2015年度 修士論文

動的解析のDeep Learningによる
亜種マルウェア推定法

提出日：2016年2月1日

指導：後藤滋樹教授

早稲田大学 基幹理工学研究科 情報理工・情報通信専攻
学籍番号：5114F062-0

武部 嵩礼

目次

図一覧

表一覧

第 1 章

序論

1.1 研究の背景

マルウェアの数が増加の一途をたどり，ネットワーク社会の大きな課題となっている．マルウェア (Malware) とは英語の Malicious (悪意のある) と Software (ソフトウェア) からなる造語であり，コンピュータウィルスやトロイの木馬などユーザが望まない不正な働きをするソフトウェアの総称である [7]．大手セキュリティベンダの 1 つである McAfee によると，2015 年の第 1 四半期までに発見されたマルウェアは 4 億種類も存在する [1]．マルウェアにマシンが感染すると個人情報の流出の被害が発生してサイバー犯罪の被害者になるだけでなく，本人の気付かない間に感染マシンが不正アクセスや DoS (Denial of Service) 攻撃など別のサイバー攻撃に悪用され，サイバー犯罪の加害者になってしまう恐れもある [25]．たとえば，マルウェアがインターネットバンキングの認証情報やクレジットカードの情報といった個人情報を流出させた場合には，二次被害として金銭的な損失が生じ得る．警察庁の発表によると，平成 27 年上半期のインターネットバンキングにおける不正送金被害件数は 754 件，被害総額は約 15 億 4400 万円であった [3]．被害者が攻撃者になってしまう例としては，2015 年 6 月，長野県上田市の庁内 LAN に接続したパソコンがマルウェアに感染し，外部へ不正アクセスをするための踏み台に悪用された形跡があると発表された事案がある [4]．

マルウェアの種類は膨大であるが，その大半は既存のマルウェアを少しだけ改変した亜種である．2015 年の G DATA 社の報告によると，マルウェアの亜種が発見されるペースは 4 秒以内に 1 件という凄まじい速度である [2]．そのため，効率よくマルウェアを解析し，亜種を特定する技術が必要である．亜種の特定ができれば，既存のマルウェアへの対処法を適切に応用することで，マルウェアの脅威に対処していくことができる．

1.2　研究の目的

本研究では動的解析の結果に自然言語処理分野の Deep Learning の技術を応用することで，急増する亜種マルウェアを効率よく推定する手法を提案する．具体的には，マルウェアの動的解析ログから抽出した API コール列 (API 関数名を実行順に抽出した文字列) を Paragraph Vector によって特徴ベクトル化する．この特徴ベクトルを教師あり機械学習手法の 1 つである SVM (Support Vector Machine) に学習させてマルウェアの亜種を推定する方法を提案する．マルウェアの解析技術は静的解析 (デバッガや逆アセンブラによるコードレベルの分析) と動的解析 (実際にマルウェアを動作させる分析) に大別される．一般に動的解析の方が容易かつ短時間での解析が可能であり，急増するマルウェアにより早く対応可能である [23]．また，機械学習において，特徴量の取捨選択と特徴ベクトルの作成は重要であり，専門家の技能が要求される [5, 28, 26]．しかし，本提案手法では Paragraph Vector によって自動的に特徴ベクトルを作成可能である．したがって，本手法は属人性を極力排し，効率よく亜種マルウェアを推定できる．

1.3　論文の構成

本論文は以下の章により構成される．

第 1 章 序論

本論文の概要を述べる．

第 2 章 Paragraph Vector

Paragraph Vector について解説する．

第 3 章 関連研究

本論文に関連する研究を紹介する．

第 4 章 提案手法

本論文の提案手法を説明する．

第 5 章 実証実験

実験の概要を説明し，その結果を提示・考察する．

第 6 章 結論

本論文の結論を述べ，残された課題を示す．

第 2 章

Paragraph Vector

本章では提案手法に利用する Paragraph Vector という技術について説明する．Paragraph Vector は Le ら [22] によって提案された自然言語処理における Deep Learning の手法であり，文書をベクトル化し，文書同士の関係性をベクトル空間上で表現する手法である．この手法は単語のベクトル表現を学習する Word Vector を拡張した技術であり，ニューラルネットワークの理論を基礎として生み出された．

そこで，まず第 2.1 節で技術的な背景にあるニューラルネットワークについて簡単に説明し，第 2.2 節で単語をニューラルネットワークの入力に適した形式に変換する方法を紹介する．続いて，第 2.3 節で Paragraph Vector の元となった Word Vector の概要を，第 2.4 節でそのアルゴリズムを解説する．最後に，第 2.5 節と第 2.6 節で Paragraph Vector の概要とアルゴリズムを記述する．

一般に，Deep Learning ではニューラルネットワークの層を多層にすることで，学習対象の特徴量を獲得し，様々な分類タスクを実現する．自然言語処理の分野における Deep Learning は，単語の意味表現を固定長のベクトルで表現し，与えられたコーパスにおいて単語の出現を推定する精度を最大化するように学習する [27]．Paragraph Vector の学習構造は多層ニューラルネットワークではないため，厳密には Deep Learning と言うことはできない．しかし，文書ベクトルを学習する仕組みから，本論文では Paragraph Vector を自然言語処理における広義の Deep Learning の技術として扱っている．

2.1　ニューラルネットワーク

ニューラルネットワークとは脳の神経系を模した機械学習の 1 種で，ニューロンの層が層間で結合した構造を持つ．図 2.1 にニューロンの例を，図 2.2 にニューラルネットワークの例を示す．また，ニューロンの入出力の関係式を式 2.1 に示す．

$$u = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n, z = f(u) \tag{2.1}$$

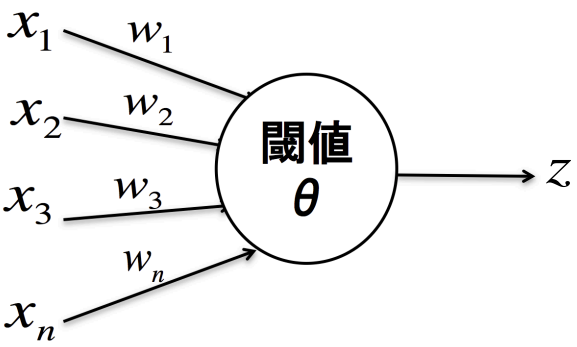

図 2.1: ニューロンの例

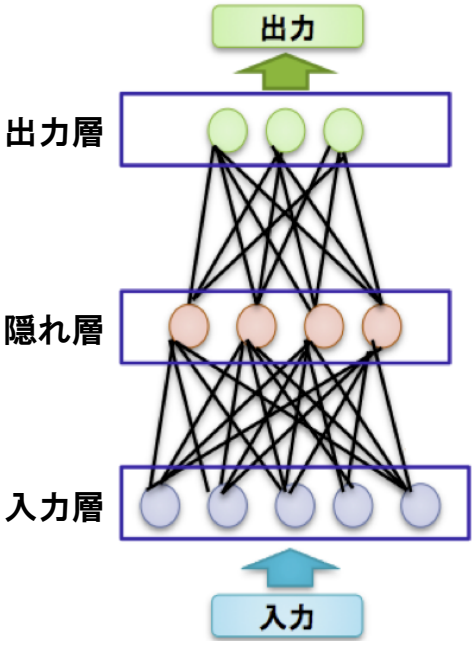

図 2.2: ニューラルネットワークの例

w_1，w_2，w_3，…，w_n はニューロン結合の重みであり，x_1，x_2，x_3，…，x_n は各ニューロンへの入力である．また，関数 f を活性化関数と呼ぶ．ニューラルネットワークでは与えられた入力に対して各ニューロンで一定の重み付け計算を行い，その結果が閾値を超えていたら次のニューロンに渡す，という風に計算をして行く．ニューラルネットワークを用いた教師あり学習では，入力データとその入力データが来た時にどのような出力データが欲しいかを考え，出力データが期待通りになるように重みの値を調整していくことになる [24].

2.2　1-of-K 符号化

文章をコンピュータで扱う場合，各単語を扱いやすくするため，単語と整数値の対応表を作成し，文章を整数値の配列によって表現する方法がある．しかし，この方法だけでは，単語の距離を数値として扱うことはできない．単語と整数値の対応には規則性はなく，その数値が近いことと，単語の意味が近いということとは一致しない.

この問題を解決するために生まれたのが，1-of-K 符号化である．単語に割り当てる 1 から K までの整数値を K 次元のベクトルで表現することにより，どの単語の距離も等しくなり，単語を数値として扱うことを可能にした [24]．たとえば，K=6 の場合を考える．すると，1 は $(1,0,0,0,0,0)$，2 は $(0,1,0,0,0,0)$，6 は $(0,0,0,0,0,1)$ と表現することができる．1 と 2 の間の数値的な距離は 1 であり，1 と 6 の間の数値的な距離は 5 である．しかし，1 と 2 のベクトル同士の距離および 1 と 6 のベクトル同士の距離はどちらも $\sqrt{2}$ と等しくなる．他のベクトル同士も距離は一定であるため，単語ごとの距離の偏りを考える必要がなくなる.

なお，文章中に表れる単語をすべて 1-of-K 符号化し，足し合わせることで文章を表現することを Bag-of-Words 表現と呼ぶ.

2.3　Word Vector の概要

Word Vector は Mikolov ら [18, 21] によって提案された単語の意味表現をベクトル化したものである．Word Vector の画期的な点は，単語をベクトルで表現することによって，単語同士の関係性をベクトル演算によって求めることができる点である．1-of-K 符号化を施された単語ベクトル同士の距離はすべて一定であったが，Word Vector によってベクトル化された単語は，意味的に関連が強い単語はベクトルが近くなり，２つのベクトルの差は２つの単語の関係を表すという特徴を示す.

図 2.3 を具体例に説明する．たとえば，BIG (大きい) と LARGE (大きい)，LITTLE (小さい) と SMALL (小さい) はそれぞれ同様な意味であるため，ベクトル空間上で近くに位置している．また，これらは形容詞であり，名詞である MAN (男)，KING (王)，WOMAN (女)，QUEEN (女王) の単語とは意味が大きく異なるため，ベクトル空間上での距離は離れている．ベクトル演算が成り立つ例としては，MAN (男) から WOMAN (女) に向かう矢印が性差を表すため，そのベクトルを KING (王) という単語に足すと QUEEN (女王) になる．さらに，単数名詞を表す単語に対し，右斜め上に延びる赤いベクトルを足すと，複数形名詞になる．つまり，この図 2.3 における単語同士のベクトル演算は下記の式 2.2，式 2.3，式 2.4 通りに記述することができる．なお，3 つの式において，wvがつく単語はその単語自身の Word Vector を意味する．

Mikolov らの理論に基づき Word Vector を作成するソフトウェアとして，word2vec [42] というオープンソースの実装が公開されている．

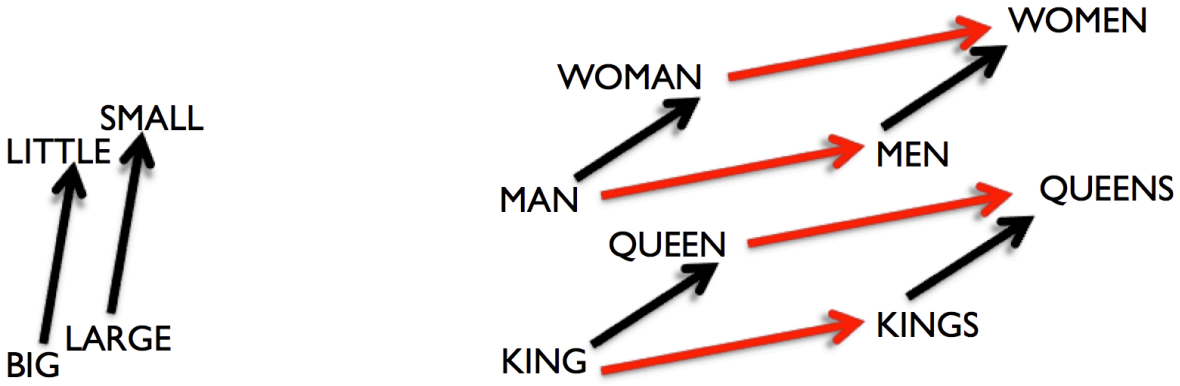

図 2.3: Word Vector 化された単語の関係

$$wv(\text{“ }KING\text{ ”}) - wv(\text{“ }MAN\text{ ”}) + wv(\text{“ }WOMAN\text{ ”}) = wv(\text{“ }QUEEN\text{ ”}) \tag{2.2}$$

$$wv(\text{“ }QUEEN\text{ ”}) - wv(\text{“ }KING\text{ ”}) + wv(\text{“ }KINGS\text{ ”}) = wv(\text{“ }QUEENS\text{ ”}) \tag{2.3}$$

$$wv(\text{“ }WOMAN\text{ ”}) - wv(\text{“ }MAN\text{ ”}) + wv(\text{“ }MEN\text{ ”}) = wv(\text{“ }WOMEN\text{ ”}) \tag{2.4}$$

