Ioffe らによると，BN を導入することにより，Dropout を導入する必要性が下がるとされている．そこで，本論文では Dropout は導入しないこととした．

## 4. 分類器の性能評価

高度人材の能力をツール化するためには，複数の分類問題について，高い精度で分類できることが望まれる．ここでは，簡単な 2 つの分類問題を例に前に述べた提案手法の正解率を求めた．

### 4.1 エンコードの分類

この実験では，16 進数文字列と Base64 文字列の分類を行う．16 進数文字列は，**表 2** 左側に示すとおり，1 Byte のデータごとに対応する 2 桁の 16 進数で表記する文字列である．Base64 文字列は，表 2 右側に示すとおり，データを 64 種類の印字可能な英数字のみを用いて表記する文字列である．

データセットの概要を**表 3** に示す．データセットの偏りを防ぐため，インターネット上から収集した圧縮されたファイルを元にエンコードをしたものを 256 Byte ごとに分割して作成した．圧縮されたデータは，0 から 255 までの値が均等にランダムに現れるデータとなっている．したがって，単純なパターンマッチングでは分類することが難しいデータである．これを元に，変換された 16 進数文字列や Base64 文字列も同様に、マジックナンバーを見つけるような単純なパターンマッチングとは異なる特徴を学習しなければ分類することができないことが期待される．

表 3　データセット A
**Table 3**　dataset A

| Type | Num of sample |
|---|---|
| 16 進数文字列 | 29,420 |
| Base64 | 19,608 |

表 4　実験結果
**Table 4**　Results of the examination

| | Optimizers | accuracy | variance |
|---|---|---|---|
| SVM(RBF) | $C = 100, \gamma = 0.1$ | 0.9979 | 7.49E-7 |
| NN | SGD | 0.9933 | 2.73E-5 |
| | Momentum SGD | 0.9910 | 4.65E-5 |
| | AdaGrad | 0.9901 | 4.55E-5 |
| | RMSprop | 0.9938 | 1.93E-5 |
| | AdaDelta | 0.9933 | 2.39E-5 |
| | Adam | 0.9902 | 2.94E-5 |

このデータセットを使用し，最適化アルゴリズムを変えながら，10 分割交差検証により正解率とその分散を求めた．NN の学習回数は 20 回であり，実験に使用した最適化アルゴリズムは以下に示す 6 種類である．

- SGD(Stochastic Gradient Descent)：初期に提唱された最も基本的な手法
- Momentum SGD：SGD に慣性項 (Momentum) を付与した手法
- AdaGrad[8]：2011 年に提唱された手法で，学習係数を自動で調整する手法
- RMSprop[9]：2012 年に提唱された手法で，AdaGrad を改良した手法
- AdaDelta[10]：2012 年に提唱された手法で，AdaGrad や RMSProp を改良した手法
- Adam（Adaptive moment estimation）[11]：2015 年に提唱された手法で，AdaGrad，RMSprop，AdaDelta を改良した手法

実験の結果を**表 4** に示す．最も正解率の高い手法は SVM で，99.79%となっていた．また，正解率の分散についても SVM が最も低い値となっていた．

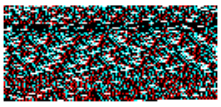
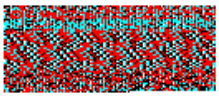
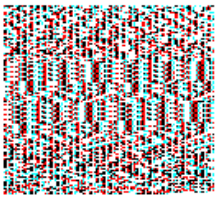
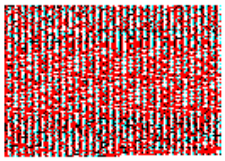
### 4.2 CPU アーキテクチャ分類

ここでは，プログラムコードの対象 CPU アーキテクチャの分類を行う．

この実験では，x86，ARM，MIPS および PowerPC の 4 種類の CPU アーキテクチャのネイティブコードを集めたデータセットを使用した．このデータセットは，クロスコンパイラである gcc[12] を使用し，以下の手順で作成した．

