2016年度 修士論文

Word Vectorを用いた亜種マルウェア判別法

提出日：2017年1月30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：5115F033-7

佐藤 拓未

目次

図一覧

表一覧

第 1 章

序論

1.1 研究の背景

マルウェアの発見数は増加の傾向にあることが報告されている．大手セキュリティベンダの McAfee のレポート [1] によれば，新しく発見されたマルウェアの種類は 2016 年の第一四半期だけでも約 4000 万種類にものぼる．

マルウェアの発見数の増加に大きく寄与する原因の 1 つとして，**亜種**の存在がある．ある既知のマルウェアに対して，大筋の動作・機能は変更せず，一部に変更を加えたものを亜種と呼ぶ．亜種はツールを用いることで容易に作成でき，このことが新しく発見されるマルウェアの種類の増加に繋がる．

2015 年の G DATA SECURITY のレポート [2] によるとマルウェアの亜種は 1 分間に 12 種類という速度で新しく生まれているという．このように，マルウェアの亜種が大量に生まれている現在の状況において，あるマルウェアがいったいどのマルウェアの亜種であるかを判別することができれば，迅速にマルウェアに応じた適切な対応を行うことができる．

1.2 研究の目的

1.1 節で述べたように，増加するマルウェアに対して，マルウェアの亜種の判別を迅速に行うことが望まれる．3 章でも述べるが，既存の機械学習を用いる手法での特徴量の選定には人の手が介在するものが多い．増加するマルウェアに対応するためには，人の手を必要としない亜種判別のための特徴量の選定方法が必要である．

本論文は人の手が介在しないマルウェアの特徴の選定方法を提案する．マルウェアの特徴を

探るための解析方法として表層解析，動的解析，静的解析が挙げられる [3]．表層解析はファイル自体が悪性であるかの情報収集や，ファイルのメタ情報の収集のために行う解析である．動的解析はマルウェアをサンドボックス環境で実行し，端末上の痕跡や実行した通信といった実際の挙動についての情報を得るための解析である．静的解析は，逆アセンブラやデバッガを用いてマルウェアのプログラムコードを解析し，マルウェアの機能や特徴的なバイト列といった情報を詳細に得るための解析である．

これらのうち，本論文では動的解析で得られる情報に着目する．具体的には，マルウェアの動的解析の結果のうちの API 呼び出し (API コール) に注目し，Deep Learning による自然言語の解析手法の一つである Word Vector を用いて，マルウェアの API 呼び出しの特徴を自動的に計算する．得られた特徴をマルウェアの亜種判別に用いることで，Word Vector を用いたマルウェアの挙動の特徴表現がマルウェアの亜種判別に有用であることを示す．

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 Word Vector

本論文で用いる Word Vector と BoW を解説する．

第 3 章 関連研究

本論文に関連する研究について述べる．

第 4 章 提案手法

本論文の提案手法を説明する．

第 5 章 評価実験

提案手法の有効性を示すためにおこなった実験とその結果および考察を示す．

第 6 章 結論

本論文の結論を述べるとともに，残された課題を示す．

第 2 章

Word Vector

本章では，本論文で利用する Word Vector について説明する．

2.1 1-of-K 符号化

単語や文章をコンピュータ上で扱うための数値化の方法として 1-of-K 符号化がある．K 個の単語が存在するとき，それぞれを K 次元のベクトルで表現する，という方法である．例えば K = 4，つまり A, B, C, D という 4 個の単語をベクトルで表現する場合，A = (1, 0, 0, 0)，B = (0, 1, 0, 0)，C = (0, 0, 1, 0)，D = (0, 0, 0, 1) というようにそれぞれを表現する．この表現を用いると各ベクトル間の距離は等しくなるため，各単語の表現の間に偏りがなくなるという利点がある．

2.2 Bag-of-Words (BoW)

2.2.1 Bag-of-Words の概要

Bag-of-Words とは，文章における単語の出現を要素とするベクトル表現であり [17]，文章に含まれる単語の 1-of-K ベクトルを加算したベクトルで表現される．1-of-K ベクトルは特定の 1 つの要素以外はすべて 0 であるベクトルであるため，Bag-of-Words 表現は文章における各単語の出現回数を要素としたものに等しい．

例えば A, B, C, D の 4 語のみが存在する文章群において，単語 B は (0, 1, 0, 0) と表せる．“A, B, C, B” という文章は (1, 2, 1, 0) と表せる．この表現は各単語が含まれているかどうか

のみを表し，出現の順序を考慮しない．また，N-gram の出現回数に対して Bag-of-Words を適用したものを Bag-of-N-gram と呼ぶ．この場合にはベクトルの各要素が N-gram の出現回数となる．

2.2.2　Bit Bag-of-Words

Bag-of-Words の表現は 1-of-K ベクトルの和によって求められることが多い．前節で挙げた例では，単語 B は文章中に 2 回出現し，Bag-of-Words 表現で B に相当する要素は 2 になった．

文章を表現する上で，単語の出現の重複を考慮しない表現方法も考えられる．つまり，単語の出現回数によらず，“単語が出現したか否か” のみでその単語要素を決定する方法である．前節の例に当てはめると “A, B, C, B” という文章は (1, 1, 1, 0) と表せる．本論文では便宜上この Bag-of-Words の表現を，前節で述べた一般的な Bag-of-Words と区別するため，出現するか否かの 0 or 1 で要素を決定することから Bit Bag-of-Words (Bit BoW) と呼ぶ．

2.3　Word Vector

Word Vector は Mikolov ら [4] 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，ニューラルネットワークに基づいている．複数の文章からなるコーパスを入力として与え，文章中に現れる単語 (Word) の特徴を文脈，つまり前後に出現する単語から Deep Learning によってベクトル化する手法である．そのため，異なる単語であっても前後に出現する単語が同じ，あるいは類似していれば Word Vector も類似するという特長がある．Word Vector のアルゴリズムにおいて重要な CBOW と Skip-gram モデルについて述べる．

2.3.1　CBOW モデル

Word Vector における CBOW モデルおよび後述する Skip-gram モデルは [4] で Word Vector を計算する際に用いられているモデルである．まず，CBOW (Continuous Bag-of-Words) モデルを説明する．CBOW モデルのニューラルネットワークの各層を図 2.1 に示す．CBOW モデルは前後の文脈から出現する単語を予測するモデルである．ある文脈で t 番目に出現する語 w_t について，その前後の $2k$ 語 $(w_{t-k}, ..., w_{t-1}, w_{t+1}, ..., w_{t+k})$ の文脈の Bag-of-Words をニューラルネットの入力として，w_t が出力となるように Word Vector を Deep Learning で学習していく．図に示した射影層は，ニューラルネットワークにおける中間層であり，入力層から中間