2.4　Word Vector のアルゴリズム

Word Vector を作成するために利用される 2 つのモデルを紹介する．両モデルで扱う言葉と記法について事前に簡単に説明しておく．学習に利用する文章をコーパスと呼び，コーパスに出現する単語を順番に w_1，w_2，w_3，$\ldots$,w_T で表す．この各単語は既に第 2.2 節において説明した 1-of-K 符号化によって学習の入力に適した数値ベクトル形式となっているものとする．また，w_1，w_2，w_3，$\ldots$,w_T の列ベクトルからなる行列を W とする．

2.4.1　CBOW モデル

CBOW (Continuous Bag-of-Words) モデルを図 2.4 に示す．

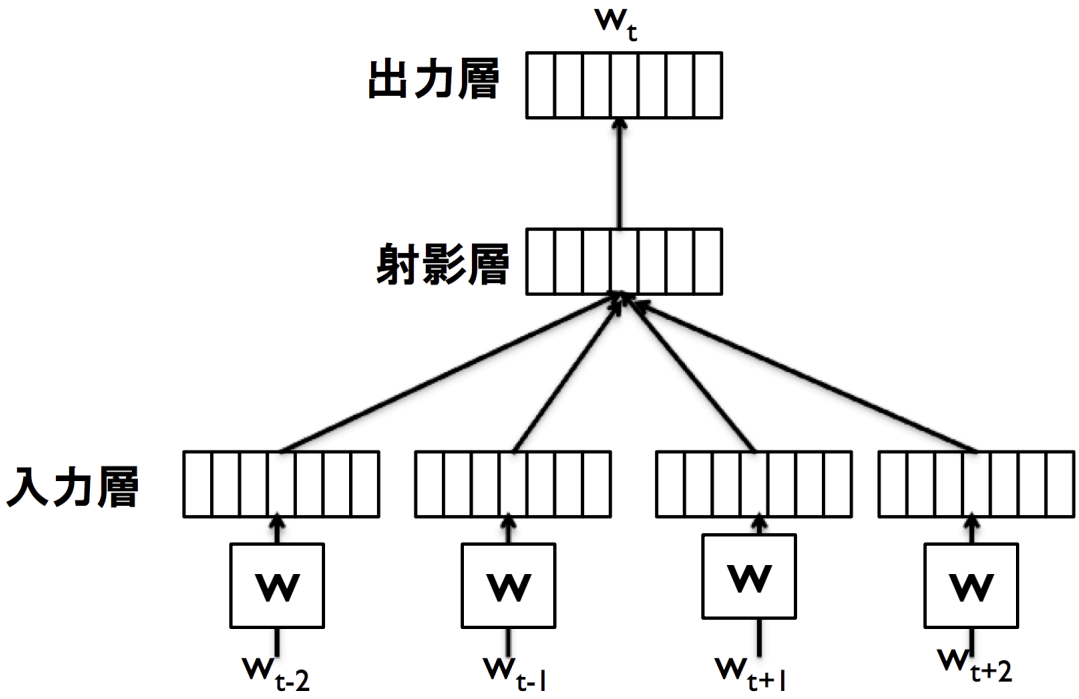

図 2.4: CBOW モデル

CBOW モデルでは Word Vector 化したい単語 w_t の前後に存在する $2k$ 個の単語を文脈と呼び，この文脈の Bag-of-Words 表現が入力に相当する．そして，単語 w_t が出力層に出現する確率を求めるよう，ニューラルネットワークの重みを調整しながら学習を進める．数式で表現すると，CBOW モデルにおける Word Vector は式 2.5 の対数尤度の平均値を最大化するように学習する [22].

$$\frac{1}{T}\sum_{t=k}^{T-k}\log p(w_t|w_{t-k},\ldots,w_{t+k}) \tag{2.5}$$

この出現確率の計算には softmax 関数による分類のような多クラス分類が使われることが多い．したがって，尤度の計算は式 2.6 のように計算される．なお，U，b を softmax 関数のパラメータ，h を単語行列 W の文脈内に存在する単語ベクトルの平均値とすると，y_i は i 番目に出力される単語 w_i を用いた非正規対数尤度 (式 2.7) により計算される．

$$\log p(w_t|w_{t-k},\ldots,w_{t+k}) = \frac{\mathrm{e}^{y_{w_t}}}{\sum_i \mathrm{e}^{y_i}} \tag{2.6}$$

$$y = b + Uh(w_{t-k},\ldots,w_{t+k};W) \tag{2.7}$$

2.4.2　Skip-gram モデル

Skip-gram モデルを図 2.5 に示す．なお，初出の論文 [18] では Continuous Skip-gram モデルという名称だが，その後の論文では単に Skip-gram モデルという名称で扱われているため，本論文でもそれに倣う．

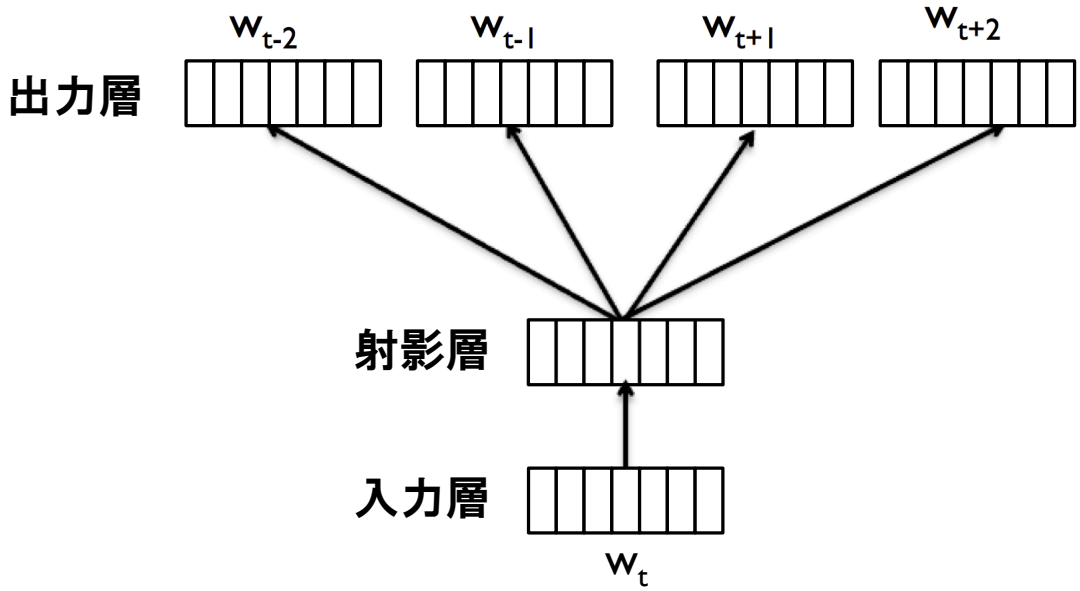

図 2.5: Skip-gram モデル

CBOW モデルは入力層に Word Vector 化したい単語の前後に存在する単語 (文脈) の情報が与えられ，その情報を元に出力層に出現する単語を推定するという構造であった．Skip-gram モデルの構造はその逆となっている．すなわち，Word Vector 化したい単語 w_t を入力層に与え，出力層では文脈中に出現する他の単語 w_{t+k} を推定できるように学習を行う [24]．数式で Skip-gram モデルを表すと，式 2.8 の対数尤度の平均値を最大化するような学習モデルとなる [19]．式 2.8 において，c は学習する文脈のサイズである．また，Skip-gram モデルにおいて，確率 $p(w_{t+j}|w_t)$ の計算は softmax 関数が利用されるため，式 2.9 のようになる．なお，式 2.9 において，w_O と W_I はそれぞれ出力層の単語と入力層の単語を，W が語彙数を意味している．また，v_w と v'_w はそれぞれ単語 w の入力時の単語ベクトル（Word Vector を学習したい単語の 1-of-K 符号化表現）と出力時の単語ベクトル（単語の Word Vector）である．

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{2.8}$$

$$\log p(w_O|w_I) = \frac{\mathrm{e}^{{v'_{w_O}}^{\mathrm{T}} v_{w_I}}}{\sum_{w=i}^{W}\mathrm{e}^{{v'_w}^{\mathrm{T}} v_{w_I}}} \tag{2.9}$$

射影層での出力は入力単語が決まれば一意に決まる．そのため，射影層の次元数を設定することで，その設定した次元の単語ベクトルが作成される．たとえば，語彙数が 20 万次元であり，射影層の次元を 200 次元に設定した際の単純化した Skip-gram モデルを図 2.6 に示す．図 2.6 のように，Skip-gram モデルは大量の語彙からなるコーパス全体を各単語と紐づいた固定長のベクトルに圧縮する操作であると言うことができる [24].

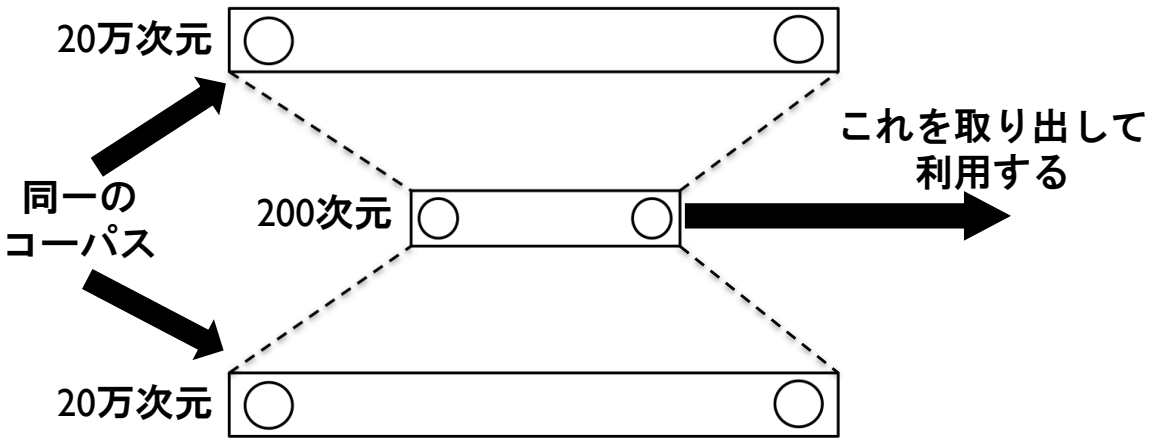

図 2.6: 単純化した Skip-gram モデルの概念図

2.5　Paragraph Vector の概要

Paragraph Vector は Le ら [22] によって提案された文章の意味をベクトルで表現する技術のことである．Word Vector は 1 つ 1 つの単語の意味をベクトルで表現していた．それに対し，Paragraph Vector は Word Vector を元に，文章全体を 1 つのベクトルで表現する．Word Vector と同様に，ベクトル化された文章は，意味的に関連が強い文章はベクトルが近くなり，2 つのベクトルの差は 2 つの文章の関係を表すという特徴を示す．また，Paragraph Vector は Word Vector を拡張した技術であるため，Paragraph Vector と Word Vector 同士のベクトル演算も可能という特徴を持つ．