1 様々なソースコードが公開されている GitHub[13] から C または C++のソースコードを機械的に抽出
2 gcc を使用し，各アーキテクチャ用のオブジェクトファイルをコンパイル

表 5　aes.cpp のコンパイル結果 (1)
**Table 5**　Compile results of aes.cpp

| CPU | Bitmap View |
|---|---|
| x86 |  |
| ARM |  |
| MIPS |  |
| PowerPC |  |

3　コンパイルに成功したオブジェクトファイルのヘッダ情報を元にネイティブコード部分のみを抽出

上記の手順で，同一のソースコード（aes.cpp）から対象 CPU アーキテクチャを変えてコンパイルした結果を Bitmap View で可視化した結果を**表 5** に示す．

表を見ると，対象 CPU アーキテクチャごとに全く異なるパターンを示していることが分かる．この理由としては，RISC 型 CPU と CISC 型 CPU の違いであったり，ステータスフラグに基づいて実行の可否が判断される条件付き命令が実装された CPU があるなど様々な理由が上げられる．しかしながら，そのような前提知識がなかったとしても，表 5 のような見本表（チートシート）と見比べながらであれば，人の目でも対象 CPU アーキテクチャの推定は可能であると思われる．

実験で使用したデータセットの概要を**表 6** に示す．「種類」は CPU アーキテクチャ名，「ファイル数」はオブジェクトファイルから取り出したネイティブコードのファイル数を示す．対象とする CPU アーキテクチャによってはコンパイルに失敗するものがあったため，ファイル数には差異が出ている．「サンプル数」は，ネイティブコードのファイルを 256 Byte 毎に分割した場合のサンプル数を示している．

このデータセットを使用し，最適化アルゴリズムを変えながら，10 分割交差検証により正解率とその分散を求めた．NN の学習回数は 20 回であり，実験に使用した最適化アルゴリズムは前の実験と同様である．

実験の結果を**表 7** に示す．最も正解率が高い手法は

表 6　データセット B
**Table 6**　dataset B

| 種類 | ファイル数 | サンプル数 |
|---|---|---|
| x86 | 504 | 8,928 |
| ARM | 1,015 | 13,878 |
| MIPS | 710 | 21,436 |
| PowerPC | 713 | 15,257 |
| 合計 | 2,942 | 59,499 |

表 7　実験結果
**Table 7**　Results of the examination

| | Optimizers | accuracy | variance |
|---|---|---|---|
| SVM(RBF) | $C = 10, \gamma = 0.01$ | 0.9960 | 7.31E-7 |
| NN | SGD | 0.9920 | 2.17E-6 |
| | Momentum SGD | 0.9952 | 7.15E-7 |
| | AdaGrad | 0.9937 | 1.19E-6 |
| | RMSprop | 0.9950 | 2.76E-6 |
| | AdaDelta | 0.9963 | 6.60E-7 |
| | Adam | 0.9962 | 1.29E-6 |

NN(AdaDelta) で，99.63%となっていた．また，正解率の分散が最も低い値となった手法も NN(AdaDelta) であった．

## 5.　可視化ツールの作成

上記実験を踏まえて，解析対象ファイルの目 grep した結果を俯瞰できるように可視化するツール（o-glasses）を作成した．本可視化は，解析対象ファイルを 256 Byte ごとに分割し，分類器で予測されたクラスごとに色分けすることにより実現する．また，本論文では，大きく分けて SVM と NN の 2 つの手法を取り上げているが，本ツールで実装されている手法は暫定的に NN のみとなっている．

本ツールは，オープソースのスクリプト言語の Python[14] 2.7.12 で開発されたコマンドラインプログラムである．また，本ツールは Chainer[15] 2.01 を使用している．Chainer は，オープンソースで開発された深層学習フレームワークであり，2015 年 6 月に公開された．本ツールの処理の流れを**図 4** に示す．本ツールは大きく分けて 3 つの機能を実装している．