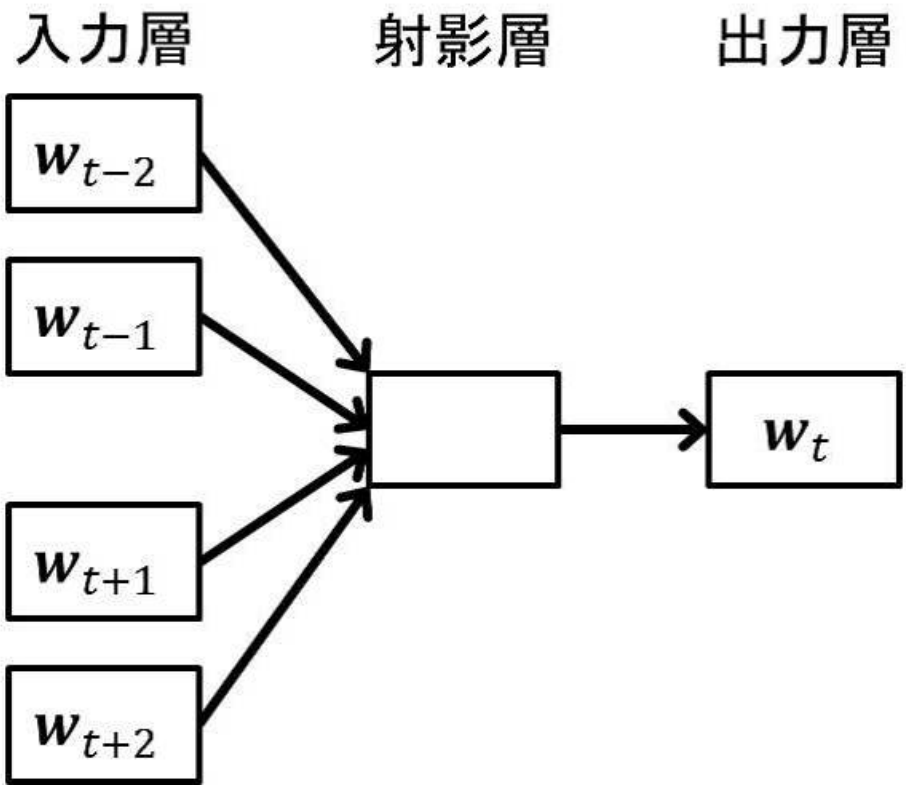

図 2.1: CBOW モデル

層，中間層から出力層への伝達の際の重みが学習によって最適な値に変化する．Word Vector の学習では，どの語に対しても同じ中間層を用いる．

2.3.2　Skip-gram モデル

Skip-gram モデルのニューラルネットワークの各層を図 2.2 に示す．Skip-gram モデルは CBOW モデルと逆向きの予測をするモデルである．つまり，ある文脈で t 番目に出現する語 w_t について，w_t を入力として，その前後の語である $w_{t-k}, ..., w_{t-1}, w_{t+1}, ..., w_{t+k}$ を出力するように Deep Learning で Word Vector を学習していく．CBOW モデルと Skip-gram モデルの比較が Mikolov [4] に述べられている．Skip-gram モデルの方が語の予測精度が高いという結果が紹介されている．

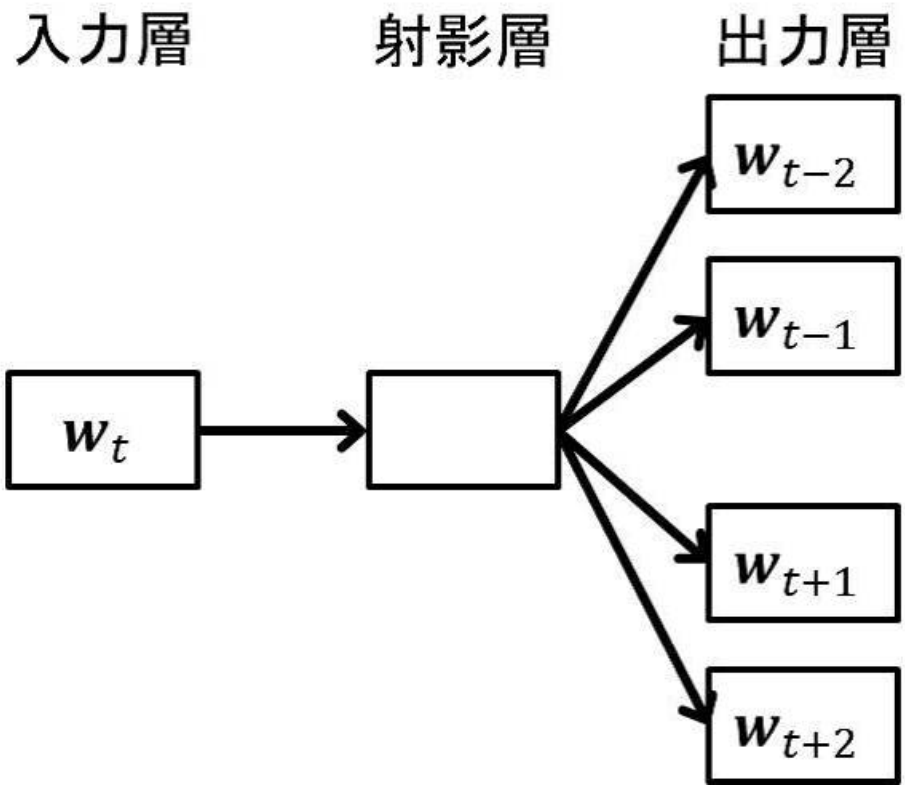

図 2.2: Skip-gram モデル

第 3 章

関連研究

本章では，マルウェアの亜種判別の関連研究について述べる．

3.1 機械学習を用いた亜種判別

機械学習を用いたマルウェアの解析の研究がおこなわれている．中村 [8] は本研究の対象と同じ API コールに着目し，API コールの N-gram のディリクレ分布をもとに，Kullback-Leibler 情報量を用いて亜種マルウェアの推定をおこなっている．青木ら [6] は，マルウェアが使用した API の名前を出現する順番の N-gram で抽出し特徴量とすることで，マルウェアの種類ごとに判別に有効な N-gram が存在することを示している．堀合 [9] はマルウェアの動的解析の前後での解析環境のログの差分を マルウェアの挙動を表す情報としてベクトル表現に変換して，ベクトルのハミング距離を計算して距離の近いものを亜種であると判定するマルウェアの亜種推定法を提案している．

このように従来の機械学習を用いる手法においては，経験則や専門家の考察に基づいて特徴量を決定する必要があるため，増加するマルウェアのファミリー数に対応できないという懸念がある．本論文で提案する手法では，Deep Learning を用いて人の手によらない自動的な特徴の決定が可能である．

3.2 ディープラーニングを用いた亜種判別

ディープラーニングを用いたマルウェア亜種判別の研究として，筆者ら [10] は，先行研究の中で Word Vector の拡張である Paragraph Vector [5] と呼ばれる文章の特徴をベクトル化する

手法を用いて，マルウェアの亜種判別をおこなった．Paragraph Vector を用いた先行研究においては，マルウェア動的解析結果に含まれる API 名と API の引数を順番に抽出したものを文章と見立て Paragraph Vector で表すことで，人の手によらず自動的にマルウェアの特徴を抽出・作成し，それによってマルウェアの亜種判別が可能であることを示した．

先行研究においては Paragraph Vector を用いてマルウェアの特徴を表現した．Paragraph Vector を計算するには前段階として Word Vector の計算が必要であり，Paragraph Vector を計算する過程で Word Vector が生成される．本論文では，Paragraph Vector と比較して計算コストの小さい Word Vector を用いることでマルウェアの亜種判別が可能であることを示す．

第 4 章

提案手法

4.1 提案手法の概要

本論文では，マルウェアの動的解析結果に含まれる API の実行を順番に並べたもの (API コール列) を抽出し，各 API を Word Vector で表したものを用いてマルウェアの亜種判別を可能とする特徴を作成する．Word Vector は自然言語の解析手法であるが，API コール列の各 API を単語に見立てると，API が連なってマルウェアの挙動を説明する，という点で API コール列は自然言語における文章と類似していると捉えられる．そこで，API コール列を文章と見立て，Word Vector を利用してマルウェアの特徴を表す，というのが本手法の着眼点である．

本手法は Deep Learning を応用して，人の手による特徴抽出を行わずに自動的に特徴を作成できる．先行研究の Paragraph Vector を利用する方法と比較して，Word Vector を用いて Paragraph Vector を求める過程が削減される．このため，Word Vector のみを用いる本論文の提案手法は計算量や時間といったリソースを削減可能な，より軽量な手法である．

提案手法は次の 3 つの手順からなる．まず手順 1 として，マルウェアの動的解析結果から抽出した API コール列をもとに各 API の Word Vector を作成する．次に手順 2 として，得られた Word Vector を用いてマルウェアの API コール列を特徴ベクトルで表す．最後に手順 3 として，API コール列の特徴ベクトルを SVM に入力・学習させ，学習した SVM によって未知のマルウェアが亜種であるか否かの判別をおこなう．それぞれの手順の詳細を以下に述べる．

4.2　手順 1: Word Vector の作成

まず，マルウェアごとに動的解析結果から得られる API の関数名を実行順に並べ，これを API コール列とする．図 4.2 に API コール列の例を示す．図 4.2 のように，同じ API が複数回登場する場合もある．

LdrGetDllHandle LdrLoadDll NtDelayExecution LdrLoadDll NtProtectVirtualMemory NtFreeVirtualMemory NtFreeVirtualMemory NtFreeVirtualMemory NtFreeVirtualMemory NtProtectVirtualMemory NtFreeVirtualMemory LdrLoadDll LdrLoadDll LdrLoadDll LdrLoadDll LdrLoadDll LdrLoadDll LdrLoadDll LdrLoadDll LdrLoadDll NtProtectVirtualMemory ...