具体例として，Dai ら [20] の研究報告をとりあげ，図 2.7 によって説明する．なお，図 2.7 に書かれている人名は，その人物に関して書かれた文章の Paragraph Vector であるが，国名を意味する形容詞 (American と Japanese) はその単語の Word Vector である．

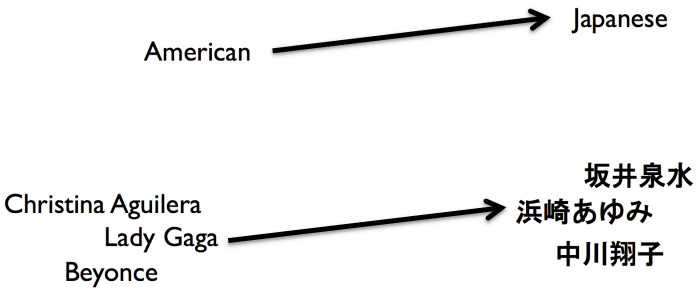

図 2.7: Paragraph Vector 化された文章の関係

たとえば，Lady Gaga，Christina Aguilera，Beyonce の 3 人はいずれも有名なアメリカの女性ミュージシャンであるという共通項を持つ．そのため，彼女たちのことを説明した文章 (出典は Wikipedia [40] の記事) の Paragraph Vector の距離はベクトル空間上で近くなる．また，浜崎あゆみ，中川翔子，坂井泉水の 3 人はいずれも有名な日本の女性ミュージシャンである．したがって，彼女たちのことを説明した文章 (出典は Wikipedia [40] の記事) の Paragraph Vector の距離もベクトル空間上で近くなる．さらに，日本人とアメリカ人の差分を表す右斜め上方向のベクトルを Lady Gaga に足すと，浜崎あゆみとなる．つまり，Lady Gaga に相当する日本人は誰か？という疑問を式 2.10 によって解決できるのである．なお，式 2.10 において pv がつく単語はその単語を説明した文章の Paragraph Vector を意味し，wv がつく単語はその単語自身の Word Vector を意味する．

Le らの理論に基づき Paragraph Vector の機能を提供する自然言語処理用ライブラリとして，

gensim [44] が存在している．本研究ではこの gensim に基づいた実装である sentence2vec [43] を利用して Paragraph Vector を作成する．

$$pv(\text{“ } Lady\ Gaga \text{ ”}) - wv(\text{“ } American \text{ ”}) + wv(\text{“ } Japanese \text{ ”}) = pv(\text{“ 浜崎あゆみ ”}) \qquad (2.10)$$

2.6　Paragraph Vector のアルゴリズム

本節では Paragraph Vector を作成するために利用される 2 つのモデルを紹介する．Word Vector の時と同様に，学習に利用する文章をコーパスと呼び，コーパスに出現する単語 (1-of-K 符号化済み) を順番に w_1，w_2，w_3，$\ldots$,w_T で表す．また，w_1，w_2，w_3，$\ldots$,w_T の列ベクトルからなる行列を W とする．Paragraph Vector では新たに Paragraph Matrix D を導入する．D は 1-of-K 符号化された各 Paragraph の ID を表す列ベクトル d_1，d_2，d_3，$\ldots$，$d_{T'}$ から構成される．

2.6.1　PV-DM モデル

PV-DM (Distributed Memory Model of Paragraph Vecotor) モデルを図 2.8 に示す．

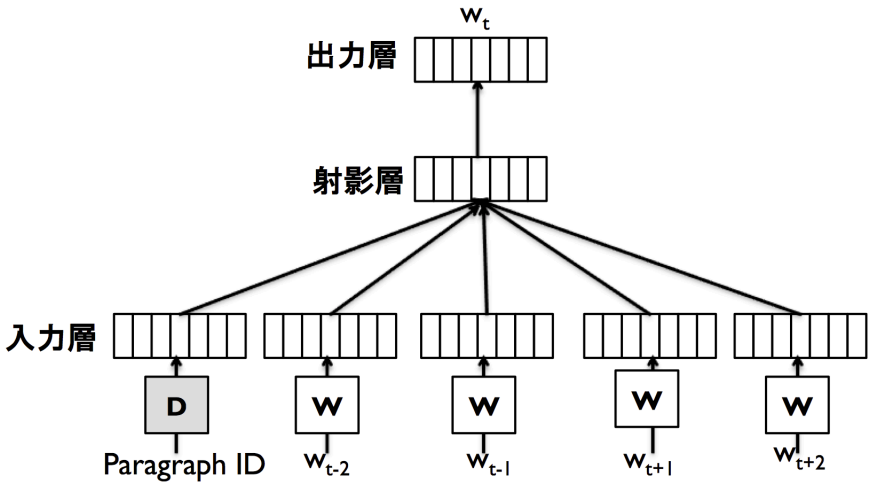

図 2.8: PV-DM モデル

基本的な学習の構造は CBOW と同様である．唯一異なる点は Word Vector を学習する時の入力層に Paragraph の情報も加える点である．つまり，出力層に出現する単語 w_t の前後に存在

する $2k$ 個の単語と Paragraph ID を入力層に与える．そして，出力層に出現する単語 w_t を推定する精度を最大化するよう，ニューラルネットワークの重みを調整しながら Paragraph Vector の学習を進めていく．

2.6.2　PV-DBOW モデル

PV-DBOW (Distributed Bag of Words Model of Paragraph Vector) モデルを図 2.9 に示す．

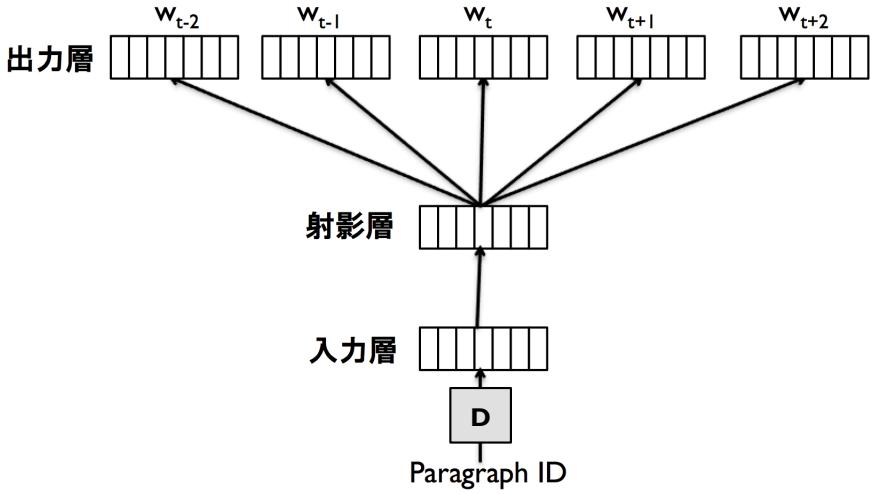

図 2.9: PV-DBOW モデル

PV-DBOW モデルの基本的な構造は Skip-gram と同様である．Skip-gram モデルでは入力層に Word Vector 化したい単語を与え，出力層で文脈内の単語を推定できるように学習をして Word Vector を作成した．PV-DBOW モデルでは入力層に Paragraph ID を与え，出力層で文脈内の単語を推定できるように学習を繰り返し Paragraph Vector を作成する．

なお，本提案手法で利用する Paragraph Vector はこの PV-DBOW モデルに基づいている．

第 3 章

関連研究

3.1 マルウェアの検出に関する研究

マルウェアの検出に関する研究は広範に行われている．ここでは特に，本研究と関連する自然言語処理の技術を応用した研究について，静的解析によるものを 1 件，動的解析によるものを 1 件，紹介する．

Kolter ら [11] はプログラムのバイナリに対して N-gram の手法を適用し，特徴量を作成し，様々な教師あり機械学習手法と組み合わせることによってマルウェアを検出する手法を提案している．

青木ら [13] は動的解析結果から API コール列を抽出し，その API コール列に自然言語処理の手法である N-gram と機械学習法の 1 つである決定木を適用し，マルウェアごとに検出精度を向上させる N の値が存在することを示した．

3.2 亜種マルウェアの推定に関する研究

マルウェアをファミリーごとに分類し，亜種を推定する研究も活発に行われている．ここでは本研究と関連する動的解析の結果を元にした亜種マルウェアの推定に関する研究を 3 件紹介する．

中村ら [12] はマルウェアの動的解析結果から得られる API コール列の N-gram を抽出し，各検体ごとの N-gram の出現回数をディリクレ分布に変換することでマルウェアの挙動を表現した．そしてディレクリ分布の類似度を Kullback-Leibler 情報量によって評価することで亜種マルウェアを高精度で推定できることを示した．

堀合ら [9, 10] はマルウェアの API コール情報ではなく，動的解析結果の前後で変化した OS 内の差分情報に着目した．動的解析前後のログ情報の差分を多次元ベクトルに変換した後，そのベクトルのハミング距離を計算し，ハミング距離が近いものを亜種マルウェアと推定する手法を提案している．

Liang ら [8] はマルウェアの動的解析結果から API コール情報だけでなく，ファイル操作，レジストリ書き替え記録，通信の情報を加工し，Jaccard 係数による類似度計算を行う手法により，亜種マルウェアを推定する手法を提案している．

3.3　本研究の着眼点

第 3.1 節と第 3.2 節でも取り上げたように，動的解析と自然言語処理の手法である N-gram を組み合わせて一定の効果を上げたマルウェア解析手法が存在している．しかし，N-gram の手法は API コール列中の API 関数の呼び出しパターンを N 個の固まりごとに数えているだけであり，その呼び出しパターンがどのタイミングで出現したかという情報までは考慮していない．また，事前に API コールパターンを調べた後，それをどのように特徴ベクトル化すべきかを研究者が考え，技巧的な変換をする必要も存在している．

本手法で利用する Paragraph Vector を利用すれば，呼び出された API 関数の最初から最後までの順番すべてを考慮した上で特徴ベクトルを作成することができる．さらに，API コール列をそのまま入力とすることで，Paragraph Vector 化ができるため，効率的なマルウェア解析が可能となる．

第 4 章

提案手法

4.1 提案手法の解析手順

本提案手法の解析手順は 3 段階に分けることができる．STEP1 では動的解析ログから API コール列を抽出し，その API コール列を Paragraph Vector 化することで特徴ベクトルを作成する．STEP2 では得られた特徴ベクトルを SVM に入力として与えることで，マルウェアの種別ごとの動作を学習させる．STEP3 では学習済み SVM モデルによって未知のマルウェアが亜種か否かを推定する．

各 STEP について，以下の第 4.1.1 節，第 4.1.2 節，第 4.1.3 節で詳しく説明する．

4.1.1 STEP1 : 特徴ベクトルの作成

本節ではマルウェアの種類を識別するための情報を有する特徴ベクトルの作成方法について説明する．まず，マルウェアごとに，動的解析結果から API 関数名を実行された順番に並べ，API コール列を作成する．この API コール列がマルウェアの挙動を表している．API コール列の例を図 4.1 に示す.

続いて，各 API 関数名に対応する Word Vector をすべての Paragraph (マルウェアの API コール列) を利用して計算しておく．たとえば，API 関数名 LdrLoadDll を Word Vector によって 100 次元の特徴ベクトルにしたものを図 4.2 に示す.

