目 次

第1章　序論

マルウェアの発見は増加の傾向にある．大手セキュリティベンダである McAfee のレポート [1] によると，2020 年第 2 四半期に，1 分あたり 419 件の脅威を発見したとされている．

マルウェアの発見数が膨大になる原因の 1 つに，マルウェアの亜種の存在がある．ある既存のマルウェアに対して，基本的な動作に変更は無く，一部分だけ変更を加えたものを，「マルウェアの亜種」と呼ぶ．マルウェアの亜種は，ツールを用いる事で容易に作成することができ，これがマルウェア発見数の増加に繋がっている．大量のマルウェアの亜種が生まれている中，そのマルウェアがどの既存マルウェアの亜種であるのかを正しく判定することができれば，それぞれのマルウェアに対し方策をとることができる．

佐藤らによる既存研究 [2] では，マルウェアの動的解析結果から得られる API コール列を特徴としてマルウェアの亜種分類を行う方法が提案されている．API コール列を Paragraph Vector の実装として sentence2vec [11] を用いて特徴ベクトルに変換し，その特徴量を用いて分類が行われている．

本研究では，API コール列をベクトル化する段階で，現在 Paragraph Vector のデファクトスタンダードの実装である Doc2Vec [9] を用いて特徴ベクトルを計算することを提案する．さらに，評価実験結果から，分類精度が向上しマルウェアの亜種判定に有効であることを示す．

第 2 章では，マルウェアの解析手法と実験に用いるデータセットである FFRI Dataset について述べる．第 3 章は特徴ベクトルの作成手法である Paragraph Vector について説明する．第 4 章で本研究の提案手法とその実験および結果を，第 5 章で結論を述べる．

第2章　マルウェアの解析手法とFFRI Dataset

2.1　マルウェアの解析手法

マルウェア解析には，目的，解析難易度や取得できる情報が異なるいくつかの解析プロセスが存在する [3].

2.1.1　表層解析

表層解析は，ファイル自体が悪性であるか否かの情報収集や，ファイルのメタ情報の収集を目的として行う解析プロセスである．アンチウイルスソフトで判定を行い，既知のマルウェアであるかを判断や，ファイルタイプや動作する CPU アーキテクチャの情報収集や，特徴的なバイト列 (通信先 URL やマルウェアが用いるコマンド等) を収集を行う．表層解析では，解析対象に対してツールを適用して情報を取得する．その他の解析のように，実際にマルウェアの動作やマルウェア内のプログラムコードの分析は行わない．

2.1.2　動的解析

動的解析は，マルウェアが実際に動作した際に端末上にどのような痕跡が残るのか，またどのような通信が発生するのかの情報収集を目的とした解析プロセスである．監視ツールをインストールした環境で実際にマルウェアを動作させ，ファイルやレジストリアクセスの情報を収集したり，通信を監視して通信先の IP アドレスやドメイン，URL，通信ペイロードといった情報を収集したりする．動的解析では，プログラムコードを詳細に分析しないためブラックボックス解析とも呼ばれる．

2.1.3　静的解析

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分析し，具備されている機能や特徴的なバイト列など詳細な情報を収集することを目的とした解析プロセスである．動的解析で実行されなかったコードを分析して潜在的に保有している機能

を明らかにしたり，マルウェア独自の通信プロトコルや通信先生成アルゴリズムのような動的解析だけでは特定が難しい情報の収集を行う．静的解析では，プログラムコードを詳細に分析するためホワイトボックス解析とも呼ばれる．

2.2　FFRI Dataset

本節では，本研究で用いるデータセットである FFRI Dataset について述べる．FFRI Dataset は，マルウェア対策研究人材育成ワークショップ (以下「MWS」) にて提供される研究用データセットの 1 つである．MWS は，研究者コミュニティから提供されたデータやサイバークリーンセンターハニーポットで過去に収集されたボット観測データを「研究用データセット」として活用するワークショップである．MWS 2020 [4] では，FFRI Dataset 2013〜2020 が提供された．

FFRI Dataset には，株式会社 FFRI セキュリティで収集したマルウェアの動的解析ログ，表層解析ログが含まれている．FFRI Dataset 2013〜2017 には動的解析ログの，2018〜2020 には表層解析ログのデータとなっている [5]．本研究では，動的解析ログデータである FFRI Dataset 2013〜2016 を用いる．FFRI Dataset 2013〜2016 の概要を以下に示す [6]．

- 2013:Cuckoo Sandbox ログファイル約 2600 検体分
- 2014:Cuckoo Sandbox，FFR yarai analyzer professional ログファイル約 3000 検体分
- 2015:Cuckoo Sandbox ログファイル約 3000 検体分
- 2016:Cuckoo Sandbox ログファイル約 8000 検体分

ここで，Cuckoo Sandbox とは，オープンソースのマルウェア解析システムである．仮想環境内でマルウェアを実行し，実行時の振る舞いをモニタリングすることが可能である．FFRI Dataset 2014 では，Cuckoo Sandbox での解析ログに加えて，FFR yarai analyzer professional を用いての解析ログが含まれているが，本研究では，Cuckoo Sandbox での解析ログファイルを実験に用いる．また，FFRI Dataset 2016 には，Windows 8.1 で動作させた動的解析ログと，Windows 10 で動作させた動的解析ログが存在するが，本研究では Windows 10 のものを用いることにする．

実際に FFRI Dataset の具体的なデータを見ていく．ログファイルは JSON 形式となっており，データ項目は表 2.1 のようになっている．

表 2.1　FFRI Dataset のデータ項目

データ項目	内容
info	解析の開始，終了時刻，id 等
signatures	ユーザ定義シグニチャとの照合結果
virustotal	VirusTotal から得られる情報
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 名，引数，返り値，動作概要等)
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

特に重要であるのは，“virustotal”と“behavior”の項目である．まず，“virustotal”項目の“scans”項目には，各セキュリティ会社がそのマルウェアをどのように命名しているか等が記述されている．例えば，図 2.1 では本研究で用いる Kaspersky の命名が書かれており，“Trojan.Win32.Jorik.Vobfus.fqzm”という名前のマルウェアであるということが分かる．そして，“behavior”項目には，API コールのログが時系列順に並んでおり，その引数や返り値等が記述されている．例えば，図 2.2 では“LdrLoadDll”が API コール名であり，API が呼ばれる順に時系列で API 名が記録されている．この API コールを取り出して，区切り文字を境に，時系列順に列挙したものを API コール列と呼ぶ．

```
"virustotal": {
        "scan_id": "c0471b6645919b3bb931ba205f90b6f127f3790d49043a1d642bca32d98ccf77-1350863687",
        "sha1": "3c8706145c9b330b004eddd5087eaf950718c3b5",
        "resource": "fc96305d53468a9c2b925c4012a509c1",
        "response_code": 1,
        "scan_date": "2012-10-21 23:54:47",
        "permalink": "https://www.virustotal.com/file/...,
        "verbose_msg": "Scan finished, scan information embedded in this object",
        "sha256": "c0471b6645919b3bb931ba205f90b6f127f3790d49043a1d642bca32d98ccf77",
        "positives": 37,
        "total": 44,
        "md5": "fc96305d53468a9c2b925c4012a509c1",
        "scans": {
                "MicroWorld-eScan": {
                        "detected": true,
                        "version": "12.0.250.0",
                        "result": "Gen:Variant.Barys.545",
                        "update": "20121022"
                },
                ...
                "Kaspersky": {
                        "detected": true,
                        "version": "9.0.0.837",
                        "result": "Trojan.Win32.Jorik.Vobfus.fqzm",
                        "update": "20121022"
                },
                ...
```