図 4.1: API コール列の例

API コール列は 1 つのマルウェアが解析中に実行した API からなるため，1 検体につき 1 つの列が作成される．ここでは解析したい検体すべての API コール列を作成する．したがって，検体数ぶんの API コール列が作成されることになる．作成されたすべての API コール列を利用して，各 API 名に対する Word Vector を計算する．

4.3　手順 2: API コール列の特徴ベクトルの作成

手順 1 で得られた Word Vector を用いて API コール列の特徴ベクトルを求める．Word Vector は計算方法の性質上，前後に出現する単語，つまり文脈にもとづいて決定される．Word Vector を求める際にすでに前後の文脈が考慮されているため，出現する単語の Word Vector を単純に足し合わせるだけで文脈も考慮した文章の特徴を捉えることができると考える．このため，2.2.1 節で述べたような出現の順序を考慮しないという BoW の欠点を補える．また， 2.2.2 節で述べたように，Bit BoW という表現方法が考えられる．この方法でも Word Vector を利用することで単なる BoW と同様のメリットを得ることができる．したがって本手法では，API コール列の特徴ベクトルを，API コール列中の各 API の Word Vector に BoW および Bit BoW を応用して表現する．

API コール列の特徴ベクトルの求め方を図 4.2 および図 4.3 に示す．各 API には対応する学習済みの Word Vector が存在する．図 4.2 に示すのは BoW を適用した求め方である．API コール列に含まれる API の Word Vector をそれぞれ足し合わせ，平均を取る方法である．API コール列の長さは同じファミリー[1]であっても，マルウェアによって大きく異なるものもある．API コール列の長さによる結果への影響を抑えるために，Word Vector を足し合わせたベクトルの平均を取る．図中では A, C が 1 回，B が 2 回出現しているため，A, C の Word Vector をそれぞれ 1 回，B の Word Vector を 2 回足し合わせたものを平均を取るため 4 で割っている．

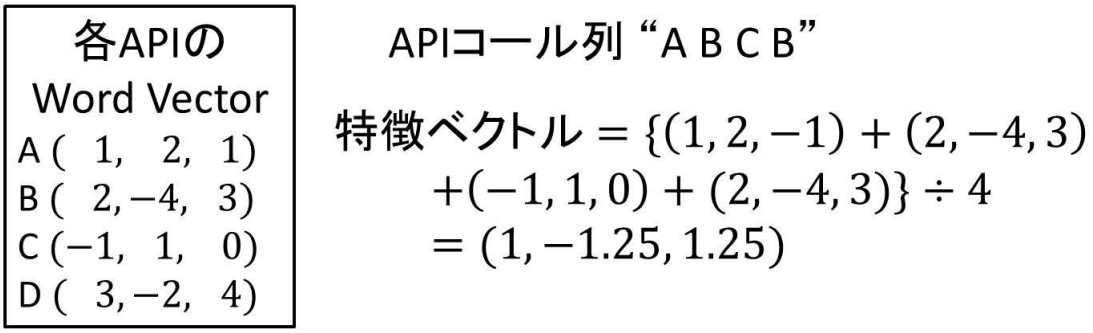

図 4.2: 一般的な BoW を Word Vector に適用し平均を取る方法

図 4.3 に示すのは，Bit BoW を Word Vector に適用して API コール列の特徴を求める方法である．API コール列の特徴ベクトルを API コール列に出現する API の Word Vector を足し合わせて求める．この際，同じ API が複数回出現する場合であっても Word Vector を加えるのは 1 回である．この方法では，足し合わされるベクトルの数は最大でも出現する API の種類と同数であるため，平均を取ることはしない．図中の API コール列には A, B, C の 3 つの API が出現しているため，3 つの API の Word Vector を足し合わせたものを API コール列の特徴ベクトルとしている．

[1]同一のマルウェアの亜種の集合をファミリーと呼ぶ.

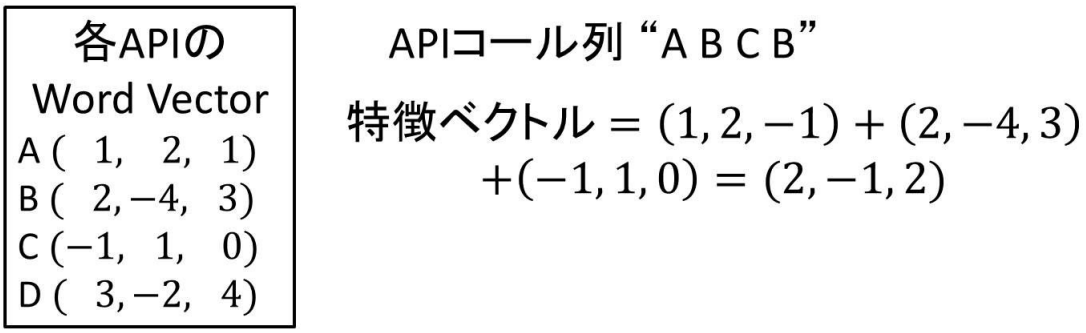

図 4.3: Bit BoW を Word Vector に適用する方法

4.4　手順 3: マルウェア亜種の判別

手順 2 で API コール列ごと，つまり検体ごとに特徴ベクトルを得た．得られた特徴ベクトルに，亜種であるか否かを判別する対象のファミリーの検体と，それ以外のファミリーの検体の 2 種類のラベル付けを行う．ラベル付けした特徴ベクトルを教師データとして入力して，教師あり学習器である SVM に学習させる．学習の後に，亜種かどうかを判別したいマルウェアの特徴ベクトルをテストデータとして学習済みの学習器に入力して，マルウェアが亜種であるか否かを判定する．

手法の一連の手順を図 4.4 に示す．まず，動的解析結果から API コール列を抽出する．次に API コール列を用いて Deep Learning を行い，Word Vector を求める (図中の学習 1)．Word Vector をもとに，BoW を応用して各マルウェアの特徴ベクトルを計算する．そして特徴ベクトルにラベル付けしたものを教師あり学習器である SVM に学習させ分類器を作成し (図中の学習 2)，テストデータを入力しマルウェアの亜種判定を行う．

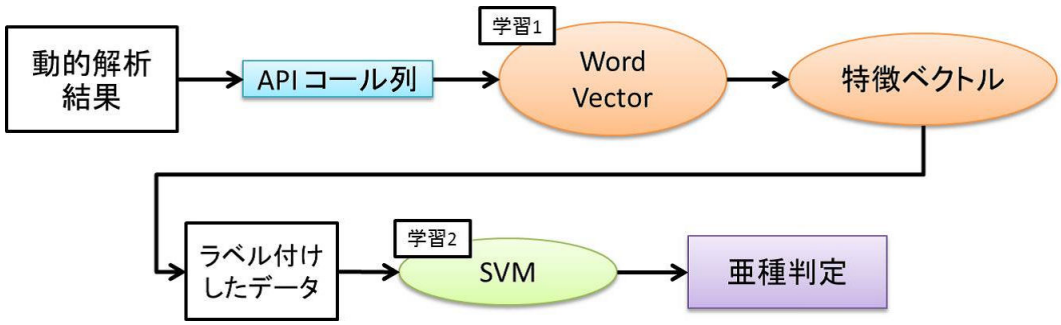

図 4.4: 提案手法の概要図

第 5 章

評価実験

5.1 実験の概要

本論文の提案手法の有効性を示すために評価実験をおこなう．提案手法を亜種を多く含むマルウェアの動的解析ログに適用し，どの程度の判別が可能であるかを測定する．

まず実験 1 として，手法の有効性を確認するために Word Vector を用いない単純な BoW や，Word Vector を用いた BoW と Bit BoW といった手法間での精度の比較をおこなう．次に，実験 2 でマルウェアの API コール列から作成する Word Vector および特徴ベクトルの次元数を変化させた際のファミリーごとの精度を求める．また，実験 3 では先行研究において用いた Paragraph Vector を利用した手法との比較をおこなう．最後に，有効性を示した上で動的解析の解析時間と判別精度との関係を実験 4 で調べる．解析時間が短ければ動的解析で得られる API コールの量，つまり学習に使用可能な API コールの量も減少する．解析時間の変化がどの程度精度に影響を与えるかを検討する．