そして，計算済みの Word Vector と Paragraph (マルウェアの API コール列) を利用して，Paragraph Vector (マルウェアの特徴ベクトル) を計算していく．本研究における Paragraph Vector 学習の枠組みを図 4.3 に示す．文脈中の API 関数を推定できるよう，Paragraph Vector は文脈のウィンドウをずらしながら学習する．これにより，長さが一定でない API 関数の並

LdrLoadDll LdrLoadDll LdrLoadDll NtCreateFile NtCreateFile NtCreateFile NtAllocateVirtualMemory RegOpenKeyExA
LdrLoadDll LdrGetProcedureAddress NtDelayExecution NtOpenKey NtQueryValueKey NtQueryValueKey NtClose LdrLoadDll
LdrGetProcedureAddress RegOpenKeyExW RegCreateKeyExW RegDeleteKeyW RegQueryValueExW RegQueryValueExW
RegQueryValueExW RegQueryValueExW RegQueryValueExW RegQueryValueExW RegQueryValueExW RegQueryValueExW
RegQueryValueExW RegQueryValueExW RegQueryValueExW RegQueryValueExW RegCreateKeyExW RegCreateKeyExW
RegCloseKey RegOpenKeyExW RegQueryValueExW RegCloseKey FindFirstFileExW NtClose FindFirstFileExW NtClose
FindFirstFileExW NtClose FindFirstFileExW NtClose LdrGetDllHandle LdrLoadDll LdrGetDllHandle LdrLoadDll
LdrGetDllHandle LdrLoadDll LdrGetDllHandle LdrLoadDll RegQueryValueExW NtAllocateVirtualMemory RegOpenKeyExW
RegOpenKeyExA RegQueryValueExA RegCloseKey RegOpenKeyExW RegOpenKeyExW LdrLoadDll NtAllocateVirtualMemory
NtOpenFile DeviceIoControl NtAllocateVirtualMemory NtOpenKey NtOpenKey NtQueryValueKey NtClose NtClose
NtCreateFile NtSetInformationFile NtSetInformationFile NtWriteFile NtReadFile NtClose NtClose NtCreateFile
NtSetInformationFile NtSetInformationFile NtWriteFile NtReadFile NtClose NtClose NtClose NtClose LdrGetDllHandle
LdrGetDllHandle LdrGetDllHandle NtCreateMutant NtCreateSection LdrGetDllHandle LdrLoadDll LdrGetDllHandle
LdrLoadDll LdrGetDllHandle LdrLoadDll LdrGetDllHandle LdrLoadDll NtCreateFile NtClose NtCreateSection
NtAllocateVirtualMemory NtAllocateVirtualMemory NtCreateSection NtClose NtClose LdrLoadDll LdrGetProcedureAddress
NtAllocateVirtualMemory LdrLoadDll LdrGetProcedureAddress NtCreateFile NtAllocateVirtualMemory FindFirstFileExW
NtWriteFile NtWriteFile NtQueryDirectoryFile LdrLoadDll LdrGetProcedureAddress NtQueryDirectoryFile NtClose
FindFirstFileExW NtQueryDirectoryFile NtQueryDirectoryFile NtWriteFile NtClose NtClose NtCreateFile
NtQueryInformationFile NtSetInformationFile FindFirstFileExW NtWriteFile NtOpenFile NtQueryDirectoryFile NtClose
LdrLoadDll LdrGetProcedureAddress LdrGetDllHandle LdrLoadDll LdrGetProcedureAddress LdrGetDllHandle LdrLoadDll
LdrGetProcedureAddress NtCreateFile NtQueryInformationFile NtCreateSection NtClose NtClose NtWriteFile
NtQueryDirectoryFile NtClose NtWriteFile NtClose CreateProcessInternalW NtClose

図 4.1: API コール列の例

0.203738 -0.023942 -0.208127 0.054912 -0.160867 -0.246226 0.269836 -0.109756 -0.133437 0.007321
-0.170173 -0.206467 -0.142925 -0.279942 0.292689 0.039801 -0.042531 0.145757 -0.043004 0.119539
0.444994 -0.308486 0.074986 -0.247011 0.087566 0.084296 -0.186915 -0.046338 0.067327 0.115546
-0.149958 -0.172055 0.124770 -0.036777 -0.133902 0.044454 -0.111959 0.380621 -0.304234 -0.237562
-0.243970 -0.038268 0.117714 -0.076579 -0.123083 -0.326448 0.040348 0.002211 0.040920 0.156814
0.001432 0.014695 -0.009557 -0.251516 0.144287 0.321070 -0.157555 -0.096549 0.427150 -0.257815
0.184450 0.078356 0.020977 -0.168283 0.462095 -0.000660 0.174512 0.129233 0.111268 -0.227443
0.135514 -0.114792 -0.110484 -0.276995 -0.141170 -0.222641 0.059408 -0.055203 -0.234310 -0.031424
-0.036605 0.057216 -0.123359 -0.420128 -0.363120 -0.131149 0.122617 0.068896 0.113879 0.263075
0.155383 0.050319 0.192273 0.001226 0.222630 0.061893 0.034269 -0.176266 -0.055438 0.006339

図 4.2: Word Vector 化した API 関数名の特徴ベクトル（次元：100）の例

びを文脈も考慮した上で，固定長のベクトルに情報圧縮することができる．また，Paragraph Vector 化されたマルウェアの特徴ベクトルの例を図 4.4 に示す．

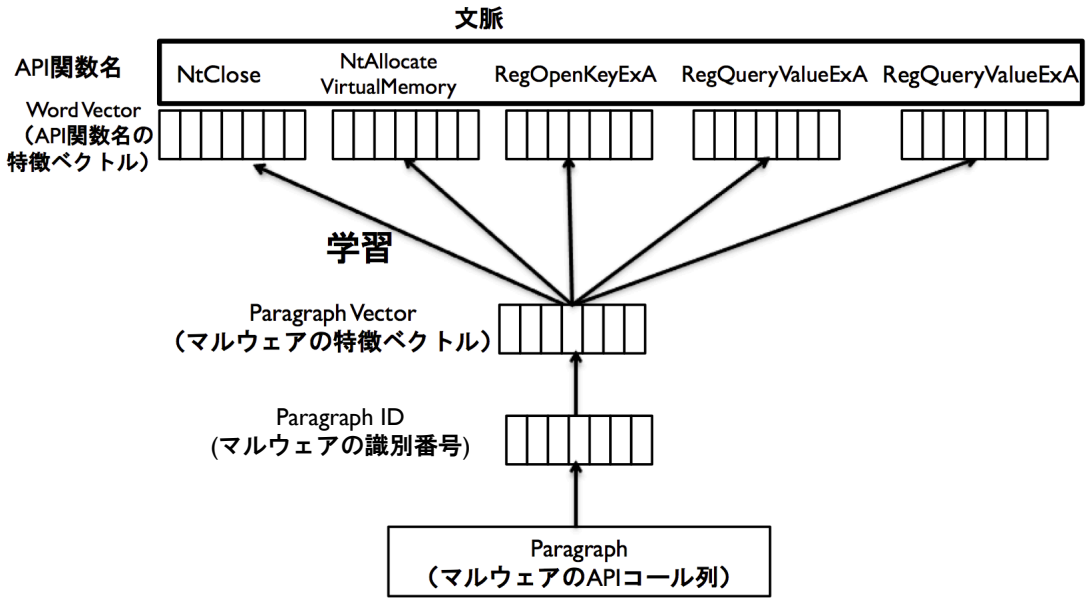

図 4.3: Paragraph Vector 学習の枠組み

-0.293371 -0.863331 -0.094008 0.475440 -0.144030 -0.203962 -0.166426 -0.342172 -0.273625 -0.223718
-0.011495 0.167099 0.820696 -0.619965 -0.152888 -0.184464 -0.478918 -0.301870 -0.356803 0.154230
0.076946 -0.338971 -0.473721 -0.021982 0.031730 0.043945 -0.279592 -0.589304 -0.491773 -0.067885
-0.520550 0.694502 -0.210480 0.491052 -0.104230 -0.585667 1.286032 0.266802 0.062833 0.390583
0.083811 -0.146313 0.585438 0.419949 0.615461 0.156701 0.667188 0.370974 0.077126 -0.061724
-0.302520 -0.849826 -0.052742 0.490824 -0.232289 -0.202784 0.009455 0.674149 -0.380789 0.518072
0.471781 -0.396228 0.045794 0.045412 -0.262772 -0.996261 0.030383 -0.243508 -0.680418 -0.327085
-0.624227 -0.414947 -0.384970 -0.183048 0.521885 0.117406 0.012220 0.514346 -0.623533 0.020475
-0.158889 -0.429264 -0.823038 -0.079238 -0.295884 0.250070 0.170756 0.247150 0.081034 -0.712944
-0.255927 -0.115084 -0.001388 0.296091 0.442982 0.054619 0.205205 0.021312 0.449073 0.342061

図 4.4: Paragraph Vector 化したマルウェアの特徴ベクトル（次元：100）の例

4.1.2　STEP2：マルウェアの特徴学習

本節ではマルウェアの特徴ベクトルを学習する方法について説明する．本研究では，マルウェアの亜種であるか否かを推定することを目的とする．そこで，SVM の分類ラベルとしては，亜種マルウェア (マルウェアファミリー名)，その他のマルウェア (Others)，の 2 種類を用意し，推定したい亜種マルウェアごとに分類器を作成する．図 4.5 に SVM に与える学習用データの例を示す．これはマルウェアが Trojan.Win32.Agent の亜種か否かを推定する分類器を作成するためのデータである．一番左側が学習に使うラベル名であり，その右側以降がそのマルウェアの特徴ベクトルとなる．この例で Others のラベルは Trojan.Win32.Agent 以外の様々な種類のマルウェアを意味している．

```
Trojan.Win32.Agent 0.044356 0.276648 -0.011800 0.065574 0.100789 ·····
Trojan.Win32.Agent -0.192727 -0.877096 -0.143621 -0.429818 -1.176162 ·····
Trojan.Win32.Agent 0.013475 -0.042282 0.208355 0.000483 0.054298 ·····
Trojan.Win32.Agent -0.120991 0.063815 0.062430 0.096735 0.080308 ·····
Trojan.Win32.Agent 0.186922 -0.066011 0.063734 0.099576 0.102097 ·····
                                    .
                                    .
                                    .
Others 0.535913 -0.547289 0.260173 -0.297913 0.184655 ·····
Others 0.737463 -0.119688 -0.154586 0.292967 -0.196917 ·····
Others 0.021600 -0.040796 0.006373 0.026232 0.063080 ·····
Others 0.899182 -0.648451 0.306486 -0.359366 0.316857 ·····
Others 0.927722 -0.606073 0.290617 -0.338055 0.315011 ·····
                                    .
                                    .
                                    .
```

図 4.5: SVM に与える学習用データの例

4.1.3　STEP3：マルウェア亜種の推定

本節では学習済みの SVM モデルを利用して，マルウェアの亜種を推定する方法について説明する．亜種か推定したいマルウェアを図 4.4 で示したように特徴ベクトル化し，テストデー

タとして学習済み SVM モデルにそのまま入力として与える．SVM モデルは STEP2 で学習したマルウェアの特徴に基づき，与えられたデータが推定したい亜種マルウェアであるか否かを判定する．

4.2　SVM

本節では第 4.1 節の提案手法で利用している SVM (Support Vector Machine) に関して，簡単に解説する．SVM は教師あり機械学習手法の 1 つであり，2 クラス分類問題に対して実験的にすぐれた成績を出すパターン識別器である．SVM の学習は二次計画問題を解くことに帰着できるため，局所解の問題は存在しない．また，SVM は本来線形識別器であるが，問題ごとにカーネルと呼ばれる関数を変えることによって，非線形の識別問題にも対応することが可能である [6]．カーネル関数には線形カーネル，多項式カーネル，ガウシアンカーネルなどがある．たとえば，赤と青の点が存在する領域を SVM で 2 つに分類したい場合，線形カーネルを利用すると図 4.6 の左側のように直線で分類し，ガウシアンカーネルを利用すると図 4.6 の右側のように曲線で分類する．線形 SVM では赤の領域の青の点が，青の領域に赤の点が多く紛れ込んでいる．非線形 SVM ではほぼ正確に赤の点と青の点が存在する領域を分類できている．