図 2.1　FFRI Dataset の“virustotal”項目の例

```
"behavior": {
        "processes": [
                {
                        "parent_id": "308",
                        "process_name": "FC96305D53468A9C2B925C4012A509C1.bin",
                        "process_id": "420",
                        "first_seen": "2013-02-28 12:03:55,646",
                        "calls": [
                                {
                                        "category": "system",
                                        "status": "FAILURE",
                                        "return": "0xc0000135",
                                        "timestamp": "2013-02-28 12:03:55,646",
                                        "thread_id": "432",
                                        "repeated": 0,
                                        "api": "LdrLoadDll",
                                        "arguments": [
                                                {
                                                        "name": "Flags",
                                                        "value": "1242916"
                                                },
                                                {
                                                        "name": "FileName",
                                                        "value": "C:\\WINDOWS\\system32\\VB6JP.DLL"
                                                },
                                                {
                                                        "name": "BaseAddress",
                                                        "value": "0x00000000"
                                                }
                                        ]
                                },
                                {
                                        "category": "registry",
                                        "status": "SUCCESS",
                                        "return": "0x00000000",
                                        "timestamp": "2013-02-28 12:03:55,656",
                                        "thread_id": "432",
                                        "repeated": 0,
                                        "api": "NtOpenKey",
                                        "arguments": [
                                                {
                                                        "name": "KeyHandle",
                                                        "value": "0x00000058"
                                                },
                                                {
                                                        "name": "DesiredAccess",
                                                        "value": "1"
                                                },
                                                {
                                                        "name": "ObjectAttributes",
                                                        "value": "Registry\\MACHINE\\System\\CurrentControlSet\\Control\\...
                                                }
                                        ]
                                },
                                ...
```


図 2.2　FFRI Dataset の“behavior”項目の例

第3章　機械学習

本章では，本研究で用いる機械学習の技術について述べる．1 つ目は，文書の数学的表現を行う技術，Paragraph Vector について，2 つ目に，二値分類の機械学習アルゴリズムである SVM(サポートベクターマシン) について説明する．

3.1　Word Vector

Word Vector は Mikolov ら [7] 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 では，単語をベクトル化 (数値化) することによって，単語間の関係性をベクトル演算によって求める事が可能である．Word Vector に変換された単語は，意味が近い単語同士はベクトルの値が近くなり，2 つのベクトルの差は 2 つの単語の関係性を表すといった特徴をもつ．

また，Word Vector を作成するために用いられる CBOW モデルおよび Skip-gram モデルについて，3.1.1 および 3.1.2 で述べる．

3.1.1　CBOW モデル

CBOW(Continuous Bag-of-Words) モデルの名前の中に含まれる Bag-of-Words とは，文章に登場する単語の出現回数の表現のことである．例えば，A，B，C，D の 4 語のみが登場する文章群において，“A B C B”という文章は，{1，2，1，0} という風に表現できる．これは，文章“A B C B”において A が 1 回，B が 2 回，C が 1 回，D が 0 回登場するということを示している．しかしながら，この表現では単語の出現回数のみを表しており，語順を考慮していない．

CBOW モデルを図 3.1 に示す．ここで，文章の単語群において，t 番目に出現する単語を w_t とする．

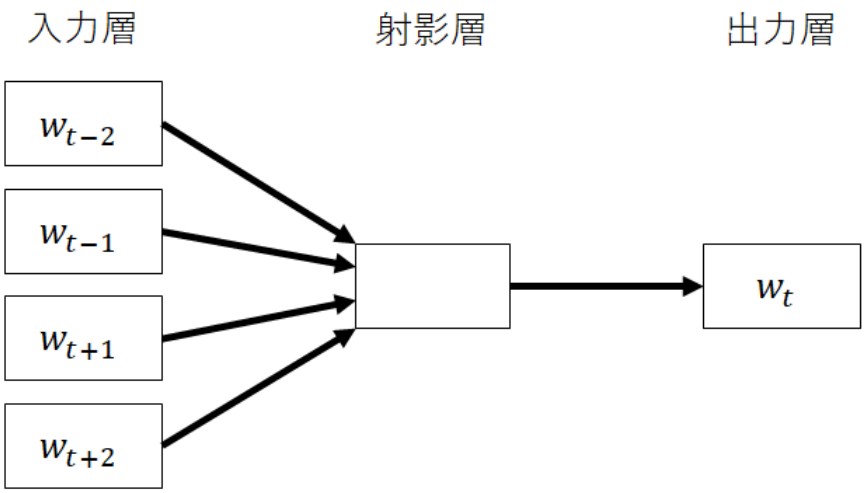

図 3.1　CBOW モデル

CBOW モデルは，周辺単語から出現する単語を予測するモデルである．w_t の前後 $2k$ 個の単語 $(w_{t-k}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+k})$ の Bag-of-Words をニューラルネットワークの入力とし，出力が w_t となるようなニューラルネットワークの重みを更新しながら学習を行う．このように学習を進め，Word Vector を求める．

3.1.2　Skip-gram モデル

Skip-gram モデルを図 3.2 に示す．

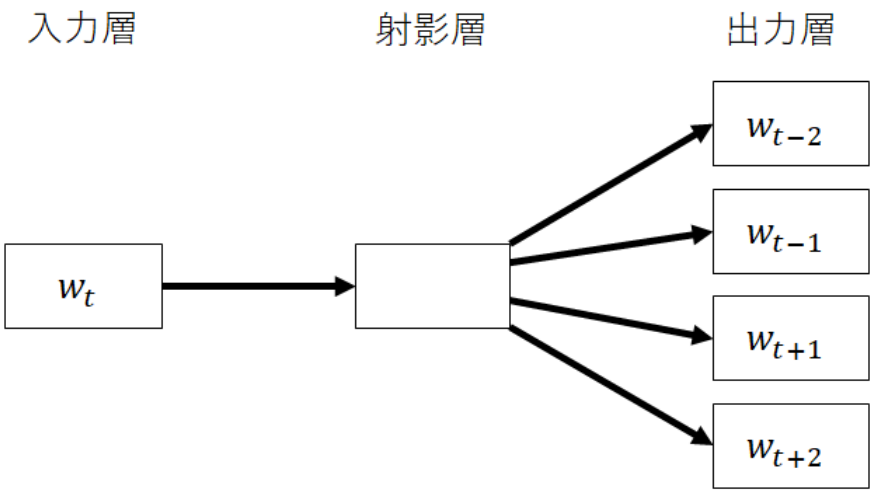

図 3.2　Skip-gram モデル

Skip-gram モデルは，CBOW モデルとは逆向きの予測を行うモデルである．つまり，ニューラルネットワークの入力を w_t の Bag-of-Words とし，その前後の単語 (w_{t-k}, . . . , w_{t-1}, w_{t+1}, . . . , w_{t+k}) を予測することで，Word Vector を学習していく．また，CBOW モデルと比較して動作は遅いが，低頻出の単語に強く予測精度が高いとされている [8]．

3.2　Paragraph Vector

Paragraph Vector は，Le ら [9] によって提案された，文章の意味表現をベクトル化する技術である．Word Vector では，文章中に現れる単語の意味をそれぞれ意味的にベクトル化を行っていた．Paragraph Vector は，それを文章にまで拡張させたものであり，文章全体が 1 つのベクトル値を持つことになる．3.2.1 および 3.2.2 では，Paragraph Vector を作成するための 2 つのモデルについて述べる．