5.2 実験に使用するデータ

今回の実験には FFRI Dataset 2013, 2014, 2015, 2016 [13, 14, 15, 16] に含まれるマルウェアの動的解析の結果を用いた．このデータセットは株式会社 FFRI が独自に収集した PE (Portable Executable) 形式かつ Windows プラットフォーム上で実行可能なマルウェアの動的解析結果である．この動的解析はオープンソースのマルウェア解析ツールである Cuckoo Sandbox [18] を用いて行い，1 検体あたり 90〜120 秒間 (データセットによって異なる) 実行した結果が json 形式でデータセットに記載されている．データセットに含まれる Cuckoo Sandbox での動的解

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

図 5.1: Kaspersky の命名規則

析で得られるデータ項目を表 5.1 に示す．実験では，データ項目内の behavior に含まれる API 呼び出しに関する項目を使用する．

表 5.1: Cuckoo Sandbox での動的解析で得られるデータ項目

項目	内容
info	解析の開始，終了時刻，id など
yara	yara (OSS のマルウェア検知・分類エンジン) の標準ルールとの照合結果
signatures	ユーザー定義シグニチャとの照合結果
virustotal	VirusTotal の検査履歴との照合結果
static	検体のファイル情報 (インポート API，セクション構造等)
dropped	検体が実行時に生成したファイル
behavior	検体実行時の API ログ (PID，TID，API の関数名，引数，返り値等)
processtree	検体実行時のプロセスツリー (親子関係)
summary	検体が実行時にアクセスしたファイル，レジストリ等の概要情報
target	解析対象検体のファイル情報 (ハッシュ値等)
debug	検体解析時の Cuckoo Sandbox のデバッグログ
strings	検体中に含まれる文字列情報
network	検体が実行時に行った通信の概要情報

FFRI Dataset 2014 には Cuckoo Sandbox での解析結果に加えて，FFRI 社のマルウェア自動解析ツールである FFR yarai analyzer Professional [19] を用いて動的解析を行った結果も含まれるが，本論文では解析方法の統一のため Cuckoo Sandbox での動的解析結果を使用した．また，FFRI Dataset 2016 には Windows 10 と Windows 8.1 での動的解析結果が含まれている．本論文の実験では Windows 10 での動的解析結果を使用した．

本研究では動的解析結果中の API 呼び出しに着目する．そこで，データセットに含まれる 16,887 検体のうち，1 つ以上の API 呼び出しログが取得できている 16,759 検体を実験に用いた．なお，本研究におけるマルウェアの名称は Kaspersky の検知結果を使い，亜種推定の実験における正解を Kaspersky の分類とした．Kaspersky の命名規則は文献 [12] にある．フォーマットを図 5.2 に示す．Prefix と Variant は必須な部分ではなく，存在しないこともある．Prefix はマルウェアを検知したサブシステムを，Variant は亜種をそれぞれ示す．Bahaviour は検出されたマルウェアが何をするか動作を表現し，Platform は実行された環境，Name はマルウェアの

ファミリー名を指す．本論文では Behaviour から Name までが一致しているマルウェア同士を亜種であると定義する．ここで Kaspersky を採用した理由は，検知率の高さから先行研究 [7, 8] において既に利用されているからである．

5.3　サンプリングしたデータセット

本論文では，マルウェアがあるマルウェアの亜種であるか否かを判定するために，データセット中に亜種が多く含まれているファミリーを実験に用いる．FFRI Dataset 2013 – 2016 において，亜種の数が 100 検体以上あるファミリー 24 種類を表 5.2 に示す．

表 5.2: 主要な検体名とデータセットに含まれる数

検体名	検体数
Trojan.Win32.Generic	2215
DangerousObject.Multi.Generic	801
Trojan.Win32.Waldek	687
Trojan-Spy.Win32.Zbot	581
Trojan.Win32.Yakes	577
Backdoor.Win32.Androm	445
Trojan.Win32.Agent	384
Trojan.Win32.Inject	351
Worm.Win32.WBNA	308
Hoax.Win32.ArchSMS	304
Trojan-PSW.Win32.Fareit	272
Trojan-PSW.Win32.Tepfer	218
Backdoor.Win32.DarkKomet	206
Trojan-Ransom.Win32.Foreign	204
Trojan-Dropper.Win32.Injector	185
Trojan.Win32.Jorik	171
Packed.Win32.Tpyn	163
Trojan.Win32.Kovter	145
Trojan.Win32.Scar	127
Downloader.Win32.LMN	123
Worm.Win32.Vobfus	120
Backdoor.Win32.Matsnu	110
Trojan-Downloader.Win32.Upatre	108
Trojan.Win32.Llac	100

表 5.2 に挙げられているもののうち，Trojan.Win32.Generic，DangerousObject.Multi.Generic の 2 種類は，Generic という名前の通り特定の種類ではなく「その他」として扱われているため，今回の亜種判定の実験の対象から除く． さらに，正確な精度を算出するために，亜種か否かを判定する検体とそれ以外の検体の数が等しくなるようにランダムにサンプリングした

データセットを作成する．例えば表 5.2 において，前述の 2 種類の「その他」を除くとファミリーは 22 種類存在する．Trojan.Win32.Waldek について判定するためのデータセットとして Trojan.Win32.Waldek とそれ以外の 21 種類のファミリーに属するマルウェアが 21:1 の比率で存在するデータセットを作成して，学習する際の教師データおよびテストデータとして用いる．

5.4　実験の方法

実験の手順を以下に示す．まず，対象の 16,759 検体の動的解析の結果から，検体ごとに API 名を実行順にすべて抽出して API コール列を作り，それを Word Vector を作成するためのコーパスとする．1 検体につき 1 つの API コール列が作成できるため，16,759 個の API コール列からなるコーパスとなる．

次に，作成したコーパスから Word Vector を作成する．Word Vector の計算には，sentence2vec [20] に含まれる word2vec の実装を使用した． 2.3.1 節および 2.3.2 節に示した Word Vector の学習アルゴリズムについては，予測精度が高いと言われている Skip-gram モデルを採用した．作成した Word Vector を用いて，BoW および Bit BoW を適用した方法で各検体の特徴ベクトルを計算する．BoW を適用した方法では，図 4.2 で示したように対象の検体の API コール列に出現する API の Word Vector を出現回数も加味して平均を取ったものを検体の特徴ベクトルとする．Bit BoW を適用した方法では，図 4.3 で示した通り出現回数を加味せず，出現した API の Word Vector の和を検体の特徴ベクトルとする．

作成した全検体分の特徴ベクトルのうち，表 5.2 で示したファミリーから名前が Generic であるものを除いた 22 ファミリーに属するマルウェアの特徴ベクトルを用いる．前述したように亜種判定したいファミリーとそれ以外の 21 ファミリーの比率が 21:1 になるよう特徴ベクトルをランダムに選出し，データセットを作成する．ファミリーごとにデータセット作成を行うため，合計 22 種類のデータセットができることになる．各データセットの特徴ベクトルに亜種と判別したいファミリーのであるか，そうでないかを 1 または –1 のラベルで設定する．このように作成した教師データを学習器 (本実験では SVM を使用する) に入力して学習させる．学習の終わった学習器にテストデータを入力し，亜種であるか否かの二値分類を行い，判定の精度を測定する．亜種判別は検体数の都合上，4 分割交差検証を行った．SVM は LIBSVM [21] を使用し，SVM のパラメータはグリッドサーチにより最適な数値を求めた．

5.5　実験結果

5.5.1　実験 1: 他の特徴ベクトル作成方法との比較

まず，検体の特徴ベクトルを複数の方法で作成し，それらの精度を比較する．比較をおこなう特徴ベクトル作成方法を以下の表 5.5.1 に示す．Word Vector を用いる方法では，Word Vector の次元は 200 とした．Word Vector を使わない方法では，Word Vector の代わりに 2.2.1 節で例示したように一般的な BoW で用いられる，1 つの要素が 1 でありそれ以外の要素は 0 である 1-of-K 符号化されたベクトルを用いる．データセットに含まれる動的解析結果に出現する API は 283 種類であったため，特徴ベクトルは 283 次元のベクトルで表される．API コール列の長さにばらつきがある場合に平均を取るべきかどうかの比較のため，Sum WV の方法では Word Vector を回数も加味して足し合わせた後に平均を取らず，そのまま特徴ベクトルとしている．