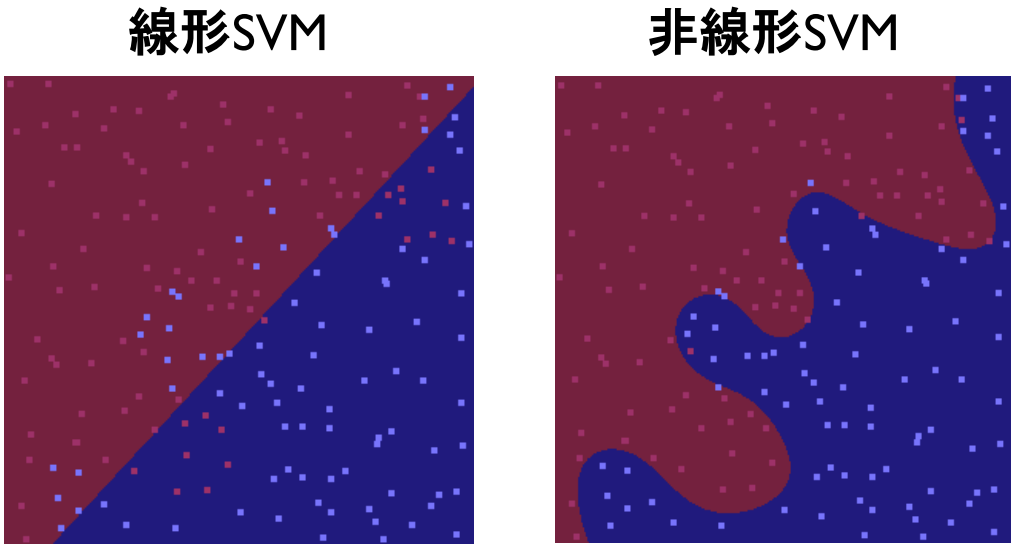

図 4.6: 線形 SVM と非線形 SVM による判別例

本研究ではカーネル関数としてガウシアンカーネルを採用した．また，実験に利用した SVM の実装は LIBSVM [33] であり，R [34] のパッケージ e1071 [35] 経由で利用した．SVM を利用する際にはコストパラメータ C とカーネル関数のパラメータ γ の 2 つのパラメータを調整することにより，汎化能力を高めることができる．パラメータの値によっては過学習が発生し，図 4.7 のように，青色の点の近傍しか青色の領域と学習できず，誤判定の原因となる．したがって，パラメータチューニングが重要である．

本研究では SVM の 2 つのパラメータはグリッドサーチにより最適化したものを用いた．

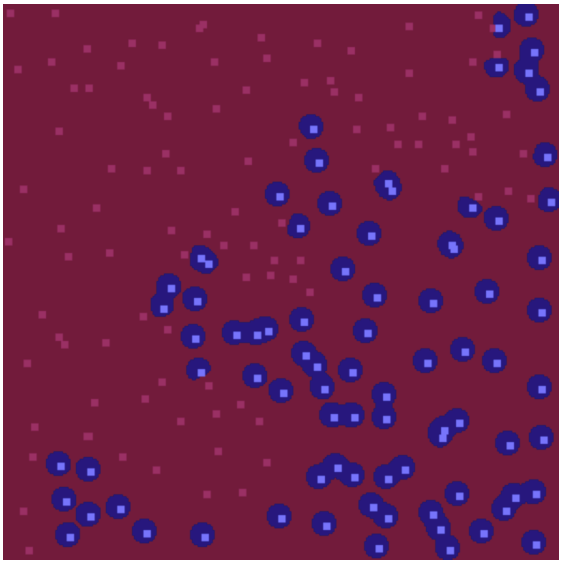

図 4.7: 非線形 SVM の過学習の例

第 5 章

実証実験

5.1 実験に用いるデータ

5.1.1 FFRI Dataset

実験にはマルウェアの動的解析結果のデータセットである FFRI Dataset 2013, 2014, 2015 [31, 32, 17] を用いる．このデータセットは MWS Dataset 2013, 2014, 2015 [15, 16, 17] の一部として研究者コミュニティに対して提供されているものである．動的解析の対象となったマルウェアは株式会社 FFRI [30] が独自に収集した PE (Portable Executable) 形式かつ Windows 上で実行可能なファイルであり，検体数は合計 8,644 個である．マルウェアの動的解析ログはオープンソースのソフトウェアである Cuckoo Sandbox [36] によって取得された．

なお，データセットに含まれるものは解析結果のログファイルのみであり，解析対象となった実際のマルウェア自体は含まれていない．FFRI Dataset で提供される動的解析ログに存在するデータ項目を表 5.1 に示す．

動的解析ログに含まれる項目 virustotal の具体的な内容例を図 5.1 に示す．図のように，virustotal の項目では，VirusTotal に登録されている様々なベンダのアンチウィルスソフトによる検知結果が書かれており，各ベンダが名付けたマルウェアの名前もわかる．

本研究では，マルウェアの命名規則には Kaspersky [38] の検知結果 (図 5.2) を採用した．亜種の推定ができることを示すためには正解となるラベルが必要となるためである．また，Kaspersky はマルウェアの検出精度が高く，その結果が先行研究 [12, 13, 14] において利用されることも多いためである．

Prefix と Variant は存在しないこともあり，それぞれ検体を検知したサブシステムと亜種の名前を表す．Behaviour は検体の挙動，Platform は検体の実行環境 (OS) ，Name は公式の検体の

表 5.1: FFRI Dataset の動的解析ログに存在するデータ項目

項目	内容
info	解析の開始時刻，解析の終了時刻，id など
yara	yara [39] の有する標準ルールセットとの照合結果
signatures	ユーザー定義シグニチャとの照合結果
virustotal	VirusTotal [37] に登録されている各アンチウィルスソフトの検出結果
static	マルウェア検体のファイル情報 (セクション構造, インポート API 等)
dropped	マルウェア検体が実行時に生成したファイルに関する情報
behavior	マルウェア検体実行時の API ログ (PID, TID, API の関数名, 引数, 返り値等)
processtree	マルウェア検体実行時のプロセスツリー (親子関係)
summary	マルウェア検体が実行時にアクセスしたファイル, レジストリキー等の情報
target	解析対象となったマルウェア検体のファイルに関する情報 (ファイルサイズ, ハッシュ値等)
debug	動的解析時の Cuckoo Sandbox のデバッグログ
strings	マルウェア検体中に含まれる文字列情報
network	マルウェア検体が実行時に行った通信に関する情報


```
"virustotal": {
    "scan_id": "398530a7f90c1d37a75f3f7beba297b05d6b457dd0362d477a26f9a2258d4540-1428627020",
    "sha1": "2a0bcd9f4e91b0510e2e06d4044456958fc65b3d",
    "resource": "85d904aaa1a3b050c3dad9b66502c2d7",
    "response_code": 1,
    "scan_date": "2015-04-10 00:50:20",
    "permalink": "https://www.virustotal.com/file/398530a7f90c1d37a75f3f7beba297b05d6b457dd0362d477a26f9a2258d4540/analysis/1428627020/",
    "verbose_msg": "Scan finished, information embedded",
    "sha256": "398530a7f90c1d37a75f3f7beba297b05d6b457dd0362d477a26f9a2258d4540",
    "positives": 44,
    "total": 57,
    "md5": "85d904aaa1a3b050c3dad9b66502c2d7",
    "scans": {
        "Bkav": {
            "detected": true,
            "version": "1.3.0.6379",
            "result": "HW32.Packed.B791",
            "update": "20150409"
        },
                    .
                    .
                    .
        "Kaspersky": {
            "detected": true,
            "version": "15.0.1.10",
            "result": "Backdoor.Win32.Androm.gbpu",
            "update": "20150410"
        },
                    .
                    .
                    .
    }
},
```


図 5.1: 項目 virustotal の内容例

[Prefix:]Behaviour.Platform.Name[.Variant]

図 5.2: Kaspersky の命名規則

ファミリー名である [41]．本論文では Behaviour.Platform.Name をマルウェアファミリー名と定義し，Variant が存在する個々のマルウェアを Behaviour.Platform.Name の亜種と定義する．たとえば，マルウェアファミリー名が Backdoor.Win32.Androm であるマルウェアには Backdoor.Win32.Androm.affh，Backdoor.Win32.Androm.bcnx，Backdoor.Win32.Androm.binu を始めとした 100 種類以上の亜種が存在している．

FFRI Dataset 2013, 2014, 2015 の動的解析ログの中に出現した API 関数名は全部で 160 種類であった．その一覧を図 5.4 に示す．

5.1.2　亜種推定用データセット

FFRI Dataset から新たに亜種推定実験用データセットを作成した．亜種推定実験に利用するマルウェアファミリーは，亜種が存在し，なおかつ 100 個以上の検体が存在する 11 種類である．FFRI Dataset 2013, 2014, 2015 において，100 個以上の検体が存在するマルウェアファミリーは 13 種類である．その一覧を検体数順に表 5.2 に示す．

表 5.2: 100 個以上の検体が存在するマルウェアファミリー

マルウェアファミリー名	検体数
Trojan.Win32.Generic	1,410
Trojan-Spy.Win32.Zbot	361
Hoax.Win32.ArchSMS	307
Worm.Win32.WBNA	288
DangerousObject.Multi.Generic	256
Trojan.Win32.Yakes	235
Trojan.Win32.Jorik	169
Trojan.Win32.Inject	160
Trojan.Win32.Agent	151
Worm.Win32.Vobfus	120
Trojan-Ransom.Win32.Foreign	119
Backdoor.Win32.Androm	118
Trojan-PSW.Win32.Tepfer	100

Trojan.Win32.Generic と DangerousObject.Multi.Generic も 100 以上の検体が存在するが，今回実験用データセットからは除外することにした．それぞれのマルウェア名は正式には HEUR:Trojan.Win32.Generic と UDS:DangerousObject.Multi.Generic であった．Kaspersky の命名規

```
"behavior": {
    "processtree": [
      {
        "parent_id": 2160,
        "pid": 3168,
        "children": [],
        "name": "2A0BCD9F4E91B0510E2E06D4044456958FC65B3D"
      }
    ],
    "processes": [
      {
        "parent_id": 2160,
        "process_name": "2A0BCD9F4E91B0510E2E06D4044456958FC65B3D",
        "process_id": 3168,
        "first_seen": "2015-05-21 12:48:14,445",
        "calls": [
          {
            "category": "system",
            "status": true,
            "return": "0x00000000",
            "timestamp": "2015-05-21 12:48:14,538",
            "thread_id": "3172",
            "repeated": 0,
            "api": "LdrLoadDll",
            "arguments": [
              {
                "name": "Flags",
                "value": "1636536"
              },
              {
                "name": "BaseAddress",
                "value": "0x73ad0000"
              },
              {
                "name": "FileName",
                "value": "C:\\Windows\\system32\\uxtheme.dll"
              }
            ],
            "id": 0
          },
          {
            "category": "misc",
            "status": true,
            "return": "0x00000016",
            "timestamp": "2015-05-21 12:48:14,538",
            "thread_id": "3172",
            "repeated": 0,
            "api": "GetSystemMetrics",
            "arguments": [
              {
                "name": "SystemMetricIndex",
                "value": "31"
              }
            ],
            "id": 1
          },
                                    .
                                    .
                                    .
```