1 つ目は，任意のデータセットを指定し，k-分割交差検証によりデータセットが本ツールで使用するネットワークに適合しているか検証するモード（Mode 1) である．任意の指定したディレクトリ（-d オプション）に格納されたファイルから，ラベル付きの特徴ベクトル（データセット）を作成する．ファイルを 256 Byte ごとに分割して特徴ベクトルを作成する作業や，ラベル名を付ける処理はツールが自動で行う．このため，ファイルを種類に応じてフォルダに分類して保存するだけでデータセットを作成することができる．作成したデータセットのクラス数（ラベルの種類）に応じて，出力層のユニット数は自動的に適切な値に設定される．構築したネットワークを用いて，途中経過を表示

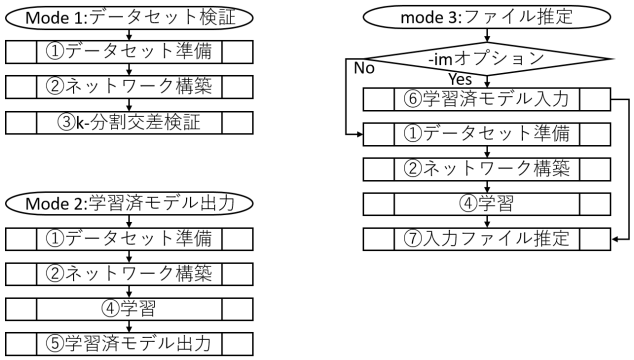



**図 4** ツールのフローチャート
**Fig. 4** Flowchart of the tool.

しながら k-分割交差検証を行う．なお，中間層のユニット数（-u オプション），k-分割交差検証の分割数（-k オプション）および学習回数（-e オプション) は任意の値に設定することができる．

2 つ目は，任意のデータセットを指定し，学習済モデルを出力するモード（Mode 2）である．Mode 1 と同様の手順でデータセット，ネットワークを構築し，作成したデータセットすべてを使用して学習を行う．学習後のネットワークを学習済モデルとして，任意の指定した名前（-om オプション）で保存する．なお，Mode 1 と同様に，中間層のユニット数および学習回数は任意の値に設定することができる．

3 つ目は，任意のデータセットまたは学習済モデルを使用し，任意のファイルの種類を推定するモード（Mode 3）である．学習済モデルのモデル名が指定（-im オプション）されている場合，その名前で保存されている情報を用いることで，Mode 2 の学習終了直後のネットワークの状態を再現する．学習済モデルのモデル名が指定されていない場合，Mode 2 と同様の手順でネットワークの構築および学習を行う．その上で，任意のファイルを読み込み，256 Byte ごとに分割し，それぞれのブロックでファイルの種類の推定を行い，その結果を表示する．すべてのブロックの推定が終わった後，推定された数が最も多いラベル名を表示する．

## 6. 目 grep の可視化の試行

ここでは，文書型のマルウェアの解析を想定し，目 grep の可視化を試行する．文書型のマルウェアには，閲覧ソフトのぜい弱性を突くか否かで大きく 2 種類に分けられる．ここでは，ぜい弱性を突く種類のマルウェアを想定する．この種のマルウェアを動的解析する場合，ぜい弱性を攻撃する Exploit を動作させるため，閲覧ソフトのバージョンや OS のバージョンなど複数の組み合わせの環境を用意する必要がある．一方，静的解析で文書型のマルウェアを解析する場合，文書ファイルの中から解析対象を絞り込む必要がある．その場合，Exploit が動作した後に実行されるシェルコードというプログラムコードに着目する．このシェルコードの位置の推定には，パターンマッチングが代表的な手法として上げられるが，未知のコードに対応できないという課題がある．本論文では，シェルコードを含めプログラムコードは通常，文書ファイル中に埋め込まれることは想定されないことに着目し，正常な文書ファイルと正常なプログラムコードを学習させた分類器で，文書型のマルウェアに埋め込まれたプログラムコードの大まかな位置が推定できるか試行する．

**表 8** データセット C
**Table 8** dataset C

| ラベル | ファイル数 | サンプル数 |
|---|---|---|
| CFB | 27 | 31,779 |
| Program | 1,147 | 49,577 |
| 合計 | 1,174 | 81,356 |