3.2.1　PV-DM モデル

PV-DM(Distributed Memory Model of Paragraph Vector) モデルを図 3.3 に示す．

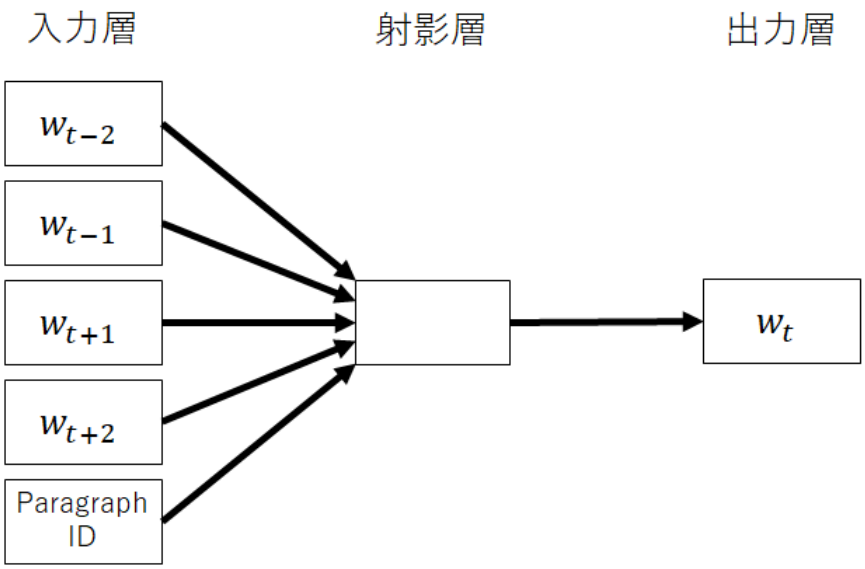

図 3.3　PV-DM モデル

基本的なアルゴリズムは CBOW モデルと同様である．CBOW モデルと異なる点は，入力層に Paragraph ID が追加されているということである．Paragraph ID とは，その文章の Paragraph Vector の値そのものである．PV-DM モデルで作成される Paragraph Vector は，単語の本質的な意味や，語順を考慮するという特徴がある．Paragraph Vector は，以下の手順で求められる．

1. 既存の文書から Word Vector，重み，Paragraph Vector を学習する．
2. 新規の文書に対して，Word Vector と重みの値を固定した上で，Paragraph Vector を求める．

このような手順で十分に学習が行われた Paragraph Vector は，その文章の特徴量として扱うことが可能である．

3.2.2　PV-DBOW モデル

PV-DBOW(Distributed Bag of Words of Paragraph Vector) モデルを図 3.4 に示す．

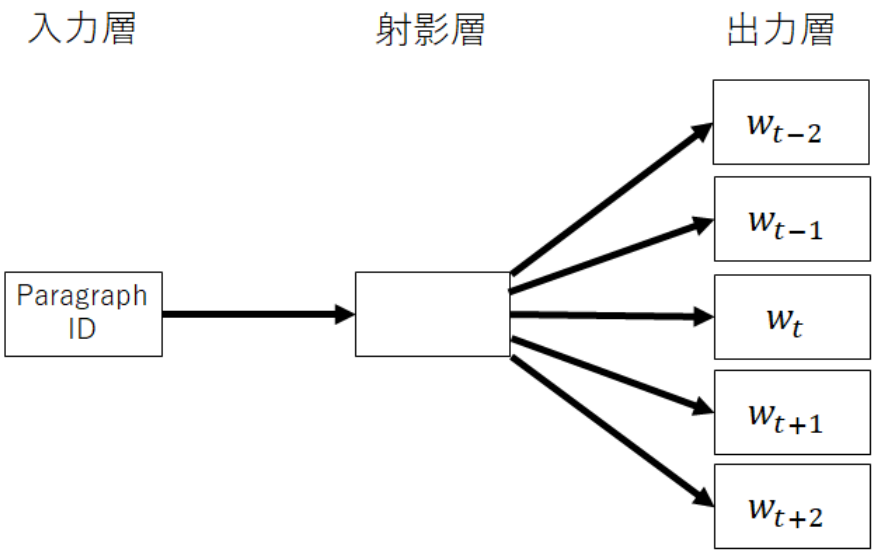

図 3.4　PV-DBOW モデル

PV-DBOW モデルは，Skip-gram モデルと類似しており，入力が Word Vector ではなく Paragraph ID となっている．PV-DM モデルでは，Paragraph Vector の学習を行うために，重みと Word Vector の両方のパラメータを必要としたが，PV-DBOW では重みのみを必要とするため，より高速に動作する．しかしながら，語順を考慮しないモデルとなっているため，一般的に PV-DM モデルより予測精度は劣るとされている．

3.3　SVM

SVM(サポートベクターマシン) とは，特徴量をもつベクトルに対して，分類を行う機械学習アルゴリズムの 1 つである．SVM について，図 3.5 を用いて説明する．図 3.5 のように，赤のような特徴を持つクラス (ベクトル群) と青のような特徴を持つクラスの分類を行うとする．これらの分類を行うために，赤と青のクラスを分ける明確な境界線を引く必要がある．この境界線の引き方は無数に存在するが，SVM ではマージン最大化という手法を用いて境界線を引く．マージンとは，境界から最も近いベクトルからの距離のことを表す．図 3.5 の例では，赤と青のクラスの内，境界線から最も近いベクトルを 2 点ずつ選び，そのマージンを最大化するように境界線を引いている．この選ばれたベクトルのことをサポートベクトルと呼ぶ．要するに，赤のクラスと青のクラスから最も距離が遠い境界線を引く．境界線を定めると，分類器と呼ばれるものが完成し，新たなベクトルデータを追加した際に，

赤と青のどちらのクラスに所属するかを判定することが可能となる．

今回の例では，2 つの特徴量を持ったベクトル，つまり 2 次元ベクトルを扱ったため，境界には直線を用いたが，3 次元になると面を扱う．この次元数を一般化し，任意の次元に対して求まる境界の事を「超平面」と呼ぶ．

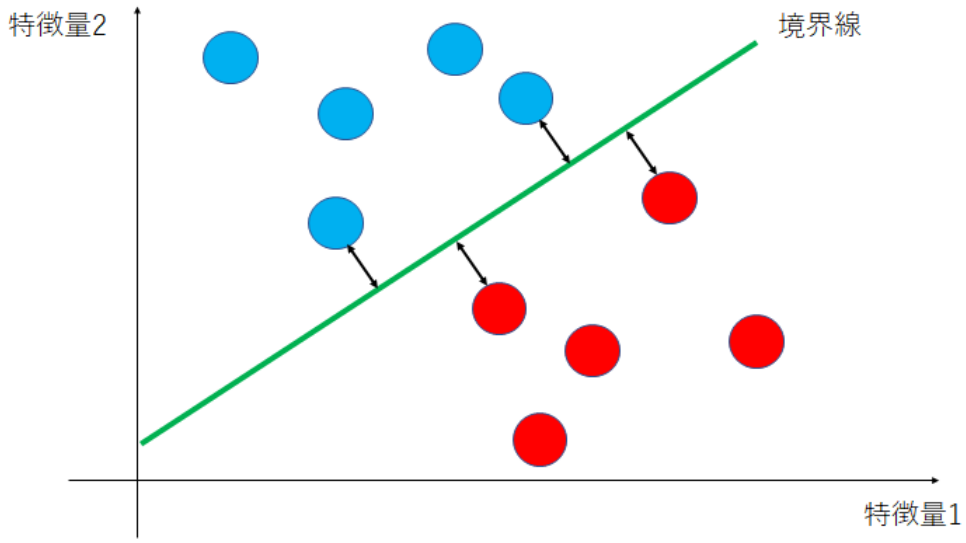

図 3.5　SVM の例

第4章　提案手法と実験

4.1　マルウェア亜種分類の提案手法

マルウェアの動的解析ログから API コール列を取得し，API コール列に対する Paragraph Vector を作成する．この Paragraph Vector を特徴量として，機械学習によりマルウェアの亜種分類を行う．本研究では，PV-DM モデルおよび PV-DBOW モデルを用いて Paragraph Vector を作成し，それぞれのモデルがマルウェアの亜種分類の精度に，どのように影響を与えるか比較を行う．つまり，マルウェアの亜種分類の提案手法を示すと以下のようになる．