図 5.3: 項目 behavior の内容例

ControlService CopyFileA CopyFileExW CopyFileW CreateDirectoryExW CreateDirectoryW CreateProcessInternalW
CreateRemoteThread CreateServiceA CreateServiceW CreateThread DeleteFileA DeleteFileW DeleteService
DeviceIoControl DnsQuery_A DnsQuery_W ExitProcess ExitThread ExitWindowsEx FindFirstFileExA FindFirstFileExW
FindWindowA indWindowExA FindWindowExW FindWindowW GetAddrInfoW GetCursorPos GetSystemMetrics
HttpOpenRequestA HttpOpenRequestW HttpSendRequestA HttpSendRequestW InternetCloseHandle InternetConnectA
InternetConnectW InternetOpenA InternetOpenUrlA InternetOpenUrlW InternetOpenW InternetReadFile InternetWriteFile
IsDebuggerPresent LdrGetDllHandle LdrGetProcedureAddress LdrLoadDll LookupPrivilegeValueW MoveFileWithProgressW
NtAllocateVirtualMemory NtClose NtCreateFile NtCreateKey NtCreateMutant NtCreateNamedPipeFile NtCreateProcessEx
NtCreateSection NtCreateThreadEx NtCreateUserProcess NtDelayExecution NtDeleteFile NtDeleteKey NtDeleteValueKey
NtDeviceIoControlFile NtEnumerateKey NtEnumerateValueKey NtFreeVirtualMemory NtGetContextThread NtLoadKey
NtOpenDirectoryObject NtOpenFile NtOpenKey NtOpenKeyEx NtOpenMutant NtOpenProcess NtOpenSection
NtOpenThread NtProtectVirtualMemory NtQueryDirectoryFile NtQueryInformationFile NtQueryKey NtQueryValueKey
NtReadFile NtReadVirtualMemory NtResumeThread NtSaveKey NtSetContextThread NtSetInformationFile NtSetValueKey
NtSuspendThread NtTerminateProcess NtTerminateThread NtWriteFile NtWriteVirtualMemory OpenSCManagerA
OpenSCManagerW OpenServiceA OpenServiceW ReadProcessMemory RegCloseKey RegCreateKeyExA RegCreateKeyExW
RegDeleteKeyA RegDeleteKeyW RegDeleteValueA RegDeleteValueW RegEnumKeyExA RegEnumKeyExW RegEnumKeyW
RegEnumValueA RegEnumValueW RegOpenKeyExA RegOpenKeyExW RegQueryInfoKeyA RegQueryInfoKeyW
RegQueryValueExA RegQueryValueExW RegSetValueExA RegSetValueExW RemoveDirectoryA RemoveDirectoryW
RtlCreateUserThread SetWindowsHookExA SetWindowsHookExW ShellExecuteExW StartServiceA StartServiceW
TransmitFile URLDownloadToFileW UnhookWindowsHookEx VirtualFreeEx VirtualProtectEx WSARecv WSARecvFrom
WSASend WSASendTo WSASocketA WSASocketW WSAStartup WriteConsoleA WriteConsoleW WriteProcessMemory
ZwMapViewOfSection __anomaly__ accept bind closesocket connect getaddrinfo gethostbyname ioctlsocket listen
recv recvfrom select send sendto setsockopt shutdown socket system

図 5.4: 動的解析ログに存在した API 関数名の一覧

則における Prefix の HEUR は HEURISTIC（発見的検出）を意味する．さらに，Name が Generic（一般的な）なので，具体的なマルウェアの種類が不明であり，亜種も存在しなかったため，今回の実験には不適切と判断し除外した．Prefix の UDS は Urgent Detection System（緊急検知システム）を意味する．UDS:DangerousObject.Multi.Generic は，ウィルス定義データではまだ反映されていない未知の脅威の総称名であり，こちらも素性がはっきりしないので除外した．なぜなら，本研究の目的は，種別が判明しているマルウェアに対し，Paragraph Vector と SVM を利用することで亜種か否かを推定できることを示すことだからである．

亜種推定の実験用に，先述した 11 種類のマルウェアファミリーから，亜種か推定したいファミリーのマルウェアを 100 検体，それ以外のファミリーのマルウェアを 100 検体 (残り 10 種類のファミリーから 10 個ずつランダムに抽出) ずつ集めたデータセットを 11 通り作成した．なお，亜種推定用データセットのマルウェアの表現形式は，既に第 4.1.1 節で説明した特徴ベクトルの形式に変換済みである．また，検体数を 100 個ずつにした理由は，統計的な偏りをなくすためである．亜種推定用データセットの例として，該当マルウェアがマルウェアファミリー Backdoor.Win32.Androm の亜種であるか否かを推定する時に利用するデータセットを図 5.5 に示す．

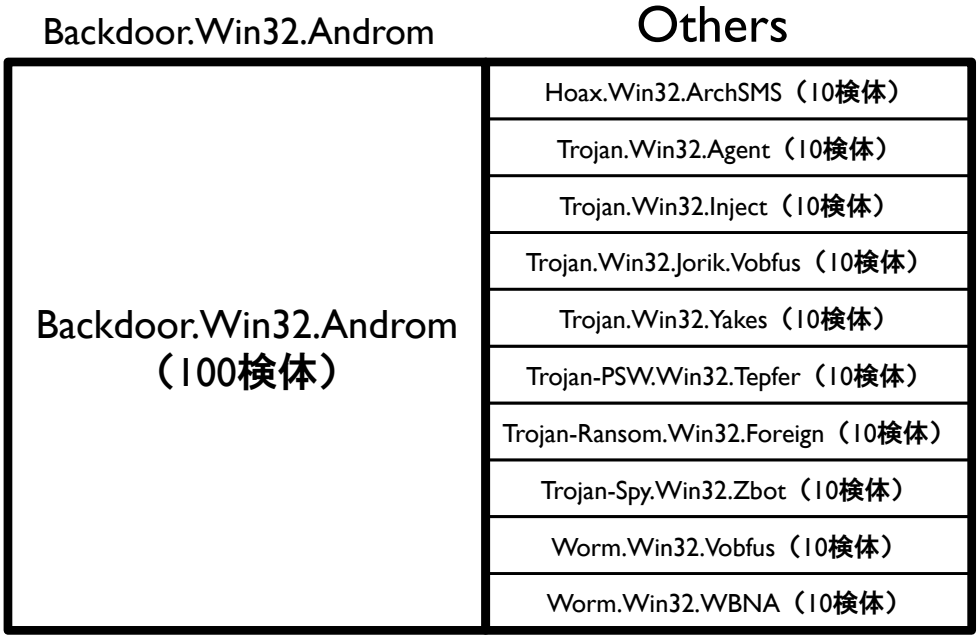

図 5.5: 亜種推定用データセットの例

5.2　実験手順

作成した亜種推定用データセットに対して，第 4 章で述べた提案手法の STEP2，STEP3 を適用し，実証実験を行う．SVM による亜種推定の評価実験には K-分割交差検定を利用する．概要を図 5.6 に示す．

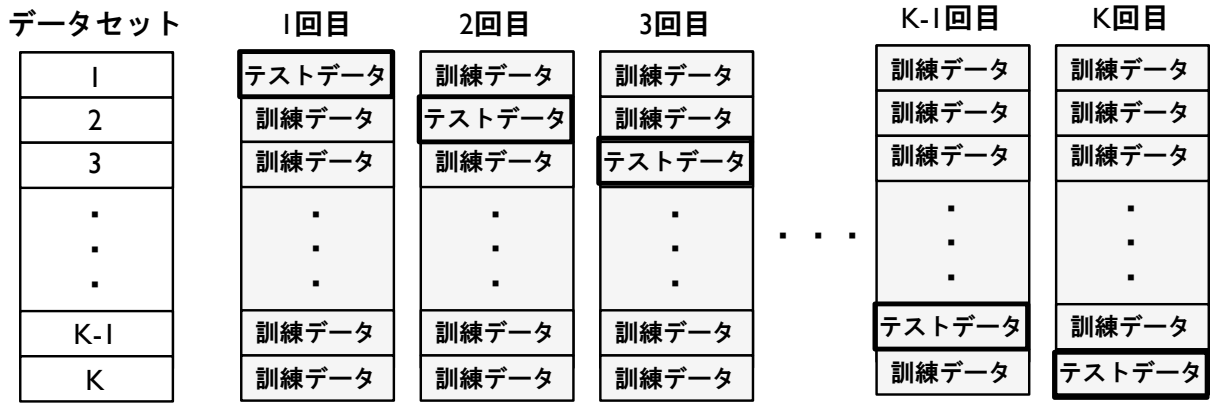

図 5.6: K-分割交差検定の概要

K-分割交差検定はすべてのデータを訓練とテストに利用し，モデルの汎用的な性能評価を可能とする手法である [28]．まず，データセット全体をデータの偏りがないよう K 個のブロック

に等分する．そして，１つのブロックを除いたデータ全体を訓練データとして利用し，取り除いた 1 ブロックをテストデータとして利用することで学習済みモデルの性能を評価する．そして，同様の操作を K 個のブロックすべてに関して行う．最終的な性能評価値は，すべての実験結果の平均値を取ることで求められる．今回は図 5.6 において K=10 に相当する 10 分割交差検定によって性能評価を行った．

なお，API コールログ列の Paragraph Vector 化には sentence2vec [43] を利用し，特徴ベクトルの次元は 100 から 1000 まで 100 刻みで変化させた．

5.3　評価指標

本研究で利用する評価指標について説明する．まず，評価値を算出するために必要な推定結果と真の結果の関係を表 5.3 に示す．表の中身はそれぞれの結果に当てはまるデータの個数を意味する．たとえば，True Positive (TP) は亜種マルウェアを正しく亜種マルウェアと推定できた数であり，False Positive (FP) は亜種マルウェアでないその他のマルウェアを誤って亜種マルウェアと推定してしまった数である．また，False Negative (FN) は亜種マルウェアを誤って亜種マルウェアでないその他のマルウェアと推定してしまった数であり，True Negative (TN) は亜種マルウェアでないその他のマルウェアを正しくその他のマルウェアであると推定できた数である．

表 5.3: 推定結果と真の結果の関係

		真の結果	
		亜種マルウェア	その他のマルウェア
推定結果	亜種マルウェア	True Positive (TP)	False Positive (FP)
	その他のマルウェア	False Negative (FN)	True Negative (TN)

評価に利用する指標は精度 (Accuracy)，適合率 (Precision) ，再現率 (Recall) ，F 値 (F-measure) の計４種類である．

精度 (Accuracy) は，推定したすべてのマルウェアの内，正しく亜種マルウェアであるか否かを推定できたものの割合 であり，表 5.3 内の記号を利用すると下記の式 5.1 のように書ける．

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{5.1}$$

適合率 (Precision) は，亜種マルウェアであると推定したマルウェアの内，正しく亜種マルウェアであると推定できたものの割合であり，下記の式 5.2 で表される．

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

再現率 (Recall) は，実際に亜種マルウェアであるマルウェアの内，正しく亜種マルウェアであると推定できたものの割合であり，下記の式 5.3 で表される．

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

F 値 (F-measure) は，適合率と再現率の調和平均により算出される評価尺度であり，下記の式 5.4 で表される．

$$F\text{-}measure = \frac{2Recall \cdot Precision}{Recall + Precision} \tag{5.4}$$

たとえば，あるマルウェアファミリーに関して，亜種推定実験の結果が表 5.4 の通りであったとする．

表 5.4: 推定結果と真の結果の関係（具体例）

		真の結果	
		亜種マルウェア	その他のマルウェア
推定結果	亜種マルウェア	9	4
	その他のマルウェア	1	6

すると，精度 (Accuracy)，適合率 (Precision) ，再現率 (Recall) ，F 値 (F-measure) はそれぞれ下記の通り求めることができる．

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{9 + 6}{9 + 4 + 1 + 6} = 0.750 \tag{5.5}$$

$$Precision = \frac{TP}{TP + FP} = \frac{9}{9 + 4} = 0.692 \tag{5.6}$$

$$Recall = \frac{TP}{TP + FN} = \frac{9}{9 + 1} = 0.900 \tag{5.7}$$

$$F\text{-}measure = \frac{2Recall \cdot Precision}{Recall + Precision} = \frac{2 \cdot 0.900 \cdot 0.6923}{0.900 + 0.6923} = 0.783 \tag{5.8}$$

5.4　実験結果

実験結果を表 5.5，表 5.6，表 5.7，表 5.8 に示す．

表 5.5: 次元ごとの亜種マルウェア推定の精度 (%)

マルウェアファミリー名＼次元	100	200	300	400	500	600	700	800	900	1000
Backdoor.Win32.Androm	82.0	82.5	81.5	80.5	77.5	81.5	79.0	79.0	79.5	81.5
Hoax.Win32.ArchSMS	94.5	97.5	93.5	97.5	96.5	98.0	95.5	95.0	96.0	95.0
Trojan.Win32.Agent	77.5	78.0	75.0	74.0	77.0	77.0	75.5	75.5	72.5	74.0
Trojan.Win32.Inject	75.5	71.5	67.5	75.0	69.0	72.5	70.5	77.0	75.5	76.0
Trojan.Win32.Jorik.Vobfus	88.5	90.5	86.5	88.0	87.5	89.0	90.5	90.5	90.0	90.5
Trojan.Win32.Yakes	84.0	84.5	82.0	80.0	83.0	82.0	84.0	81.5	84.5	86.0
Trojan-PSW.Win32.Tepfer	82.5	83.5	83.0	84.0	83.5	82.0	85.5	85.5	87.0	84.5
Trojan-Ransom.Win32.Foreign	76.5	82.0	81.5	82.5	86.0	82.0	80.5	82.0	80.5	86.5
Trojan-Spy.Win32.Zbot	80.5	83.5	81.5	82.0	78.5	79.5	81.0	80.5	80.0	74.0
Worm.Win32.Vobfus	88.0	89.0	89.5	88.5	90.5	90.0	87.5	89.0	88.5	89.0
Worm.Win32.WBNA	88.5	92.5	91.5	93.0	93.0	92.0	90.0	94.5	91.5	91.0