表 5.3: 比較をおこなう特徴ベクトル作成方法

作成方法	Word Vector の使用	BoW ・ Bit BoW の使用	備考
Ave 1_of_K	×	BoW	
Bit 1_of_K	×	Bit BoW	
Ave WV	○	BoW	
Bit WV	○	Bit BoW	
Sum WV	○	BoW	平均を取らない

各マルウェアファミリーごとに亜種であるかを判別した SVM の精度を実験結果とする．ここで精度というのは，正しく亜種であるもしくは亜種でないと判定できたマルウェアの割合である．実験結果を表 5.4 および図 5.2 に示す．

表 5.4: 特徴ベクトル作成方法ごとの亜種判別精度 (%)

ファミリー名＼次元数	Ave 1_of_K	Bit 1_of_K	Ave WV	Bit WV	Sum WV
Trojan.Win32.Jorik	78.27	91.67	88.69	90.48	87.80
Worm.Win32.WBNA	71.83	81.55	93.25	92.06	90.08
Trojan.Win32.Agent	69.49	73.51	71.73	71.28	70.83
Worm.Win32.Vobfus	63.69	94.64	89.88	94.05	92.26
Trojan.Win32.Yakes	73.21	84.03	77.88	83.43	77.28
Trojan-PSW.Win32.Tepfer	64.88	76.49	76.49	74.70	70.54
Trojan-Spy.Win32.Zbot	59.92	80.85	75.00	80.06	74.01
Trojan-Dropper.Win32.Injector	63.99	77.38	76.19	71.73	74.40
Hoax.Win32.ArchSMS	59.52	96.83	96.43	96.23	94.44
Trojan.Win32.Inject	50.74	73.66	71.88	74.70	72.17
Trojan-Ransom.Win32.Foreign	67.56	83.33	78.27	83.33	74.70
Trojan.Win32.Scar	67.26	77.38	66.67	77.38	68.45
Trojan.Win32.Llac	61.31	74.40	69.05	78.57	73.81
Backdoor.Win32.DarkKomet	73.51	84.52	82.74	83.93	83.04
Downloader.Win32.LMN	50.60	94.05	91.07	94.05	87.50
Backdoor.Win32.Androm	56.79	81.19	77.50	84.76	75.71
Trojan-PSW.Win32.Fareit	57.74	82.94	79.76	81.75	78.77
Trojan-Downloader.Win32.Upatre	73.81	85.71	83.33	88.69	77.98
Packed.Win32.Tpyn	70.83	88.10	84.52	86.31	79.17
Trojan.Win32.Kovter	92.86	93.45	93.45	91.07	88.69
Trojan.Win32.Waldek	69.94	96.80	94.72	96.73	90.85
Backdoor.Win32.Matsnu	85.71	91.67	90.48	93.45	93.45
平均値	67.43	84.73	82.23	84.94	80.72
最高値	92.86	96.83	96.43	96.73	94.44
最低値	50.60	73.51	66.67	71.28	68.45

Ave 1_of_K の方法が特に精度が低く，単純な BoW の表現方法では亜種の判別が難しいことが読み取れる．Ave 1_of_K の方法を除くと平均 80% 以上，最大 96.83% の精度が実現できており，高い精度でマルウェアの亜種判別ができている．精度が 75% 程度と提案手法では精度の悪いファミリーもある．Word Vector を用いた場合でも用いない場合でも，Bit BoW を適用した方法の精度が高くなっていることが分かる．Ave WV と Sum WV の比較では全体的に Ave WV の精度が高くなっている．平均を取らない場合，API コール列の長さによって特徴ベクトルの大きさがばらついてしまうため，精度を下げる結果となったと考えられる．Bit BoW を用いた方法では，Worm.Win32.WBNA のように Word Vector を用いた方が用いない場合よりも 10% 以上精度が高いファミリーもある．Word Vector を用いた方が精度が大きく向上するファミリーもあるため，Word Vector を用いて Bit BoW の表現でマルウェアの特徴ベクトルを表す提案手法が亜種判別に有効であることが分かる．

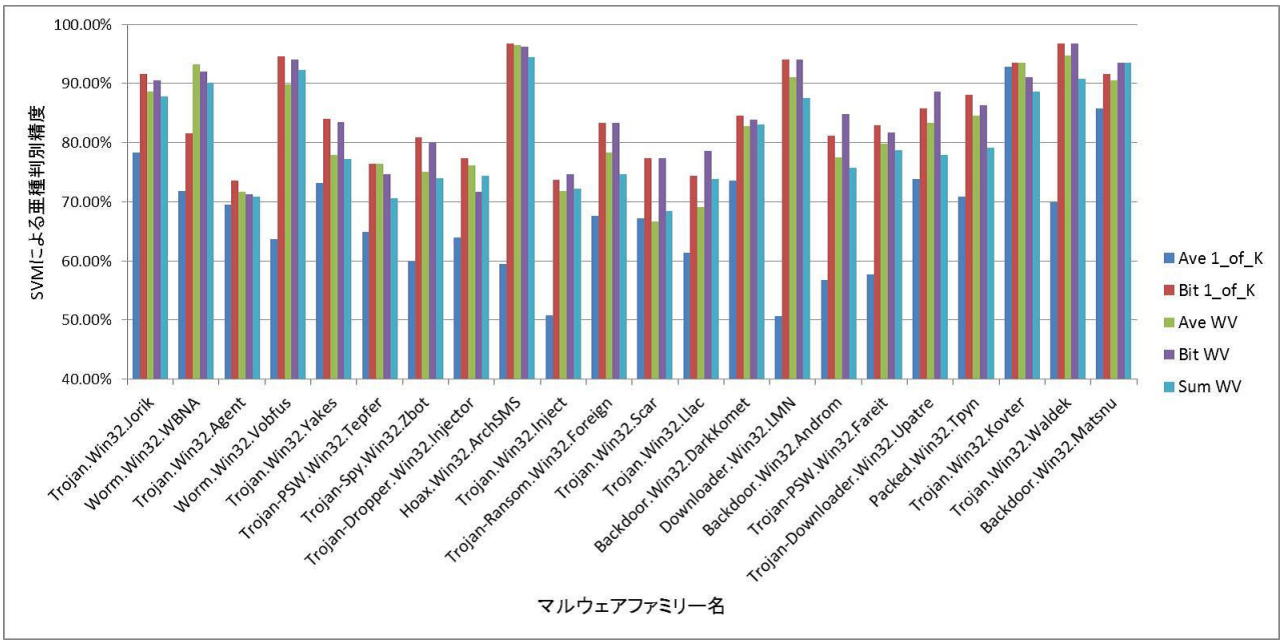

図 5.2: 特徴ベクトル作成方法ごとの精度の比較

5.5.2　実験 2: Word Vector・特徴ベクトルの次元数による比較

次に，求める Word Vector の次元数を 100 から 500 まで 100 刻みで変化させて特徴ベクトルを作成し，精度を比較する．特徴ベクトルの作成方法は，実験 1 の結果で精度の高かった Bit WV を採用した．実験結果を表 5.5 および表 5.3 に示す．

表 5.5: Bit WV の手法での次元数ごとの SVM の精度 (%)