提案手法: 機械学習を用いたマルウェア亜種分類

入力: 学習を行う検体 $a_1, \ldots, a_n$ マルウェアファミリ b

出力: 検体がマルウェアファミリかどうか分類できる SVM

方法:

1. SVM に学習させるためのデータセット $d_i = (API(a_i), c_i), i = 1, \ldots, n$ を作成する．ここで $API(a_i)$ は検体 a_i の動的解析結果から得られる API コール列を表し，検体 a_i がマルウェアファミリ b の検体ならラベル $c_i = 1$，検体でないならラベル $c_i = -1$ とする．
2. 各検体について $API(a_i)$ から Doc2Vec を用いて特徴ベクトルを作成する．
3. 特徴ベクトルとラベル c_i を用いて SVM で二値分類器を作成する．

22 種類のファミリがあり，Doc2Vec には 2 つのモデルがあるため，44 種類の SVM を用意する．この手法に関して，4.2 節では具体的な実装について述べる．

4.2　提案手法の評価方法

本研究では，Python 言語によって記述したプログラムを用いて実験を行い，その実験結果の評価を行う．表 4.1 にその実験環境を示す．

表 4.1　実験環境

CPU	Intel Core i7-7700 3.60[GHz]
OS	Windows 10 Pro
メモリ	16.0[GB]
インタプリタ	Python 3.9.1

また，本研究におけるマルウェアの名称は，Kaspersky 社の命名規則のものを使用する [10]．Kaspersky の命名規則を以下に示す．

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

Prefix および Variant は必須部分ではなく，存在しないマルウェアも存在する．Prefix はマルウェアを検知したサブシステムを表し，Variant は亜種を示す．Behaviour はそのマルウェアの動作を示し，Platform は実行された環境，Name はマルウェアのファミリ名を示す．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する．

4.2.1　API コール列の作成

FFRI Dataset 2013〜2016 から API コール列を取得し，Paragraph Vector に変換するためのデータを作成する．API コール列を取得するに当たって，以下の条件に当てはまるマルウェアを実験対象とする．

- 読み込み可能である JSON ファイルであるもの．
- “Kaspersky”の項目が存在し，かつ Kaspersky の命名規則に当てはまるもの．
- “Behavior”の項目が存在し，ログに API コールが 1 つ以上記述されているもの．
- ファミリに含まれるマルウェア亜種の検体数が 100 以上であるもの．
- Name が Generic ではないもの．

これらの条件に当てはまるマルウェアを，ファミリごとに集計した結果を表 4.2 に示す．

表 4.2　データセットに含まれるマルウェアのファミリ名と検体数

ID	検体名	検体数
1	Trojan.Win32.Waldek	687
2	Trojan-Spy.Win32.Zbot	581
3	Trojan.Win32.Yakes	577
4	Backdoor.Win32.Androm	445
5	Trojan.Win32.Agent	383
6	Trojan.Win32.Inject	351
7	Worm.Win32.WBNA	308
8	Hoax.Win32.ArchSMS	304
9	Trojan-PSW.Win32.Fareit	272
10	Trojan-PSW.Win32.Tepfer	218
11	Backdoor.Win32.DarkKomet	206
12	Trojan-Ransom.Win32.Foreign	204
13	Trojan-Dropper.Win32.Injector	185
14	Trojan.Win32.Jorik	171
15	Packed.Win32.Tpyn	163
16	Trojan.Win32.Kovter	145
17	Trojan.Win32.Scar	127
18	Downloader.Win32.LMN	123
19	Worm.Win32.Vobfus	120
20	Backdoor.Win32.Matsnu	110
21	Trojan-Downloader.Win32.Upatre	108
22	Trojan.Win32.Llac	100

これら合計 5888 検体，22 ファミリを本実験の対象とする．そして，それぞれの検体から API コール列を取得し，空白を区切り文字として API コール列を表現する．例えば，図 4.1 は検体“Trojan.Win32.Jorik.Vobfus.fqzm”から得られた API コール列である．

LdrLoadDll NtOpenKey NtQueryValueKey NtClose LdrLoadDll LdrGetProcedureAddress
LdrGetProcedureAddress . . . NtClose NtWriteFile NtClose CreateProcessInternalW NtClose

図 4.1　API コール列の例

4.2.2　Paragraph Vector の作成

データセットから取得した API コール列を用いて，Paragraph Vector の作成を行う．本実験では，Paragraph Vector を作成するツールとして，多くの企業で用いられている gensim [12] の Doc2Vec を用いる．Paragraph Vector のモデルには PV-DM モデルと PV-DBOW モデルがある．Doc2Vec でもこれらの 2 つのモデルを使って Paragraph Vector を計算することができる．Paragraph Vector は 100 次元で作成する．図 4.1 の API コール列を Paragraph Vector(PV-DM モデル) に変換したものを図 4.2 に示す．

Trojan.Win32.Jorik.Vobfus.fqzm -0.553390 -0.034782 1.156945 0.869348 0.036302
-0.127194 -0.671012 . . . -0.106845 0.274399 0.221303 0.468480 0.511919

図 4.2　API コール列を元に作成した Paragraph Vector

この API コール列を数値化した Paragraph Vector を用いて 4.2.3 節では亜種の判定を行っていく．

4.2.3　亜種分類のためのデータセットの作成と亜種分類の評価方法

SVM で亜種判定を行うために，SVM による学習器を作成する．その学習器を作成するためにデータセットが必要になる．SVM は入力とする検体があるファミリに含まれるか含まれないかの二値分類を行う．22 ファミリには検体が 5888 検体あり，判定したいファミリとそれ以外のファミリについて，それぞれの比率が 21：1 になるように検体をサンプリングする．今回，22 ファミリあるので，22 種類の SVM を使った各ファミリに関する分類器を作成する．21：1 にサンプリングするための手順は以下のようになる．

1. 判定したいファミリから 21 検体をランダムに選出する．

2. それ以外のファミリそれぞれから 1 検体ずつランダムに選出する．

3. 上記いずれかが選出できない場合，サンプリングを終了する．

4. 選出した 42 検体の Paragraph Vector を，判定したいファミリのデータセットに加える．

5. 上の手順を繰り返す．

この作業を各ファミリごとに行い，データセットを 22 種類作成する．

次に，サンプリングしたデータにラベリングを行う．これは，判定したいファミリの検体の Paragraph Vector であるか，それ以外のファミリの検体の Paragraph Vector であるかどうかを表す．前者であれば 1 を，後者であれば-1 のラベルを付加する．