表 5.6: 次元ごとの亜種マルウェア推定の適合率 (%)

マルウェアファミリー名＼次元	100	200	300	400	500	600	700	800	900	1000
Backdoor.Win32.Androm	81.1	76.7	77.0	74.1	73.2	84.3	76.2	74.7	72.8	76.0
Hoax.Win32.ArchSMS	95.6	99.0	97.8	98.2	98.1	98.2	98.2	96.4	98.0	97.4
Trojan.Win32.Agent	78.8	84.7	73.4	78.9	83.3	87.0	72.2	76.8	70.8	75.7
Trojan.Win32.Inject	77.7	72.9	65.1	76.5	66.6	75.3	72.0	75.0	70.0	74.0
Trojan.Win32.Jorik.Vobfus	89.6	92.7	90.0	87.5	84.4	85.5	86.5	88.4	91.7	92.0
Trojan.Win32.Yakes	88.5	86.0	79.2	83.7	77.1	88.2	91.9	82.7	87.3	87.3
Trojan-PSW.Win32.Tepfer	98.8	98.8	97.1	90.2	86.6	97.5	88.8	95.9	93.6	91.6
Trojan-Ransom.Win32.Foreign	69.9	84.4	78.0	75.7	82.9	81.0	75.4	77.9	75.8	85.4
Trojan-Spy.Win32.Zbot	76.7	79.7	80.5	80.1	74.8	77.1	77.3	74.1	79.6	76.7
Worm.Win32.Vobfus	83.1	89.6	86.8	88.3	89.5	86.2	85.6	88.1	88.1	84.4
Worm.Win32.WBNA	89.3	90.8	89.3	92.4	95.9	91.0	88.4	94.6	94.3	91.6

表 5.5 から，全体として平均精度 84.0%で亜種マルウェアの推定が可能であるとわかる．特に Hoax.Win32.ArchSMS の推定精度はどの次元数においても 90%以上と高精度の推定ができることがわかる．また，次元数を変えることで，同一のマルウェアファミリーに関して，推定精度が最大で 10%程度変化することもわかる．しかし，次元数を増やせば推定精度が上がるわけではなく，マルウェアファミリーごとに推定しやすい次元があるとわかった．

表 5.6 から，全体的に適合率も精度と同様の傾向を示し，平均適合率は 84.4%と高い結果となった．したがって，本提案手法において，未知のマルウェアを亜種マルウェアと推定した場合，そのマルウェアが本当にその亜種マルウェアである確実性は高いといえる．

表 5.7: 次元ごとの亜種マルウェア推定の再現率 (%)

マルウェアファミリー名＼次元	100	200	300	400	500	600	700	800	900	1000
Backdoor.Win32.Androm	86.0	95.0	93.0	96.0	93.0	81.0	85.0	88.0	95.0	97.0
Hoax.Win32.ArchSMS	94.0	96.0	89.0	97.0	95.0	98.0	93.0	94.0	94.0	93.0
Trojan.Win32.Agent	79.0	69.0	80.0	68.0	68.0	64.0	83.0	77.0	80.0	72.0
Trojan.Win32.Inject	70.0	69.0	82.0	73.0	78.0	67.0	69.0	82.0	90.0	85.0
Trojan.Win32.Jorik.Vobfus	88.0	89.0	83.0	91.0	95.0	96.0	98.0	95.0	90.0	90.0
Trojan.Win32.Yakes	79.0	84.0	90.0	76.0	96.0	78.0	75.0	80.0	81.0	86.0
Trojan-PSW.Win32.Tepfer	66.0	68.0	68.0	77.0	81.0	66.0	83.0	75.0	80.0	77.0
Trojan-Ransom.Win32.Foreign	97.0	80.0	90.0	97.0	93.0	85.0	93.0	90.0	93.0	89.0
Trojan-Spy.Win32.Zbot	90.0	92.0	84.0	88.0	88.0	87.0	90.0	95.0	82.0	71.0
Worm.Win32.Vobfus	97.0	89.0	95.0	90.0	93.0	97.0	92.0	91.0	90.0	98.0
Worm.Win32.WBNA	88.0	95.0	95.0	95.0	90.0	94.0	93.0	95.0	89.0	91.0

表 5.8: 次元ごとの亜種マルウェア推定の F 値

マルウェアファミリー名＼次元	100	200	300	400	500	600	700	800	900	1000
Backdoor.Win32.Androm	0.828	0.846	0.837	0.834	0.811	0.815	0.799	0.805	0.823	0.846
Hoax.Win32.ArchSMS	0.944	0.974	0.930	0.974	0.964	0.980	0.953	0.950	0.958	0.948
Trojan.Win32.Agent	0.774	0.753	0.761	0.716	0.736	0.728	0.770	0.762	0.742	0.724
Trojan.Win32.Inject	0.720	0.700	0.721	0.736	0.715	0.699	0.692	0.772	0.785	0.784
Trojan.Win32.Jorik.Vobfus	0.883	0.903	0.860	0.888	0.888	0.902	0.916	0.911	0.901	0.904
Trojan.Win32.Yakes	0.831	0.847	0.835	0.789	0.852	0.813	0.820	0.808	0.838	0.857
Trojan-PSW.Win32.Tepfer	0.780	0.796	0.791	0.825	0.828	0.782	0.853	0.827	0.859	0.830
Trojan-Ransom.Win32.Foreign	0.809	0.815	0.826	0.849	0.872	0.826	0.829	0.833	0.829	0.869
Trojan-Spy.Win32.Zbot	0.824	0.849	0.818	0.831	0.805	0.811	0.828	0.831	0.799	0.731
Worm.Win32.Vobfus	0.893	0.889	0.901	0.886	0.908	0.910	0.880	0.891	0.887	0.903
Worm.Win32.WBNA	0.885	0.926	0.917	0.932	0.926	0.920	0.903	0.945	0.909	0.909

表 5.7 から，再現率に関しても全体的に高く，平均再現率は 86.1%であった．したがって，既知のマルウェアファミリーに所属する亜種マルウェアを見逃す割合は少ないことがわかる．しかし，Trojan.Win32.Agent と Trojan.Win32.Inject は Paragraph Vector の次元によっては 60%台まで下がるときもある．

適合率と再現率は一般にトレードオフの関係にある．そのため，適合率と再現率の総合評価である F 値が高ければ高いほどバランスの取れた性能の高さと言える．表 5.8 から全体的な平均 F 値は 0.842 であるとわかる．また，Trojan.Win32.Inject に関しては 0.70 に達しない時があるものの，その他に関しては精度と同様に高い値を示している．

表 5.5，表 5.6，表 5.7，表 5.8 の各実験結果の平均値を，マルウェアファミリーごとに計算した．その結果を図 5.7 に示す．図 5.7 から本提案手法が全体として平均的に高い性能を示すことが一目でわかる．また，Trojan.Win32.Agent と Trojan.Win32.Inject の推定に関しては，他のマルウェアの推定と比べて 10%程度性能が低いこともわかる．

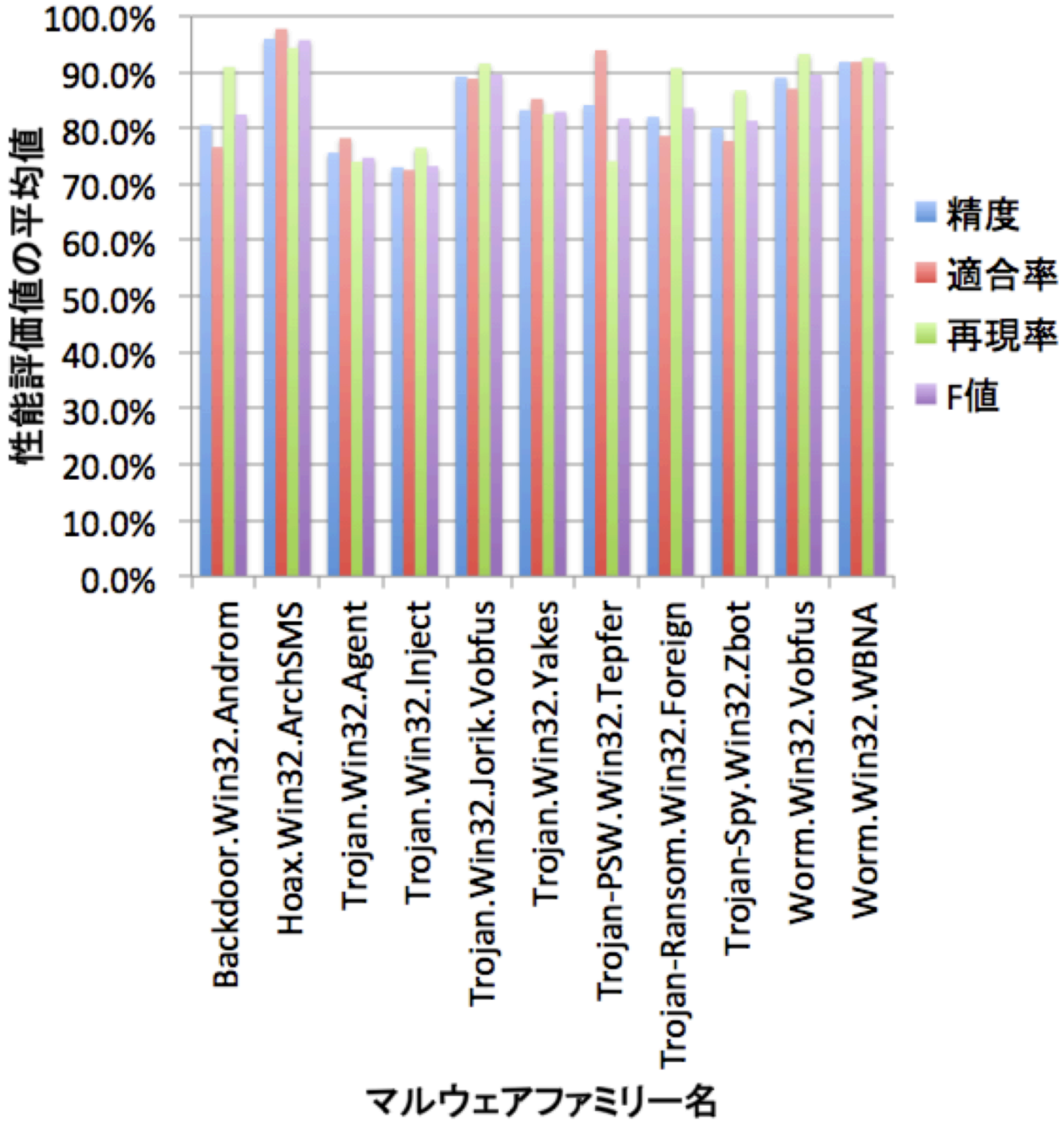

図 5.7: 実験結果の平均値

5.5　考察

表 5.5 と表 5.8 の結果から，全体的な性能は良好であると考えられる．特に，Hoax や Worm に関しては精度，F 値ともに 9 割近い値を達成しているため，Paragraph Vector はマルウェアの挙動を特徴ベクトル化することに成功していると判断できる．