**表 9** 実験結果
**Table 9** Results of the examination

| | Optimizers | accuracy | variance |
|---|---|---|---|
| SVM(RBF) | $C = 10, \gamma = 0.01$ | 0.9787 | 2.87E-7 |
| NN | SGD | 0.9725 | 7.11E-6 |
| | Momentum SGD | 0.9769 | 1.87E-6 |
| | AdaGrad | 0.9716 | 2.33E-6 |
| | RMSprop | 0.9784 | 2.27E-6 |
| | AdaDelta | 0.9795 | 2.68E-6 |
| | Adam | 0.9792 | 3.36E-6 |

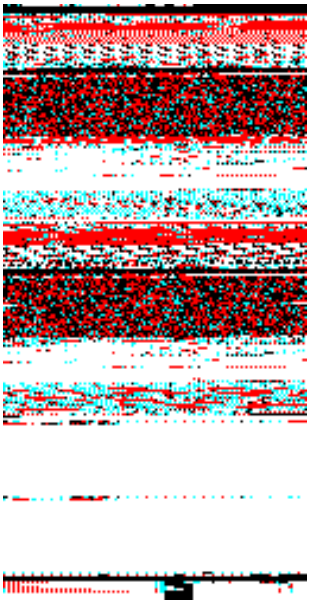
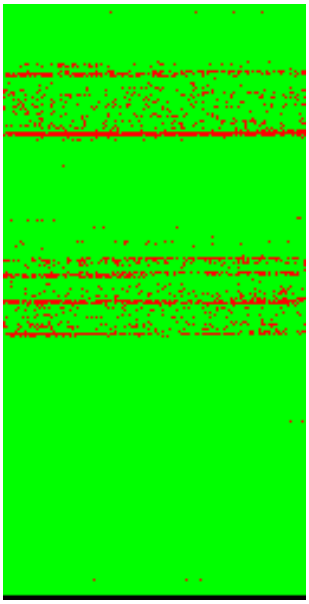
まず，インターネット上から収集した正常な CFB（Compound File Binary[16]）ファイル（doc 拡張子，xls 拡張子等）を「CFB」，分類器の性能評価実験と同様の手順で作成した正常なプログラムコードを「Program」とラベル付した教師データを作成する．作成したデータセットの概要を**表 8** に示す．

このデータセットを使用し，最適化アルゴリズムを変えながら，10 分割考査検証により正解率とその分散を求めた．NN の学習回数は 20 回であり，実験に使用した最適化アルゴリズムは前章と同様である．

実験の結果を**表 9** に示す．最も正解率の高い手法は NN（AdaDelta）で，97.95%となっていた．また，正解率の分散が最も低い値となった手法は SGD であった．

このデータセットをツールに入力し，Mode 2 により学習済モデルを作成する．学習回数は 20 回，最適化アルゴリズムは AdaDelta とした．作成した学習済モデルを使用し，Mode 3 により，入力された悪性文書ファイルの推定結果を表示する．

表 10　ある悪性文書ファイルに対する目 grep の可視化結果
**Table 10** Visualization of Intuitive Binary Classification against a malicious document

| Bitmap View | 目 grep |
|---|---|
|  |  |

ある悪性文書ファイルの推定結果を**表 10** に示す．目 grep の推定結果は，正常な CFB 部分と分類した部分を緑色，プログラムコードと分類した部分を赤色の 2 色で示している．この場合，シェルコードの大きさが連続した数百 Byte であると仮定すると，シェルコードが埋め込まれている可能性が高い場所を 1 ヶ所に絞りこむことができる．

## 7. 考察

### 7.1 目 grep の再現性

作成したツールは，正解率 99.63%という正解率でネイティブコードの CPU アーキテクチャを推定することができた．これは，本ツールに適切なデータセットを入力することで，誰でも高度人材が行う目 grep と同様の分析ができる可能性を示している．データセットの作成はファイルの種類ごとにフォルダに分類するだけであり，本論文で扱ったものとは別の分類問題でもある程度の正解率はすぐに達成できる．しかしながら，適切なデータセットの設計には，本論文で行なったようにファイルから必要な部分だけを切り出すなど，ある程度の経験が必要となってくる．