以上の操作を 2 つの Paragraph Vector のモデルに対して同様に行うので，結果として 44 種類の SVM が得られる．

4.2.4　亜種分類の評価

本実験では，SVM の実装として LIBSVM [13] を用いる．ラベルを付け加えたデータセットを用いて SVM を使って分類器を作成する．SVM を用いた分類精度の評価方法として，本研究では，4 分割交差検証により学習と検証を行う．4 分割交差検証では，4 回の検証を行い判定精度を測定する．その平均値を判定精度の測定結果とする．なお，SVM の学習において必要となるパラメータは，グリッドサーチによるパラメータチューニングにより求めた最適な値を用いた．

4.3　評価実験結果

Paragraph Vector の各モデルを用いたデータセットに対して，SVM によるマルウェアの亜種判定を行った判定精度の結果を表 4.3 に示す．

表 4.3　モデルごとの SVM による精度 (単位:%)

ファミリ名 \ モデル	佐藤らの研究結果	Doc2Vec	
		PV-DM	PV-DBOW
Trojan.Win32.Waldek	91.59	97.47	99.03
Trojan-Spy.Win32.Zbot	74.80	96.79	97.89
Trojan.Win32.Yakes	81.25	95.78	98.07
Backdoor.Win32.Androm	79.88	96.19	97.38
Trojan.Win32.Agent	72.02	93.91	97.88
Trojan.Win32.Inject	39.64	94.34	98.51
Worm.Win32.WBNA	91.87	96.08	98.97
Hoax.Win32.ArchSMS	94.84	97.10	97.95
Trojan-PSW.Win32.Fareit	78.37	95.43	98.41
Trojan-PSW.Win32.Tepfer	75.60	94.04	97.38
Backdoor.Win32.DarkKomet	81.25	95.23	97.91
Trojan-Ransom.Win32.Foreign	86.31	92.85	96.72
Trojan-Dropper.Win32.Injector	70.24	94.04	98.51
Trojan.Win32.Jorik	90.48	95.53	98.80
Packed.Win32.Tpyn	79.76	88.49	94.04
Trojan.Win32.Kovter	90.48	98.01	99.20
Trojan.Win32.Scar	75.00	87.30	94.84
Downloader.Win32.LMN	89.29	97.02	96.42
Worm.Win32.Vobfus	94.05	94.64	98.80
Backdoor.Win32.Matsnu	-	92.85	98.21
Trojan-Downloader.Win32.Upatre	76.79	89.88	95.23
Trojan.Win32.Llac	76.16	94.04	95.83
平均値	84.69	94.40	97.54
最高値	97.78	98.01	99.20
最低値	69.94	87.30	94.04

本研究で提案したどのモデルにおいても，平均 90 %以上の精度で判定できている．個別

にみれば 99 %以上といった高い精度で判定できているファミリも複数存在する．

既存研究の結果である sentence2vec と本研究の結果を比較すると，全体的に Doc2Vec の方がより良い精度となっている．更に，本研究の各モデルを比較すると，PV-DM モデルよりも PV-DBOW モデルの方が平均値として判定精度が高くなっている．これは，周辺の API コールから 1 つの API コールを予測するよりも，API コールまたは API コール列全体の Paragraph Vector から周辺の API コールを予測する方がマルウェアの特徴を表現する事に適していることを示している．つまり，マルウェアファミリの特徴を表すには，API コールの順序よりもその API コールの種類や登場回数 (Bag-of-Words) の方が重要であることが考えられる．

この結果を亜種判定のさらなる精度向上にどのように利用していくかは今後の課題である．

第5章　結論

本研究ではマルウェアの亜種分類のために，Paragraph Vector の実装である Doc2Vec を用いた手法を提案する．既存研究で示された結果と比較し高精度な結果が得られた．また，Paragraph Vector の PV-DM モデルおよび PV-DBOW モデルで約 95 %といった高い精度の結果が得られることがわかった．

モデルについて比較すると，PV-DBOW モデルの方が良い精度の結果が得られた．これは API コールの種類や登場回数が精度の結果に影響していることを示す．この結果をマルウェアの分類精度の向上にどのように利用していくかについては今後の課題である．

謝辞

本研究を行うにあたり，常日頃より懇切丁寧にご指導頂いた高橋寛教授，甲斐博准教授，王森レイ講師に心より感謝致します．また，日頃より励ましを頂きました研究室の大学院生および 4 年生に御礼申し上げます．

参考文献

［1］ McAfee Labs 脅威レポート，https://www.mcafee.com/enterprise/ja-jp/assets/reports/rp-quarterly-threats-nov-2020.pdf.

［2］ 佐藤拓未，後藤滋樹，武部嵩礼，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 論文集，2016 巻，2 号，pp.298-304，2016.

［3］ 八木毅，青木一史，秋山満昭，幾世知範，髙田雄太，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］ マルウェア対策研究人材育成ワークショップ 2020，https://www.iwsec.org/mws/2020 (参照 2021-1-14).

［5］ FFRI Dataset 2020 のご紹介, http://www.iwsec.org/mws/2020/files/FFRI_Dataset_2020.pdf (参照 2021-1-14).

［6］ FFRI Dataset 2017 のご紹介, https://www.iwsec.org/mws/2017/20170606/FFRI_Dataset_2017.pdf (参照 2021-1-14).

［7］ Tomas Mikolov，Kai Chen，Greg Corrado，Jeffrey Dean，Efficient Estimation of Word Representations in Vector Space，Proceedings of Workshop at ICLR，pp.1-12 (2013).

［8］ word2vec，https://code.google.com/archive/p/word2vec (参照 2021-1-12).

［9］ Quoc Le，Tomas Mikolov，Distributed Representations of Sentences and Documents，Proceedings of the 31th International Conference on Machine Learning，pp.1188-1196 (2014).

［10］ Rules for naming，https://encyclopedia.kaspersky.com/knowledge/rules-for-naming (参照 2021-1-19).

［11］ klb3713/sentence2vec，https://github.com/klb3713/sentence2vec.

［12］ RaRe-Technologies/gensim，https://github.com/RaRe-Technologies/gensim.

［13］ LIBSVM，https://www.csie.ntu.edu.tw/ cjlin/libsvm.

付 録A　プログラムリスト

リスト A.1　Family.py

```
# マルウェアファミリのクラス
class family:
    def __init__(self):
        self.name = ''          # ファミリ名
        self.samples = []       # 検体のリスト
        self.num = 0            # 検体数
        self.par_vec = []       # Paragraph Vectorのリスト
        self.sampling_count = 0 # サンプリングできる回数
        self.svm_data = []      # SVMに追加するデータリスト
        self.svm_label = []     # SVMに追加するデータリストの教師ラベル
```