しかし，Trojan.Win32.Inject には最低推定精度に 60%台が存在し，Trojan.Win32.Agent は推定精度がすべて 70%台と全体的に推定精度が低かった．F 値に関しても，同様の傾向で低い結果があった．この理由について，実際の判定結果をもとに考察する．どちらも，他の Trojan 系統のマルウェアを亜種であると誤判定していることが多かった．今回実験に利用した亜種推定データセットの中身としては，Trojan 系統のマルウェアの割合が多かった．そのため，Trojan 系統のマルウェアの亜種を推定する際，他の Trojan 系統のマルウェアとの違いが特徴に出にくく，その結果として他のマルウェアと比べ，低い推定精度になったのではないかと考えられる．

亜種推定用データセット内で Hoax に分類されるマルウェアファミリーは Hoax.Win32.ArchSMS だけであり，この亜種マルウェアの推定精度，適合率，再現率，F 値は非常に高かった．その一方で，Backdoor に分類されるマルウェアファミリーも Backdoor.Win32.Androm だけであり，再現率に関しては 6 通りの次元においてが 90%を越える値を出したが，精度と適合率の結果は 80%前後に留まった．学習済み SVM モデルが Backdoor.Win32.Androm と誤って判定したマルウェアの多くが Trojan 系統のマルウェアであった．一般に，Trojan は正規のプログラムを装い，ユーザーの手によってインストールされることで感染し，ユーザの陰で様々な不正な挙動を示す [7]．Backdoor はユーザに知られることなく感染 PC を外部から遠隔操作することを可能にするマルウェアであり，その類似性が誤判定の原因として考えられる．

第 6 章

結論

6.1 まとめ

本研究では，マルウェアの挙動を表した API コール列を一連の文章と見なし，自然言語処理分野の Deep Learning の手法である Paragraph Vector を応用して特徴ベクトルを作成した．その特徴ベクトルを SVM に学習させることで，亜種マルウェアを推定する手法を提案した．

実験の結果，亜種マルウェアを推定することが可能であることを示した．本手法はマルウェアの挙動に着目した手法であるため，静的解析に比べると短時間でマルウェアの解析を行うことができ，さらに難読化されたマルウェアにも対応可能という点で優れている．また，特徴ベクトルを自動で生成することができるため，非常に効率の良いマルウェア解析を行うことに応用可能である．したがって，本研究はマルウェア解析を行う研究者，マルウェアへの対策を早急に打つ必要があるセキュリティベンダに対して貢献できる．

6.2 今後の課題

6.2.1 推定精度の向上

Hoax.Win32.ArchSMS に関しては最大 98.0%という非常に高い精度で亜種の推定が可能であった．一方で，Trojan.Win32.Agent，Trojan.Win32.Inject に関してはどの次元でも亜種マルウェアの推定精度が 80%未満と低い結果となった．本研究では API 関数名のみを利用して API コール列を作成した．動的解析ログからはその他にも API 関数の戻り値や引数などの情報も存在する．API 関数名以外の情報の活用，および静的解析との組み合わせによって，推定精度がさらに向上する可能性がある．

6.2.2　新種のマルウェアファミリーへの対応

本手法では，事前に SVM に対してあるマルウェアファミリーに属するマルウェアの特徴を学習させ，その学習結果を基に未知のマルウェアがあるファミリーに属するか否か（亜種マルウェアであるか否か）を推定する．そのため，既知のマルウェアファミリーに属する亜種マルウェアは推定可能だが，未知のマルウェアファミリーに属する亜種マルウェアの推定をすることはできない．完全に新種のマルウェアファミリーに対応するためには，Paragraph Vector を応用して作成したマルウェアの特徴ベクトルを教師無し学習によって分類する手法が考えられる．

6.2.3　特徴ベクトル自体の考察

本手法では，自然言語処理分野の Deep Learning である Paragraph Vector を応用した．自然言語処理の分野では Word Vector が単語の意味を表すこと，Paragraph Vector が文書の意味を表すことを確認するため，その単語ベクトル同士の加算や減算をしている．本研究の結果，Paragraph Vector によって特徴ベクトル化されたマルウェアは，亜種などの似た動作をする場合は類似した特徴を持つことはわかった．本手法の有効性や応用性を確かめるため，得られた特徴ベクトル同士の加算や減算をすることによって，そのマルウェアの意味がどのようにベクトル空間上に射影されたのかを考察する必要がある．

謝辞

本修士論文の作成にあたり，日ごろよりご指導を頂いた早稲田大学基幹理工学研究科の後藤滋樹教授に深く感謝いたします．本研究を進めるにあたり，貴重なアドバイスをしてくださった後藤研究室の皆様に感謝いたします．特に，青木一樹氏にはFFRI Datasetの取り扱いを始め，マルウェア解析に関する様々なアドバイスを頂き，大変お世話になりました．また，仲山裕也氏には亜種マルウェアの分類に関して相談に乗って頂きました．両氏に深く感謝いたします．

本研究のために貴重なデータセットを提供して頂いた株式会社FFRIの諸氏に心より感謝いたします．本研究はFFRI Dataset無しに行うことはできませんでした．

最後に，研究室内で共に励まし合った同期の皆様に感謝いたします．

参考文献

[1] McAfee Labs, “McAfee 脅威レポート：2015 年第 1 四半期,” McAfee, http://www.mcafee.com/jp/resources/reports/rp-quarterly-threat-q1-2015.pdf , June 2015.

[2] G DATA SECURITYLABS, “G DATA SECURITYLABS MALWARE REPORT,” G DATA Software AG, https://public.gdatasoftware.com/Presse/Publikationen/Malware_Reports/GData_PCMWR_H2_2014_EN_v1.pdf , May 2015.

[3] 警察庁, “平成 27 年上半期のインターネットバンキングに係る不正送金事犯の発生状況等について,” https://www.npa.go.jp/cyber/pdf/H270903_banking.pdf, September 2015.

[4] 読売新聞（YOMIURI ONLINE）, “年金機構に続き１０組織でウイルス感染・流出,” http://www.yomiuri.co.jp/science/goshinjyutsu/20150702-OYT8T50024.html, June 2015.

[5] I. Guyon, and A. Elisseeff, “An Introduction to Variable and Feature Selection,” The Journal of Machine Learning Research Volume 3, pp.1157–1182, May 2003.

[6] 津田 宏治, “サポートベクターマシンとは何か,” 電子情報通信学会誌, Vol. 83, No. 6, pp. 460–466, June 2000.

[7] 井上 大介, 中尾 康二, “マルウェアって? (特集マルウェア),” 情報処理, Vol. 51, No. 3, pp.237–243, March 2010.

[8] G. Liang, J. Pang, and C. Dai, “A Behavior-Based Malware Variant Classification Technique,” International Journal of Information and Education Technology, Vol. 6, No. 4, pp.291–295, April 2016.

[9] 堀合 啓一, 今泉 隆文, 田中 英彦, “ハミング距離によるマルウェア亜種の自動分類,” 情報処理学会研究報告コンピュータセキュリティ（CSEC） 2008(45(2008-CSEC-041)), pp.61–66, May, 2008.

[10] 堀合 啓一, 今泉 隆文, 田中 英彦, “マルウェア亜種の動的挙動を利用した自動分類手法の提案と実装,” 情報処理学会論文誌 50(4), pp.1321–1333, April 2009.

[11] J.Z. Kolter, M.A. Maloof, “Learning to Detect and Classify Malicious Executables in the Wild,” Journal of Machine Learning Research 7 (2006), pp.2721–2744, December 2006.

[12] 中村 燎太, 松宮 遼, 高橋 一志, 大山 恵弘, “Kullback-Leibler 情報量を用いた亜種マルウェアの同定,” コンピュータセキュリティシンポジウム 2013 論文誌, pp.877–884, October 2013.

[13] 青木 一樹， 後藤 滋樹， “マルウェア検知のための API コールパターンの分析”， 電子情報通信学会総合大会講演論文集 2014 年，pp.179，March 2014.

[14] 藤野 朗稚, 森 達也, “自動化されたマルウェア動的解析システムで収集した大量の API コールログの分析,” コンピュータセキュリティシンポジウム 2013 論文誌, pp.618–625, October 2013.

[15] 神薗 雅紀, 畑田 充弘, 寺田 真敏, 秋山 満昭, 笠間 貴弘, 村上 純一, “マルウェア対策のための研究用データセット～MWS Datasets 2013～,” 情報処理学会 コンピュータセキュリティシンポジウム 2013, vol.2013, no.4, pp.1–8, October 2013.

[16] 秋山 満昭, 神薗 雅紀, 松木 隆宏, 畑田 充弘, “マルウェア対策のための研究用データセット～MWS Datasets 2014～,” 情報処理学会研究報告 Vol.2014-CSEC-66, No.19, pp.1–7, June 2014.

[17] 神薗 雅紀, 秋山 満昭, 笠間 貴弘, 村上 純一, 畑田 充弘, 寺田 真敏, “マルウェア対策のための研究データセット～MWS Datasets 2015～,” 情報処理学会研究報告 Vol.2015-CSEC-70, No.6, pp.1–8, June 2015.

[18] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space”, Proceedings of Workshop at ICLR, pp.1–12, May 2013.

[19] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality, ” Neural Information Processing Systems (NIPS) 2013, pp.1–9, December 2013.

[20] A.M. Dai, C. Olah, Q.V. Le, and G. Corrado, “Document Embedding with Paragraph Vectors,” NIPS Deep Learning Workshop (#68), pp.1–9, December 2014.

[21] T. Mikolov, W. Yih, and G. Zweig, “Linguistic Regularities in Continuous Space Word Representations,” Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013), pp.746–751, May 2013.

[22] Q.V. Le, T. Mikolov, “Distributed Representations of Sentences and Documents”, Proceedings of the 31st International Conference on Machine Learning, pp.1188–1196, June 2014.

[23] 新井 悠, 岩村 誠, 川古谷 裕平, 青木 一史, 澤 裕二, “アナライジング・マルウェア―フリーツールを使った感染事案対処,” オライリージャパン, December 2010.

[24] 西尾 泰和, “word2vec による自然言語処理,” オライリー・ジャパン, May 2014.

[25] 八木 毅, 村山 純一, 秋山 満昭, “コンピュータネットワークセキュリティ,” コロナ社, March 2015.

[26] 松尾 豊, “人工知能は人間を超えるか ディープラーニングの先にあるもの,” KADOKAWA/中経出版, March 2015.

[27] 麻生 英樹，安田 宗樹，前田 新一，岡野原 大輔，岡谷 貴之，久保 陽太郎，ボレガラ ダヌシカ, “深層学習,” 近代科学社, November 2015.

[28] W. Richert, L.P. Coelho, 斎藤 康毅, “実践 機械学習システム,” オライリー・ジャパン, October 2014.

[29] “マルウェア対策研究人材育成ワークショップ 2015 (MWS2015),” `http://www.iwsec.org/mws/2015/`.

[30] “株式会社 FFRI,” `http://www.ffri.jp/`.

[31] 株式会社 フォティーンフォティ技術研究所, “FFRI Dataset 2013 のご紹介,”
http://www.iwsec.org/mws/2013/files/FFRI_Dataset_2013.pdf.

[32] 株式会社 FFRI, “FFRI Dataset 2014 のご紹介,”
http://www.iwsec.org/mws/2014/files/FFRI_Dataset_2014.pdf.

[33] Chih-Chung Chang, and Chih-Jen Lin, “LIBSVM – A Library for Support Vector Machines,”
http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[34] “The R Project for Statistical Computing,”
https://www.r-project.org/.

[35] “CRAN - Package e1071,”
https://cran.r-project.org/web/packages/e1071/index.html.

[36] “Cuckoo Sandbox,”
http://www.cuckoosandbox.org.

[37] “VirusTotal,”
https://www.virustotal.com.

[38] “Kaspersky,”
http://www.kaspersky.com.

[39] “YARA - The pattern matching swiss knife for malware researchers,”
http://plusvic.github.io/yara/.

[40] “Wikipedia,”
https://www.wikipedia.org/.

[41] SECURELIST, “Rules for naming detected objects,”
https://securelist.com/threats/rules-for-naming/.

[42] “word2vec,”
https://code.google.com/p/word2vec/.

[43] “sentence2vec,”
https://github.com/klb3713/sentence2vec.

[44] “gensim: Topic modelling for humans,”
https://radimrehurek.com/gensim/.