また，本論文では 2 種類の分類問題について SVM と NN の分類器の正解率を比較した．実験の結果，この 2 つの手法では，正解率に大きな差はなく，どちらの手法が目 grep に適しているかの結論は出ていない．今後，様々な分類問題について実験を行うことで，差が明らかになると思われる．

### 7.2 ツールの効果

#### 7.2.1 分類機能の性能

実験の結果，提案手法は複数の分類問題で 99%以上の正解率を示していた．この値が有効か否かについては，その目的により異なる．

例えば，対象 CPU アーキテクチャの推定を行う場合について以下に述べる．CPU アーキテクチャの推定は，対象のプログラムコード全体を 256 Byte ごとに分割し，それぞれの分類結果を集計し，最も多く分類されたクラスをプログラムコード全体の予測結果（多数決方式）とした場合，256 Byte のコードの断片の正解率から推測すると，ほぼ確実に CPU アーキテクチャの分類に成功することが予想される．この分類器の活用法としては，IoT ハニーポットが上げられる．IoT 機器に搭載される CPU アーキテクチャは ARM，MIPS など様々であり，IoT 機器を対象としたマルウェアも様々な CPU アーキテクチャに対応している．IoT 機器を対象とした攻撃の全体像を把握する手法としては，複数の CPU アーキテクチャに対応したハニーポットの構築があげられる．効率的に攻撃を観測するためには，様々な IoT 機器の動作を模倣するフロントエンドレスポンダで既知の命令の反応をエミュレーションし，未知の命令については対応する CPU アーキテクチャのサンドボックスで処理する方法がある [17]．未知の命令が実行ファイル形式で保存されていれば，ファイルヘッダ等の情報から，対象の CPU アーキテクチャを推測することができる．しかしながら，メモリ上にしか保存されない命令であった場合，その対象 CPU アーキテクチャを自動的に推測することは困難であった．しかしながら，提案手法は，コードの断片から CPU アーキテクチャを推測することができるため，メモリ上にのみ展開される命令コードの対象 CPU アーキテクチャを推測することができる．そのため，複数のアーキテクチャに対応した IoT ハニーポットの構築がより容易に可能となることが期待される．

一方，文書ファイルに埋め込まれたプログラムを抽出しようとした場合は，同じ正解率でも評価は異なる．この点については，分類結果の可視化の効果の考察部分で述べる．

#### 7.2.2 学習済モデルの入出力の効果

作成したツールは，学習済モデルの入出力に対応することで，データセットそのものを共有することなく，解析能力の共有を行うことができる．今回，実験に使用したデータセットはインターネット上から収集したものであり，機密情報を含まない．しかしながら，著作権の関係でデータセットをそのままの状態で共有することは難しいと思われる．さらに，実際に攻撃に使用されたマルウェアで作成されたデータセットや，組織内のファイルを正常なファイル群として作成したデータセットは，より共有しにくいと思われる．このように，データセットそのものは共有するには障壁があるものの，匿名化された学習済モデルであれば，

共有は比較的容易になると考えられる．

さらに，データセットのサイズは場合によっては非常に大きくなるが，本論文で使用したモデルであれば，学習済モデルのファイルサイズは数 MB 程度であり，ファイルサイズの面からも共有は比較的容易である．

加えて，学習済モデルの作成には，大規模なデータセットを使用したり，学習回数を多くした場合，大量のメモリや計算量が必要となる．しかしながら，学習済モデルを使用した推定だけであれば，一般的なノートパソコンでも十分高速に動作すると思われる．