リスト A.2　Program.py

```
# メインプログラム
import sys
import random
from copy import *
import glob
import re
import os
import shutil
import openpyxl
import numpy as np
from scipy.sparse import csr_matrix
import Update
import Arrange
import Family
import Check

import gensim
from gensim.models.doc2vec import Doc2Vec
from gensim.models.doc2vec import TaggedDocument

sys.path.append('C:\\Library\\libsvm-3.24\\python\\')
sys.path.append('C:\\Library\\libsvm-3.24\\tools\\')
from svm import *
from svmutil import *
from commonutil import *
from grid import *
```

```
27
28 CROSS_VAL_NUM = 4 # 分割交差検証の分割数
29 SAMPLING_DATASET_PATH = '.\\SamplingDataset'
30 DATASET_PATH = '.\\Dataset'
31
32 def data2vec(family_list):
33   while True:
34     answer = input('Update \'FamilyList.txt\' ?(y/n)')
35     if answer == 'y' or answer == 'n':
36       break
37   # FamilyList.txtの作成
38   if answer == 'y':
39     Update.create_family_name_list()
40
41   family_list = Arrange.arrange_list(family_list)   #
         Genericと100未満のファミリを省いてリスト化
42
43   family_name_list = list() # familyクラスのインスタンスの'name'をリスト化したもの
44   print('family_name_list↓')
45   for fam in family_list:
46     print(fam.name)
47     family_name_list.append(fam.name)
48   print('\n')
49
50   Update.create_dir(family_name_list, SAMPLING_DATASET_PATH)      # 作成されたリストから
       、ファミリのフォルダを作成する
51   Update.create_dir(family_name_list, DATASET_PATH)
52
53   while True:
54     answer = input('Update Dataset list and \'SampleList.txt\' ?(y/n)')
55     if answer == 'y' or answer == 'n':
56       break
57
58   # 使用する検体をリスト化したファイルを作成
59   if answer == 'y':
60     for name in family_name_list:
61       if name == '' and is_empty == True:
62         print('\'family_name_list\' is empty.', file = sys.stderr)
63         sys.exit(1)
64       else:
65         is_empty = False
66
67       name = DATASET_PATH + '\\' + name
68
69       # ファミリのディレクトリ初期化
70       shutil.rmtree(name)
71       os.mkdir(name)
72
73     Update.create_API_call_file(family_name_list)
74
75   family_list = Arrange.assign_samples(family_list, 'SampleList.txt')   # 検体を
```

```
        familyインスタンスに割り当て
76
77    while True:
78      answer = input('Update Paragraph Vector?(y/n)')
79      if answer == 'y' or answer == 'n':
80        break
81
82    # Doc2Vecの処理
83    if answer == 'y':
84      documents = []
85      for fam in family_list:
86        for index in range(0, fam.num):
87          with open(DATASET_PATH + '\\' + fam.name + '\\' + str(index) +'.txt', 'r') as f:
88            sentence = f.read()
89          documents.append(TaggedDocument(words = sentence, tags = [fam.name + '_' + str(
                index)]))
90      print('Documents have been created.')
91
92      print('Calculating Paragraph Vector...')
93      model = Doc2Vec(documents, dm=1, vector_size=100, window=5, min_count=1)
94      model.save('doc2vec.model')
95    else:
96      model = Doc2Vec.load('doc2vec.model')
97
98    #print(model.docvecs['Trojan.Win32.Waldek_0'])      # [value1 value2 value3 ...
          value100]
99    #print(type(model.docvecs['Trojan.Win32.Waldek_0']))  # <class 'numpy.ndarray'>
100
101   while True:
102     answer = input('Doc2Vec has completed. Continue?(y/n)')
103     if answer == 'y' or answer == 'n':
104       break
105   if answer == 'n':
106     sys.exit(0)
107
108   family_list = Arrange.assign_parvec(family_list, model)
109
110   return family_list
111
112 def sampling_data(family_list):
113   rate = len(family_list) - 1   # 1つのファミリとそれ以外のファミリの比率(例.ファミリ
          が22種類→21:1の21を指す)
114   print('rate : ' + str(rate) + ':1')
115
116   # ファミリごとにデータセット作成
117   for fam in family_list:
118     # 以下で21n個のサンプルを取る場合のnを求める
119     tmp_fam_list = deepcopy(family_list)  # コピー
120     is_empty = False
121     is_empty, tmp_fam_list = Check.check_more_rate(is_empty, rate, fam, tmp_fam_list)
122     while(is_empty == False):
```

```
123         fam.sampling_count += 1
124         is_empty, tmp_fam_list = Check.check_more_rate(is_empty, rate, fam, tmp_fam_list)
125
126       # k分割交差検証でkで割れる値にするため、(rate+1) * n = kの倍数の形にする
127       while((rate + 1) * fam.sampling_count % CROSS_VAL_NUM != 0):
128         fam.sampling_count -= 1
129
130       n = fam.sampling_count
131       print(fam.name)
132       print( 'num:' + str(fam.num) + ' / rate:' + str(rate) + ' -> ' + 'sampling count : '
              + str(fam.sampling_count))
133
134       # 対象のファミリの検体をランダムに抽出
135       index_list = random.sample(range(0, fam.num), rate * n)
136       par_vec_list = []
137       for index in index_list:
138         par_vec_list.append(fam.par_vec[index])
139
140       # 対象以外のファミリの検体をランダムに抽出
141       for other in family_list:
142         index_list = []
143         if not fam.name == other.name:
144           index_list = random.sample(range(0, other.num), n)
145           for index in index_list:
146             par_vec_list.append(other.par_vec[index])
147
148       random.shuffle(par_vec_list)
149
150       for i, name in enumerate(par_vec_list):
151         path = SAMPLING_DATASET_PATH + '\\' + fam.name + '\\' + str(i) + '.txt'
152         with open(path, 'w') as file:
153           file.write(name)
154
155     return family_list, rate
156
157
158     """
159     # 例
160     fam = Family.family()      # インスタンス作成
161     fam.name = 'A.B.C'         # ファミリ名の代入
162     fam.samples.append('A.B.C.D') # リストに検体追加
163     fam.samples.append('A.B.C.D.E') # 〃
164     fam.samples.append('A.B.C.F') # 〃
165     for line in fam.samples:    # 検体リストの表示
166       print(line)
167     """
168
169 # サンプリングしたデータにラベル付けを行い、SVM学習を行う
170 def sendSVM(family_list, rate):
171     # Excelファイルの書き込み準備
172     wb = openpyxl.load_workbook('SVMResult.xlsx')
```

```
173   sheet = wb['Sheet1']
174   sheet_count = 3
175
176   for fam in family_list:
177     fam_path = SAMPLING_DATASET_PATH + '\\' + fam.name + '\\'
178     train_path = fam_path + 'SVMData.train'
179     if os.path.exists(train_path):
180       os.remove(train_path)
181     file_list = glob.glob(fam_path + '*.txt')
182
183     for file_name in file_list:
184       with open(file_name, 'r') as file:
185         lines = file.read().split() # 空白区切りでParagraph Vectorのデータ読み込み
186
187       sample_name = lines.pop(0)
188       family_name = re.search(r"([a-zA-Z0-9-])+\.([a-zA-Z0-9])+\.([a-zA-Z0-9])+",
              sample_name).group()
189
190       # 判定したいファミリのデータなら1、そうでないなら-1のラベルを付加する
191       if family_name == fam.name:
192         t = 1
193       else:
194         t = -1
195
196       lines_float = [float(s) for s in lines]   # 数値はfloat型に変換しておく
197       fam.svm_data.append(lines_float)
198       fam.svm_label.append(t)
199
200     # スケーリング
201     csr = csr_matrix(fam.svm_data)
202     param = csr_find_scale_param(csr)
203     csr = csr_scale(csr, param)
204     svm_data = csr.toarray().tolist()
205
206     with open(train_path, 'a') as data_file:
207       for i, l in enumerate(svm_data):
208         data_file.write(str(fam.svm_label[i]))
209         for index, value in enumerate(l):
210           data_file.write(' ' + str(index+1) + ':' + str(value))
211         data_file.write('\n')
212
213     print('↓↓↓' + fam.name + '↓↓↓')
214     grid_result = find_parameters(train_path, ['-v', str(CROSS_VAL_NUM), '-out',
            train_path + '.out'])
215     print('Result -> rate:{0}, cost:{1}, gamma:{2}'.format(str(grid_result[0]), str(
            grid_result[1]['c']), str(grid_result[1]['g'])))
216     average = (int(grid_result[0] * 100)) / 100.0 # 小数点第3位以下を切り捨て
217     '''
218     all_num = (rate + 1) * fam.sampling_count
219
220     result_list = []  # k分割交差検証のそれぞれの結果を格納し、最後にその平均を求める
```

```
221
222     for set_i in range(0, CROSS_VAL_NUM):
223       train_data = []
224       test_data = []
225       train_label = []
226       test_label = []
227       for i in range(0, all_num):
228         # データの1/kを検証データ、k-1/kを教師データとする
229         if i >= set_i/CROSS_VAL_NUM * all_num and i < (set_i + 1)/CROSS_VAL_NUM *
                all_num:
230           test_data.append(fam.svm_data[i])
231           test_label.append(fam.svm_label[i])
232         else:
233           train_data.append(fam.svm_data[i])
234           train_label.append(fam.svm_label[i])
235
236       # 以下、Libsvmによる計算
237       prob = svm_problem(train_label, train_data)
238       param = svm_parameter('-s 0 -c {0} -g {1}'.format(str(grid_result[1]['c']), str(
              grid_result[1]['g'])))
239
240       m = svm_train(prob, param)
241       result = svm_predict(test_label, test_data, m)
242
243       result_list.append(result[1][0])
244
245     # 交差検証により求めた結果から、平均を算出する
246     average = sum(result_list)/len(result_list)
247     average = (int(average * 100)) / 100.0  # 小数点第3位以下を切り捨て
248     print(f'Average : {average}')
249     '''
250     # Excelファイルに書き込み
251     cell_id = 'B' + str(sheet_count)
252     sheet[cell_id] = average
253     sheet_count += 1
254
255   wb.save('SVMResult.xlsx')
256
257 def main():
258   family_list = list()      # 扱うべきファミリを格納するリスト
259   family_list = data2vec(family_list) # 2013～2016年のDatasetをParagraph
          Vectorに変換するまで
260
261   family_list, rate = sampling_data(family_list)  # 学習器に通すデータセットを作成する
262
263   sendSVM(family_list, rate)    # SVMへデータを送る
264
265 if __name__ == '__main__':
266   main()
```