ファミリー名＼次元数	d100	d200	d300	d400	d500
Trojan.Win32.Jorik	91.07	90.48	91.07	91.07	91.67
Worm.Win32.WBNA	92.06	92.06	92.46	92.26	92.66
Trojan.Win32.Agent	71.13	71.28	71.87	72.77	72.92
Worm.Win32.Vobfus	94.05	94.05	93.45	91.67	92.86
Trojan.Win32.Yakes	81.94	83.43	82.64	82.74	83.43
Trojan-PSW.Win32.Tepfer	74.70	74.70	75.30	74.40	76.19
Trojan-Spy.Win32.Zbot	79.86	80.06	79.37	80.46	80.26
Trojan-Dropper.Win32.Injector	75.30	71.73	70.54	72.02	69.94
Hoax.Win32.ArchSMS	96.43	96.23	96.03	95.63	96.03
Trojan.Win32.Inject	73.07	74.70	73.36	72.47	72.92
Trojan-Ransom.Win32.Foreign	80.65	83.33	80.36	80.36	79.76
Trojan.Win32.Scar	76.79	77.38	76.79	75.00	73.21
Trojan.Win32.Llac	72.62	78.57	76.79	77.38	76.79
Backdoor.Win32.DarkKomet	84.23	83.93	84.52	86.01	83.33
Downloader.Win32.LMN	95.83	94.05	95.83	95.83	95.83
Backdoor.Win32.Androm	84.64	84.76	84.17	84.40	85.12
Trojan-PSW.Win32.Fareit	81.55	81.75	80.75	81.75	81.35
Trojan-Downloader.Win32.Upatre	86.31	88.69	89.29	88.10	88.69
Packed.Win32.Tpyn	86.90	86.31	86.31	86.90	86.90
Trojan.Win32.Kovter	91.07	91.07	91.67	91.67	91.07
Trojan.Win32.Waldek	96.06	96.73	96.80	96.43	96.50
Backdoor.Win32.Matsnu	94.64	93.45	95.24	95.24	93.45
平均値	84.59	84.94	84.75	84.75	84.59
最高値	96.43	96.73	96.80	96.43	96.50
最低値	71.13	71.28	70.54	72.02	69.94

次元数を変化させることで最大 5% ほど精度が変化していることが分かる．200 次元のものの精度が比較的高いため，以降の実験では 200 次元の Word Vector を用いて Bit BoW で表現した特徴ベクトルを亜種判別に用いる．

5.5.3　実験 3: 先行研究の手法と提案手法の比較

先行研究 [10] で用いた，Paragraph Vector を利用して API コール列の特徴を表す手法と提案手法の比較をおこなう．Paragraph Vector の計算には sentence2vec [20] を用いて，ベクトルの次元数は先行研究と揃えるため 100 とした．

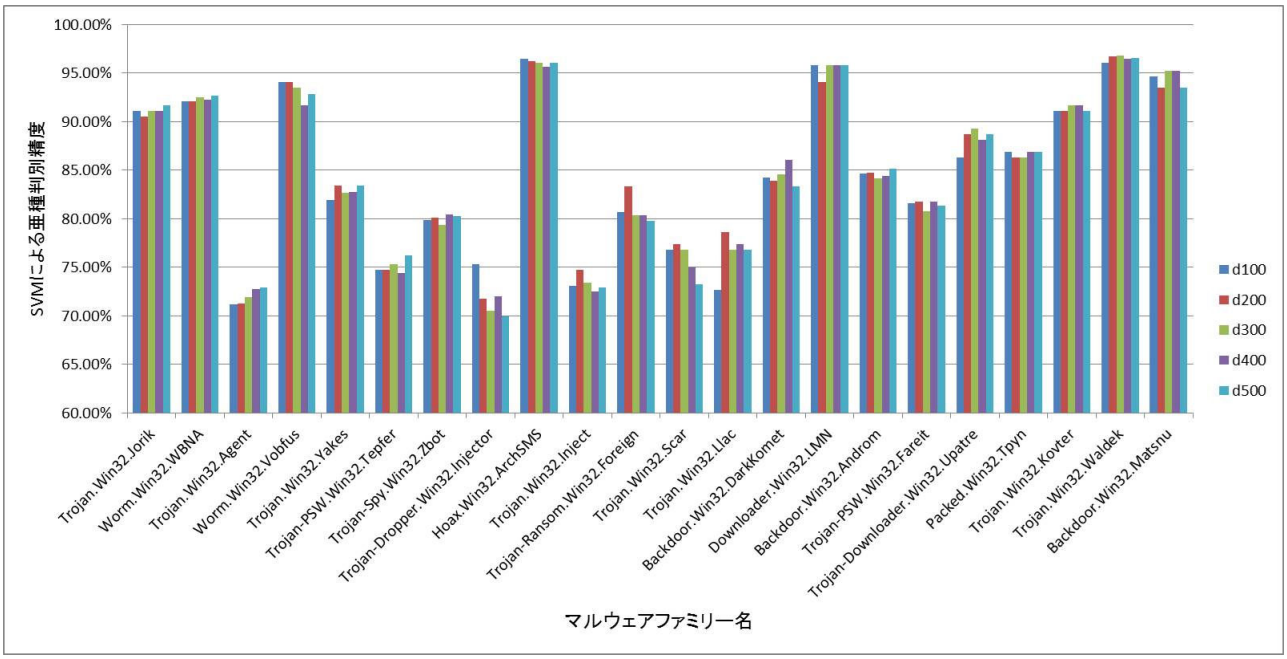

図 5.3: Bit WV での次元数ごとの精度の比較

表 5.6: Word Vector と Paragraph Vector を用いた場合の亜種判別精度 (%)

ファミリー名＼手法	Bit WV	Paragraph Vec
Trojan.Win32.Jorik	90.48	88.10
Worm.Win32.WBNA	92.06	91.87
Trojan.Win32.Agent	71.28	70.54
Worm.Win32.Vobfus	94.05	89.29
Trojan.Win32.Yakes	83.43	79.17
Trojan-PSW.Win32.Tepfer	74.70	75.60
Trojan-Spy.Win32.Zbot	80.06	77.18
Trojan-Dropper.Win32.Injector	71.73	73.81
Hoax.Win32.ArchSMS	96.23	95.63
Trojan.Win32.Inject	74.70	69.49
Trojan-Ransom.Win32.Foreign	83.33	80.95
Trojan.Win32.Scar	77.38	74.40
Trojan.Win32.Llac	78.57	72.02
Backdoor.Win32.DarkKomet	83.93	82.44
Downloader.Win32.LMN	94.05	89.29
Backdoor.Win32.Androm	84.76	79.17
Trojan-PSW.Win32.Fareit	81.75	80.16
Trojan-Downloader.Win32.Upatre	88.69	84.52
Packed.Win32.Tpyn	86.31	82.74
Trojan.Win32.Kovter	91.07	91.07
Trojan.Win32.Waldek	96.73	91.67
Backdoor.Win32.Matsnu	93.45	94.05
平均値	84.94	82.42
最高値	96.73	95.63
最低値	71.28	69.49

全体的に Word Vector を用いて特徴ベクトルを求める提案手法の方が Paragraph Vector を特徴ベクトルとする先行研究の手法よりも精度が高くなっていることが分かる．Trojan.Win32.Inject や Backdoor.Win32.Androm のファミリーのように 5% 以上 Bit WV の方が精度が高くなっているファミリーもある．Paragraph Vector を求めるには Deep Learning で Word Vector を計算したうえで更に 2 回目の Deep Learning をおこなう必要がある．提案手法では Deep Learning は Word Vector を求めるための 1 回で済み，なおかつ Word Vector をもとに特徴ベクトルを計算するための所要時間も Deep Learning を用いる Paragraph Vector より短くすむ．したがって，先行研究の手法より提案手法の方が時間・計算量の面で軽量かつ亜種判別精度が高いという点で優れていると言える．

5.5.4　実験 4: 動的解析の時間と亜種判別精度の関係の調査

実験 3 までで，API コール列から求めた Word Vector を用いてマルウェアの特徴ベクトルを計算することが亜種判別において有効であることを確認した．本節では，Word Vector を求めるために必要な動的解析の解析時間と亜種判別精度の関係について検討する．

まず，FFRI データセットのすべての動的解析ログに存在する API コールの分布を表 5.7 および図 5.4 に示す．表では各経過時間の間に呼び出された API コールの数と，その経過時間までの API コール数の累積割合を示している．例えば 0.3 秒のところでは，解析を開始して 0.2 秒から 0.3 秒の間に 1379625 個の API が呼び出され，開始から 0.3 秒の間にデータセット内の API コール全体のうち 12.58% が存在していることを示している．