#### 7.2.3 分類結果の可視化の効果

可視化部分の効果は現時点では明らかではない．今回，試行した悪性文書ファイルでは，シェルコードのおおまかな位置の推定に使用することができたが，毎回推定できるとは限らない．目 grep は表層解析の 1 種であり，解析作業の絞り込みを目的としたものである．現時点の可視化手法では，正解率が非常に高くない限り，自動的に位置推定することは難しい．何故ならば，仮に正解率が 99%としても，100KB のファイルを可視化すると単純計算で 1KB つまり 1,000 ヶ所も推定が誤っていることになる．シェルコードの位置の推定に限れば，エントロピーを使いシェルコードの位置の絞り込みをしていた岩本らの手法 [18] の絞り込みの代替措置とするなどすると，この問題が解決する可能性はある．シェルコードの位置の推定以外の場合にも対応できるよう，この誤りの根本的な解決法について，今後検討する必要がある．

## 8. おわりに

本論文では，高度人材の能力のうち，特にバイナリ解析分野の「目 grep」に焦点を当て，機械学習を用いることでその能力の再現を試みた．その結果，プログラムコードの断片から，その対象とする CPU アーキテクチャを推定するクラス分類では 99%以上の正解率で分類することに成功した．また，文書ファイルとプログラムコードを学習させた分類器を用いてプログラムコードが埋め込まれた悪性文書ファイルの分析結果を可視化したところ，埋め込まれたプログラムコードのおおまかな位置の推定ができた．

今後は，様々な分類問題について実験を行い，より目 grep に適した手法について評価したい．

### 参考文献


[1] 警察庁：平成 28 年中におけるサイバー空間をめぐる脅威の情勢等について，https://www.npa.go.jp/publications/statistics/cybersecurity/data/H28cyber_jousei.pdf (2017).

[2] 経済産業省：27 年度調査研究レポート（METI/経済産業省），http://www.meti.go.jp/policy/it_policy/jinzai/27FY_report.html (2016).

[3] サイバーセキュリティ戦略本部：サイバーセキュリティ人材育成プログラム，https://www.nisc.go.jp/active/kihon/pdf/jinzai2017.pdf (2017).

[4] Vapnik, V. and Lerner, A.: Pattern Recognition using Generalized Portrait Method, *Automation and Remote Control*, Vol. 24 (1963).

[5] 岡谷貴之：深層学習 (機械学習プロフェッショナルシリーズ)，講談社 (2015).

[6] Ioffe, S. and Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, *CoRR*, Vol. abs/1502.03167 (online), available from ⟨http://arxiv.org/abs/1502.03167⟩ (2015).

[7] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting., *Journal of Machine Learning Research*, Vol. 15, No. 1, pp. 1929–1958 (2014).

[8] Duchi, J., Hazan, E. and Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization, *Journal of Machine Learning Research*, Vol. 12, No. Jul, pp. 2121–2159 (2011).

[9] Tieleman, T. and Hinton, G.: Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural Networks for Machine Learning (2012).

[10] Zeiler, M. D.: ADADELTA: An Adaptive Learning Rate Method, *CoRR*, Vol. abs/1212.5701 (online), available from ⟨http://arxiv.org/abs/1212.5701⟩ (2012).

[11] Kingma, D. P. and Ba, J.: Adam: A Method for Stochastic Optimization, *CoRR*, Vol. abs/1412.6980 (online), available from ⟨http://arxiv.org/abs/1412.6980⟩ (2014).

[12] the GCC team: GCC, the GNU Compiler Collection, https://gcc.gnu.org/.

[13] GitHub: The world's leading software development platform, https://github.com/.

[14] Python Software Foundation: python, https://python.org/.

[15] Preferred Networks: Chainer: A Powerful, Flexible, and Intuitive Framework for Neural Networks, https://chainer.org/.

[16] Microsoft: [MS-CFB]: Compound File Binary File Format, https://msdn.microsoft.com/ja-jp/library/dd942138.aspx.

[17] Pa, Y. M. P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T. and Rossow, C.: IoTPOT: Analysing the Rise of IoT Compromises, *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C., USENIX Association, (online), available from ⟨https://www.usenix.org/conference/woot15/workshop-program/presentation/pa⟩ (2015).

[18] 岩本一樹，　和克己：文書型マルウェアに対するエントロピーとエミュレーションを用いたシェルコード特定方法，情報処理学会論文誌，Vol. 56, No. 3, pp. 892–902 (2015).