リスト A.3　Update.py

```
# ファイル等、データを更新する関数を記述
import json
import os
import re
import glob
import collections
import sys
import shutil
import Family

DATASET_PATH = '.\\Dataset'

# ファミリ名を昇順に並べたものを記述したファイルを作成する
def create_family_name_list():
  family_name_list = list()

  count_all = 0
  count_able = 0
  count = 0

  # 2013～2016年のDataset
  for year in range(2013, 2017):
    # ファイル読み込み、辞書の作成
    file_list = glob.glob('FFRI_Dataset_{}//*.json'.format(year))
    for i, name in enumerate(file_list):
      json_open = open(name, 'r')
      print('File Name :', name)

      count_all += 1

      try:
        json_load = json.load(json_open)
      except json.JSONDecodeError:
        json_open.close()
        continue

      # 'behaivior' 及び 'processes' が存在するかの確認
      try:
        processes_range = range(len(json_load['behavior']['processes']))
      except KeyError:
        json_open.close()
        continue

      # apiの確認
      is_api = False
      is_key = True
      for processes_i in processes_range:
        try:
          calls_range = range(len(json_load['behavior']['processes'][processes_i]['calls
              ']))
```

```
50          except KeyError:
51            is_key = False
52            break
53
54          for calls_i in calls_range:
55            if(json_load['behavior']['processes'][processes_i]['calls'][calls_i]['api']):
56              is_api = True
57              count_able += 1
58              break
59
60          if is_api == True:
61            break
62
63        if is_api == False:
64          json_open.close()
65          continue
66
67        # 'Kaspersky' の有無を確認
68        try:
69          sample_name = json_load['virustotal']['scans']['Kaspersky']['result']
70        except KeyError:
71          json_open.close()
72          continue
73
74        # 'sample_name' から正規表現に当てはまる部分を取り出す
75        try:
76          family_name = re.search(r"([a-zA-Z0-9-])+\.([a-zA-Z0-9])+\.([a-zA-Z0-9])+",
                sample_name).group()
77        except:
78          json_open.close()
79          continue
80
81        try:
82          family_name_list.append(family_name)
83        except AttributeError:
84          continue
85
86        count += 1
87
88        json_open.close()
89
90    # リストからファミリの検体数を集計し、昇順に並べる
91    c = collections.Counter(family_name_list)
92    sort_list = c.most_common()
93    """
94    # 大文字と小文字を区別しない
95    for i in range(0,len(sort_list)):
96      for j in range(i+1,len(sort_list)):
97        if sort_list[i][0].lower() == sort_list[j][0].lower():
98          l_i = list(sort_list[i])
99          l_j = list(sort_list[j])
```

```
100         l_i[1] += l_j[1]
101         l_j[1] = 0
102
103         t_i = tuple(l_i)
104         t_j = tuple(l_j)
105
106         del sort_list[i]
107         sort_list.insert(i,t_i)
108         del sort_list[j]
109         sort_list.insert(j,t_j)
110   """
111   with open('FamilyList.txt', 'w') as family_file:
112     for i in range(0,len(sort_list)):
113       family_file.write(sort_list[i][0])
114       family_file.write('\n')
115       family_file.write(str(sort_list[i][1]))
116       family_file.write('\n')
117
118   print('Number of ALL Samples:', count_all)
119   print('Number of Available Samples(API Call > 0):', count_able)
120   print('Number of Counted Samples:', count)
121   print('Updated \'FamilyList.txt\'')
122
123 # APIコール列のファイルを作成する
124 def create_API_call_file(family_name_list):
125   count = 0
126   sample_list_file = open('SampleList.txt', 'w')
127
128   # 2013～2016年のDataset
129   for year in range(2013, 2017):
130     # ファイル読み込み、辞書の作成
131     file_list = glob.glob('FFRI_Dataset_{}//*.json'.format(year))
132
133     for i, name in enumerate(file_list):
134       json_open = open(name, 'r')
135       print('File Name :', name)
136
137       try:
138         json_load = json.load(json_open)
139       except json.JSONDecodeError:
140         json_open.close()
141         continue
142
143       # 'Kaspersky' の有無を確認
144       try:
145         sample_name = json_load['virustotal']['scans']['Kaspersky']['result']
146       except KeyError:
147         json_open.close()
148         continue
149
150       # 'family_name' が正規表現に当てはまるかを確認
```

```
151     try:
152       family_name = re.search(r"([a-zA-Z0-9-])+\.([a-zA-Z0-9])+\.([a-zA-Z0-9])+",
              sample_name).group()
153     except:
154       json_open.close()
155       continue
156
157     # family_name_listに存在しない検体については無視
158     try:
159       if not family_name in family_name_list:
160         continue
161       else:
162         print(family_name)
163     except AttributeError:
164       continue
165
166     # 'behaivior' 及び 'processes' が存在するかの確認
167     try:
168       processes_range = range(len(json_load['behavior']['processes']))
169     except KeyError:
170       json_open.close()
171       continue
172
173     # apiの確認
174     is_api = False
175     is_key = True
176     for processes_i in processes_range:
177       try:
178         calls_range = range(len(json_load['behavior']['processes'][processes_i]['calls
              ']))
179       except KeyError:
180         is_key = False
181         break
182
183       for calls_i in calls_range:
184         if(json_load['behavior']['processes'][processes_i]['calls'][calls_i]['api']):
185           is_api = True
186           break
187
188       if is_api == True:
189         break
190
191     if is_key == False:
192       json_open.close()
193       continue
194
195     if is_api == True:
196
197       # ファイルにAPIコール名を書き込み
198       write_lines = ''
199       for processes_i in processes_range:
```

```
200           calls_range = range(len(json_load['behavior']['processes'][processes_i]['calls
                  ']))
201           for calls_i in calls_range:
202             if(json_load['behavior']['processes'][processes_i]['calls'][calls_i]['api'
                    ]):
203               write_lines += json_load['behavior']['processes'][processes_i]['calls'][
                      calls_i]['api']
204               write_lines += ' '
205
206         if not write_lines == '':
207           file_list = glob.glob(DATASET_PATH + '\\' + family_name + '\\*.txt')
208           with open(DATASET_PATH + '\\' + family_name + '\\' + str(len(file_list)) + '.
                  txt', 'w') as API_call_file:
209             API_call_file.write(write_lines[:-1])
210
211
212         sample_list_file.write(sample_name + '\n')
213
214         count += 1
215
216         """
217         try:
218           path = '.\\' + format(family_name) + '\\' + sample_name + '.txt'
219         except:
220           continue
221         # 検体数が100以上のファミリでないとファイルパスのエラーが出る
222         try:
223           API_call_file = open(path, 'w')
224         except:
225           continue
226
227         # ファイルにAPIコール名、引数名、引数値を書き込み
228         for processes_i in processes_range:
229           calls_range = range(len(json_load['behavior']['processes'][processes_i]['calls
                  ']))
230           for calls_i in calls_range:
231             if(json_load['behavior']['processes'][processes_i]['calls'][calls_i]['api
                    ']):
232               arg_range = range(len(json_load['behavior']['processes'][processes_i]['
                      calls'][calls_i]['arguments']))
233               API_call_file.write(json_load['behavior']['processes'][processes_i]['calls
                      '][calls_i]['api'])
234               API_call_file.write(' ')
235               for arg_i in arg_range:
236                 API_call_file.write(json_load['behavior']['processes'][processes_i]['
                        calls'][calls_i]['arguments'][arg_i]['name'])
237                 API_call_file.write(' ')
238                 if not json_load['behavior']['processes'][processes_i]['calls'][calls_i
                        ]['arguments'][arg_i]['value'] is None:
239                   API_call_file.write(json_load['behavior']['processes'][processes_i]['
                          calls'][calls_i]['arguments'][arg_i]['value'])
```

```
240                   API_call_file.write(' ')
241                 else:
242                   API_call_file.write('NULL')
243                   API_call_file.write(' ')
244
245         API_call_file.close()
246         """
247
248       json_open.close()
249
250   sample_list_file.close()
251   print('\n')
252   print('Number of Counted Samples:', count)
253   print('Updated \'APICallSeq.txt\' and \'SampleList.txt\'\n')
254
255 # 与えられたリストから、その名前のフォルダを作成する
256 def create_dir(family_name_list, dir = '.'):
257   is_empty = True
258
259   for name in family_name_list:
260     if name == '' and is_empty == True:
261       print('\'family_name_list\' is empty.', file = sys.stderr)
262       sys.exit(1)
263     else:
264       is_empty = False
265
266     name = dir + '\\' + name
267
268     # ファミリのディレクトリ作成(初期化)
269     if not os.path.exists(name):
270       os.mkdir(name)
```