表 5.7: 動的解析経過時間における API コール数

経過時間 [sec]	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
API コール数	2,358,457	2,073,723	1,379,625	917,365	936,533	728,236	687,482	762,158	678,217
累積割合 [%]	5.11%	9.60%	12.58%	14.57%	16.60%	18.17%	19.66%	21.31%	22.78%
経過時間 [sec]	1	2	3	4	5	6	7	8	9
API コール数	696,155	8,097,050	6,781,108	3,444,069	1,814,091	1,690,625	1,139,081	938,311	774,522
累積割合 [%]	24.29%	41.81%	56.49%	63.95%	67.88%	71.54%	74.00%	76.03%	77.71%
経過時間 [sec]	10	20	30	40	50	60	70	80	90
API コール数	704,976	2,777,803	1,132,102	1,663,188	813,301	618,170	626,881	610,731	561,096
累積割合 [%]	79.24%	85.25%	87.70%	91.30%	93.06%	94.40%	95.76%	97.08%	98.30%
経過時間 [sec]	100	110	120	130					
API コール数	288,384	251,124	247,891	44					
累積割合 [%]	98.92%	99.46%	100.00%	100.00%					

データセットでは Cuckoo Sandbox 上で 90 秒〜120 秒間解析をおこなっている．図 5.4 に示

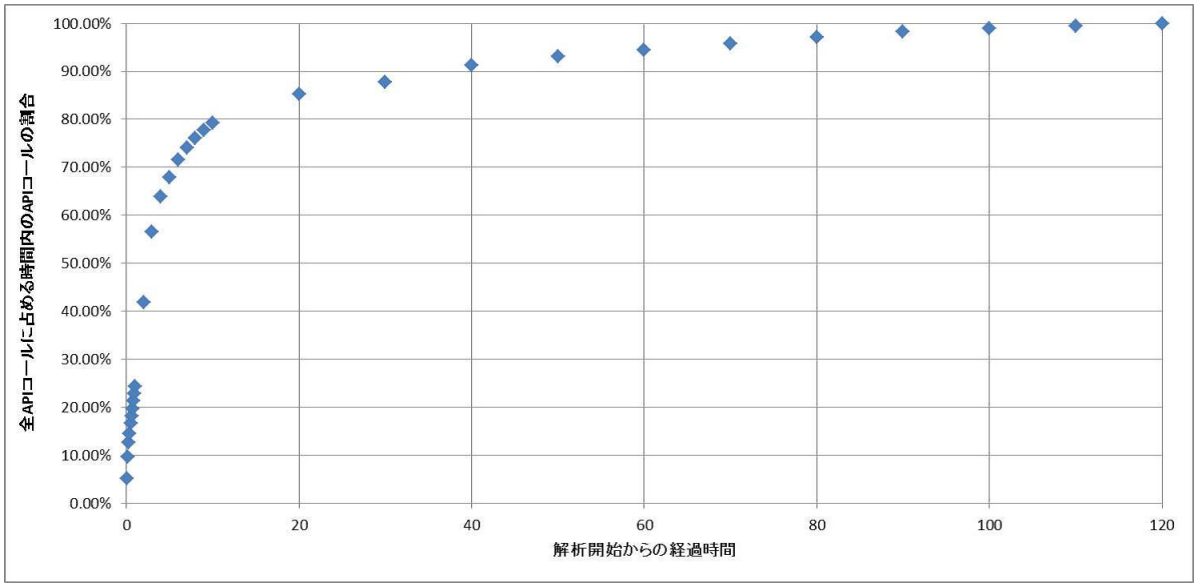

図 5.4: 動的解析時間と API コール数の関係

した通り，解析開始から 10 秒程度で割合は急激に増え，40 秒程度で 90% 以上の API コールが出現していることが分かる．

次に，どの程度の分量の API コールを動的解析によって取得できれば亜種判別に有効な特徴として用いることができるのかを調べる．これにより，亜種判別のために必要な動的解析時間の目安が得られることが期待される．データセットから API コール列を抽出する際に，特定の経過時間以内に呼び出された API コールのみを抽出する．API コール列を抽出してからはこれまでの実験同様の手順を取り，特徴ベクトルは Bit BoW を 200 次元の Word Vector に適用する方法 (Bit WV) で求める．表 5.7 を参考に，API コール数の割合が約 25% 刻みとなるよう 1 秒，3 秒，8 秒，40 秒，すべて (120 秒) の 5 通りについて API コールを抽出して特徴ベクトルを作成し，亜種判別をおこなった．実験結果を表 5.8 および図 5.5 に示す．

表 5.8: 各解析時間における亜種判別精度 (%)

ファミリー名＼解析時間 (API コール数の割合)	1sec (24%)	3sec (56%)	8sec (76%)	40sec (91%)	120sec (100%)
Trojan.Win32.Jorik	72.92	90.18	88.99	90.48	90.48
Worm.Win32.WBNA	86.51	92.46	93.85	91.67	92.06
Trojan.Win32.Agent	56.70	70.24	70.98	71.43	71.28
Worm.Win32.Vobfus	61.90	94.05	92.86	92.86	94.05
Trojan.Win32.Yakes	61.41	82.14	83.04	82.84	83.43
Trojan-PSW.Win32.Tepfer	58.33	76.19	77.98	75.00	74.70
Trojan-Spy.Win32.Zbot	55.16	76.19	78.57	79.07	80.06
Trojan-Dropper.Win32.Injector	50.00	69.64	73.81	72.32	71.73
Hoax.Win32.ArchSMS	74.21	94.25	95.63	96.03	96.23
Trojan.Win32.Inject	57.74	66.82	70.68	73.81	74.70
Trojan-Ransom.Win32.Foreign	60.71	74.11	80.36	80.95	83.33
Trojan.Win32.Scar	55.95	74.40	80.95	73.21	77.38
Trojan.Win32.Llac	52.98	74.40	75.60	75.00	78.57
Backdoor.Win32.DarkKomet	63.69	83.33	84.23	83.93	83.93
Downloader.Win32.LMN	76.79	87.50	94.64	95.83	94.05
Backdoor.Win32.Androm	59.17	76.07	81.19	83.45	84.76
Trojan-PSW.Win32.Fareit	60.52	77.98	78.77	80.56	81.75
Trojan-Downloader.Win32.Upatre	55.36	88.10	85.71	88.69	88.69
Packed.Win32.Tpyn	66.67	90.48	88.10	90.48	86.31
Trojan.Win32.Kovter	70.83	89.29	93.45	92.26	91.07
Trojan.Win32.Waldek	72.47	95.98	95.91	96.28	96.73
Backdoor.Win32.Matsnu	67.86	95.24	96.43	91.67	93.45
平均値	63.54	82.68	84.62	84.45	84.94
最高値	86.51	95.98	96.43	96.28	96.73
最低値	50.00	66.82	70.68	71.43	71.28

動的解析の時間の増加とともに，取得する API コールの数が増えるため精度も向上する．解析時間 1 秒では精度は極めて低いが，解析時間 3 秒以降は精度が急激に上がり，解析時間が 8 秒以上になると精度が上がるファミリーもあるが横ばいのファミリーも多いことが分かる．

5.6　考察

5.6.1　特徴ベクトルの計算方法

実験 1 にて比較した通り，各 API を Word Vector を用いて表し，それらを用いてマルウェアの特徴ベクトルを計算した方が Word Vector を用いずに特徴ベクトルを求めた場合より高い精度での亜種判別をおこなえる．これにより Word Vector を用いた提案手法が，マルウェアの特徴を捉えるために有効であることが分かる．また，実験 3 の結果に示されるように Paragraph Vector を用いる先行研究より提案手法の方が精度，計算量の二点で優れていることが分かる．

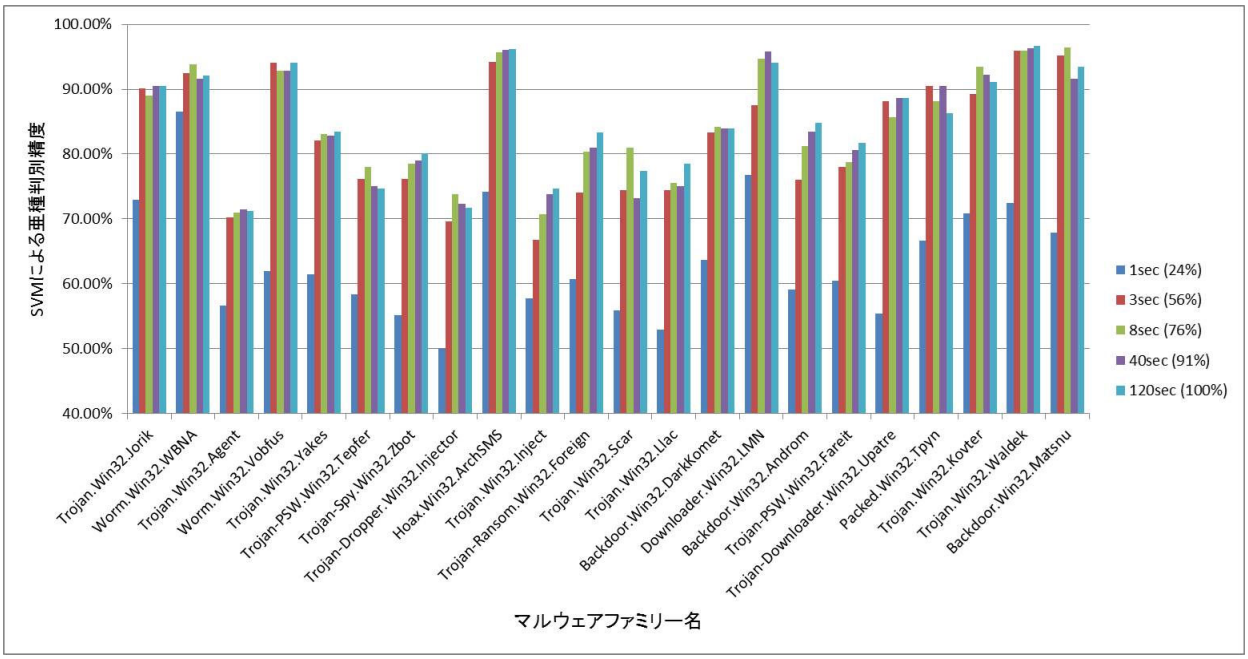

図 5.5: 動的解析時間と亜種判別精度の関係

Word Vector を用いる場合でも用いない場合でも，API コール列の特徴ベクトルを求める際に一般的な BoW を適用した方法よりも Bit BoW を適用した方法の方が亜種判別精度が高かった．一般的な BoW と比較した Bit BoW の特徴として，API の出現回数を加味しないというものがある．一般的な BoW では出現回数の多い API の要素にベクトルが偏ってしまう場合があるため，API の出現回数がマルウェアの特徴を捉える際にノイズになりうる．

出現回数がノイズとなる場合，重要なのは “出現すること” 自体であると考えられる．API の出現の組み合わせ，つまり API の共起が重要であると考えると，実験 1 において Bit 1_of_K の方法でも一定の精度での亜種判別ができたことと首尾一貫する．その状況において Word Vector を用いた方が精度が高くなったのは，Word Vector を求める際に対象の API の前後に出現する API，つまり API コール列の文脈を考慮したベクトルが生成されているためであると考えられる．

5.6.2　動的解析の実行時間

実験 4 の結果に示した通り，動的解析の実行時間を長くすれば API コールの取得数も増えるため，マルウェア亜種判別の精度が高くなることが分かった．API コールがそのままマルウェアの特徴ベクトルに直結するため，十分な量の API コールを取得できなければ当然精度が低

くなる．

その一方で，図 5.5 に示した通り現状の 90〜120 秒まででなくとも一定時間まで API コールを取得できれば 90〜120 秒まで動的解析をおこなった場合と同等の精度を実現できることが分かる．図 5.4 に示したように，40 秒ほど動的解析をおこなうと 90〜120 秒まで動的解析をおこなった場合に取得できた API コール数の 90% 以上の API コールを取得できる．図 5.5 の示している結果も併せて，動的解析時間は最低でも 3 秒程度必要であり，40 秒程度の動的解析でも亜種判別のための相当量の情報が得られると考える．

第 6 章

まとめ

6.1 結論

本論文は，マルウェアの動的解析の結果に含まれる API 名の列を自然言語の 1 つの文章のように取り扱い，Word Vector を “出現回数を加味しない”BoW で表現することによって特徴ベクトルを作成する手法を提案した．また，作成した特徴ベクトルによってマルウェアの亜種判定が可能であることを示した．判別精度はファミリーによって異なるが，SVM を用いることで平均 84.94% ，最大 96.73% の精度が実現可能であった．

本手法は Deep Learning によって得られる Word Vector をもとに特徴ベクトルを作成するため，人手によらず自動的にマルウェアの特徴ベクトルを求めることができる．また，自動的におこなえるマルウェアの動的解析の結果に着目しているため，解析から判別までの一連の処理を自動でおこなうことへの応用も可能である．

また，BoW と Bit BoW の比較により，API の出現回数がマルウェアの特徴を捉えるにあたりノイズとなりうることを示した．さらに，動的解析の経過時間と亜種判別精度を比較することで，マルウェアの特徴を捉えるために必要な動的解析時間について考察した．

6.2 今後の課題

本論文で提案した手法に残された課題を述べる．提案手法では亜種判別としては不十分な精度となるファミリーが存在した．精度の低いファミリーについては API 名から得られる情報のみでは判別精度がこれ以上向上しないことが考えられる．そのため，API の引数・引数の返り値やおこなった通信など，動的解析から得られる別の情報も加えた亜種判別をおこなうことで

精度の向上が見込まれる．

本論文においては Word Vector を用いて API コール列の特徴をベクトル表現で表した．具体的にベクトル表現のどの要素が元の API コール列のどの要素と強く関係があるのか，というようなベクトルで表す前後の関連性についての考察が必要となる．自然言語における Word Vector や Paragraph Vector の成功例として良く引用される，ベクトル化した後のベクトル演算の意義が本手法の場合にはどの程度あるのかという考察も，本手法のさらなる発展のために有効であると考える．

謝辞

本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．研究活動を進めるにあたり，多くのご助言をいただいた先輩の武部嵩礼氏に心より感謝いたします．

最後に，研究に打ち込める環境を提供してくださった後藤研究室の諸氏に感謝いたします．

参考文献

[1] “McAfee 脅威レポート 2016 年第 1 四半期,” http://www.mcafee.com/jp/resources/reports/rp-quarterly-threats-may-2016.pdf, 2016.

[2] “G DATA SECURITYLABS MALWARE REPORT HALF-YEAR-REPORT JANUARY - JUNE 2015,” https://public.gdatasoftware.com/Presse/Publikationen/Malware_Reports/G_DATA_PCMWR_H1_2015_EN.pdf, 2015.

[3] 八木 毅, 青木 一史, 秋山 満昭, 幾世 知範, 高田 雄太, 千葉 大紀, “実践サイバーセキュリティモニタリング”, コロナ社, 2016.

[4] 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.

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

[6] 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, 2014.

[7] Akinori Fujino, Junichi Murakami, Tatsuya Mori, “Discovering similar malware samples using API call topics,” 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), pp. 140–147, Jan. 2015.

[8] 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, Oct. 2013.

[9] 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, Apr. 2009.

[10] 佐藤 拓未, 武部 嵩礼, 後藤滋樹, “Paragraph Vector を用いたマルウェアの亜種推定法,” コンピュータセキュリティシンポジウム 2016 論文誌, pp.299–304, Oct. 2016.

[11] 武部 嵩礼, 後藤滋樹, “Paragraph Vector を利用した亜種マルウェア推定法,” 電子情報通信学会総合大会講演論文集 2016 年_ 情報・システム (2), pp.197, 2016.

[12] “Rules for naming,” https://securelist.com/threats/rules-for-naming/

[13] 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, マルウェア対策研究人材育成ワークショップ 2013 (MWS2013), Oct. 2013.

[14] 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, Vol.114, No.118, CSEC, pp.125–131, Jul. 2014.

[15] 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用データセット〜MWS datasets 2015〜,” 情報処理学会, Vol.115, No.121, CSEC, pp.37–44, Jul. 2015.

[16] 高田 雄太, 寺田 真敏, 村上 純一, 笠間 貴弘, 吉岡 克成, 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 2016〜,” 情報処理学会, 2016-CSEC-74, CSEC, pp.1–8, Jul. 2016.

[17] 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.

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

[19] “FFR yarai analyzer Professional,”
http://www.ffri.jp/products/yarai_analyzer_pro/

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

[21] “Libsvm,”
https://github.com/arnaudsj/libsvm