リスト A.4　Arange.py

```
1  # データを整える関数を記述
2  import sys
3  import shutil
4  import re
5  import Family
6
7  # 必要なファミリをリスト化
8  def arrange_list(family_list):
9    with open('FamilyList.txt', 'r') as file:
10     name = file.readline()
11     num = file.readline()
12
13     if name == '' or num == '':
14       print('Family including 100 samples or more does NOT exist in \'FamilyList.txt\'.'
             , file = sys.stderr)
15       sys.exit(1)
16
```

```
17      while int(num) >= 100:
18        if name == '' or num == '':
19          break
20
21        # Genericを含むものは省く
22        if not 'Generic' in name:
23          fam = Family.family()
24          fam.name = name.rstrip('\n')
25          family_list.append(fam)
26
27        name = file.readline()
28        num = file.readline()
29
30    return family_list
31
32  # 検体ファイルとfamilyインスタンスのリストを用いて、検体をファミリごとに割り当てる
33  def assign_samples(family_list, sample_file_name):
34    # 検体のリストを読み込み
35    with open(sample_file_name) as file:
36      sample_list = file.read().splitlines()
37      print(sample_file_name + ' loaded (number of samples : ' + str(len(sample_list)) + '
          )')
38      print('(number of family_list : ' + str(len(family_list)) + ')')
39
40    for i, sample_name in enumerate(sample_list):
41      # 'sample_name' から正規表現に当てはまる部分を取り出す
42      family_name = re.search(r"([a-zA-Z0-9-])+\.([a-zA-Z0-9])+\.([a-zA-Z0-9])+",
            sample_name).group()
43
44      for fam in family_list:
45        if fam.name == family_name:
46          fam.samples.append(sample_name)
47          print(fam.samples[-1] + '->' + fam.name)
48          break
49
50    for fam in family_list:
51      fam.num = len(fam.samples)
52      print(fam.name + ':' + str(fam.num))
53
54    print('\n')
55    print('Malware samples have assigned to family instances.')
56    print('\n')
57
58    return family_list
59
60  # 'sentX'を検体名に置換
61  def arrange_parvec(vec_file_name, sample_file_name):
62    # バックアップの作成
63    back_file = vec_file_name + '.bak'
64    shutil.copy(vec_file_name, back_file)
65
```

```
66    # 検体のリストを読み込み
67    with open(sample_file_name) as file:
68      sample_list = file.read().splitlines()
69      print(sample_file_name + ' loaded')
70
71    # vecファイルの読み込み
72    with open(vec_file_name, 'r') as vfile:
73      vec_lines = vfile.read()
74      print(vec_file_name + ' loaded')
75
76    sent_count = 0
77    for name in sample_list:
78      sent = 'sent_' + str(sent_count)
79      if sent in vec_lines:
80        vec_lines = vec_lines.replace(sent, name, 1)
81        print('sent_' + str(sent_count) + 'replaced with' + name)
82
83      sent_count += 1
84
85    with open(vec_file_name, 'w') as file:
86      file.write(vec_lines)
87
88  # modelのParagraph Vectorの値をファミリの属性に追加
89  def assign_parvec(family_list, model):
90    for fam in family_list:
91      for i in range(0, fam.num):
92        tag = fam.name + '_' + str(i)
93        data = tag
94        for value in model.docvecs[tag]:
95          data += ' ' + str(value)
96
97        fam.par_vec.append(data)
98
99    return family_list
```

リスト A.5　Check.py

```
1   # データが適した状態かどうかをチェックする関数
2
3   # 対象ファミリはrate以上、それ以外のファミリが1以上検体がリストに存在するかどうかを確認
4   def check_more_rate(flag, rate, target, tmp_fam_list):
5
6       for tmp_fam in tmp_fam_list:
7           if tmp_fam.name == target.name:
8               if tmp_fam.num < rate:
9                   flag = True
10                  return flag, tmp_fam_list
11              else:
12                  tmp_fam.num -= rate
13          else:
14              if tmp_fam.num < 1:
```

```
15                 flag = True
16                 return flag, tmp_fam_list
17             else:
18                 tmp_fam.num -= 1
19
20     flag = False
21     return flag, tmp_fam_list
```