2017年度 修士論文

Paragraph Vectorによる
悪性JavaScriptの検知法

提出日：2018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：5116F031-7

喜入　亮

目次

第 1 章　序論 **5**

1.1　研究の背景 5

1.2　研究の目的 6

1.3　論文の構成 6

第 2 章　Paragraph Vector **7**

2.1　Bag of Words の考え方 7

2.2　Word Vector の概要 7

2.2.1　CBoW 8

2.2.2　skip-gram 9

2.3　Paragraph Vector の概要 9

2.3.1　PV-DM 10

2.3.2　PV-DBoW 10

第 3 章　関連研究 **12**

3.1　機械学習を用いた悪性 JavaScript 検知 12

3.2　ディープラーニングを用いた研究 13

第 4 章　提案手法 **14**

4.1　提案手法の概要 14

4.2　手順 1: 特徴ベクトルの作成 14

4.3　手順 2: JavaScript の悪性判定 16

第 5 章　実証実験 **17**

5.1　実験に用いるデータ 17

5.1.1 悪性データセット 17
5.1.2 良性データセット 18
5.2 評価指標 18
5.3 実験手順 19
5.4 実験結果 20
5.4.1 SVM のパラメータ 20
5.4.2 epoch 20
5.4.3 window 21
5.4.4 size 22
5.4.5 結果 25
5.5 考察 25

第 6 章 結論 **27**

6.1 まとめ 27
6.2 今後の課題 27
6.2.1 ソースコードの整形方法の工夫 27
6.2.2 さまざまなデータでの検証 28

謝辞 **29**

参考文献 **30**

図一覧

2.1 CBoW モデル 8
2.2 skip-gram モデル 9
2.3 PV-DM モデル 10
2.4 PV-DBoW モデル 11

4.1 単語分割の例 15
4.2 単語置換の例 15

5.1 epoch 数の選定 21
5.2 window の選定 23
5.3 size の選定 24
5.4 誤検知の例 26

表一覧

5.1 混同行列 18
5.2 SVM パラメータ 20
5.3 epoch 選定に用いるパラメータ 20
5.4 window 選定に用いるパラメータ 22
5.5 size 選定に用いるパラメータ 22
5.6 最適な Pragraph Vector のパラメータ 25
5.7 実験結果 25

第 1 章

序論

1.1 研究の背景

Drive-by Download 攻撃 [1] やフィッシングサイトなどの, Web ページを介して行われるサイバー攻撃の被害は後を絶たない [2]. 特に, Drive-by Download 攻撃により被害を受けるランサムウェア (Ransomeware) の脅威は拡大している [3].

一般に, Web ページを介したサイバー攻撃はいくつかのステップからなる. 被害者を攻撃者の用意したサーバへ導く中継ステップ, Web ブラウザやプラグインの脆弱性を調べるステップ, 最終的にマルウェアが配布されるステップ, といった 3 ステップが主な攻撃の段階である. このうち前半の 2 つのステップでは JavaScript を利用している場合が多く, これらの多くは攻撃検知を避ける目的で難読化が施されている.

Drive-by Download 攻撃の多くは, Exploit Kit と呼ばれるツールを用いて行われている. Exploit Kit はサイバー攻撃を行うことができるテンプレートのようなものである. Exploit Kit を用いることで専門的な知識がなくとも攻撃を行うことができ, アンダーグラウンドではサービスとしても販売されている [4]. Exploit Kit により作成される攻撃コードには類似性があり, この特徴を元にした研究も多く存在する. 筆者 [5] は, こうした特徴に基づき Drive-by Download 攻撃を検知する研究を行った.

1.2　研究の目的

1.1 節でも述べたように, 攻撃コードの特徴に基づいた研究は数多く存在する．第 3 章で詳しく説明するが, その中でも JavaScript に注目した機械学習を用いた研究は既に多く存在する．しかし既存の機械学習を用いた手法では, 特徴量を人間が選定しているものが多い.

本研究では, 人間による解析を必要としない自動的な特徴量の選定方法を提案する．具体的には, Deep Learning による自然言語の解析手法の一つである Paragraph Vector を用いて, JavaScript のソースコードの特徴を自動的に計算する．得られた特徴を用いて JavaScript の悪性判定を行うことで, Paragraph Vector を用いた JavaScript の特徴表現が有効であることを示す.

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

本論文の概要について述べる.

第 2 章 Paragraph Vector

本論文で用いる Paragraph Vector 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説明する.

第 4 章 提案手法

本論文の提案手法を述べる.

第 5 章 性能評価

提案手法の性能評価と考察を行う.

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, 本論文で用いる Paragraph Vector について説明する. まずは関連した技術である Bag of Words を第 2.1 節, Word Vector を第 2.2 節で説明する. その後, Paragrapg Vector について第 2.3 説で述べる.

2.1 Bag of Words の考え方

ドキュメントをベクトル表現する手法として, 従来 Bag of Words が広く用いられてきた. Bag of Words とは, 文章に出現する単語の出現を要素とするベクトル表現である [6]. ベクトルの次元数は語彙の数と一致する.

例えば A, B, C, D の 4 語のみが存在する文章を考える. 語彙数は 4 であるため, ベクトルの次元数も 4 となる. このとき, 単語 B は (0, 1, 0, 0) と表せる. この表現方法を 1-of-K 符号化と呼ぶ. Bag of Words は, 1-of-K 符号化したベクトルを加算したベクトルで表現される. “A, C, A, D” という文は, (2, 0, 1, 1) というベクトルとして表せる. このように単純な表現で方法で文章のベクトルが得られるが, このベクトル表現では単語の意味や出現順序を考慮しない.

2.2 Word Vector の概要

Word Vector は Mikolov [7] によって提案された, 単語のベクトル表現を得る手法である. この手法では文脈によって単語のベクトル表現が決定されるため, 意味を考慮したベクトルが得

られる. Word Vectorでは, ベクトル表現を得るためのアルゴリズムとしてCBoW (Continuons Bag of Words) と skip-gram の 2 種類が存在する.

2.2.1　CBoW

CBoW モデルを図 2.1 に示す.

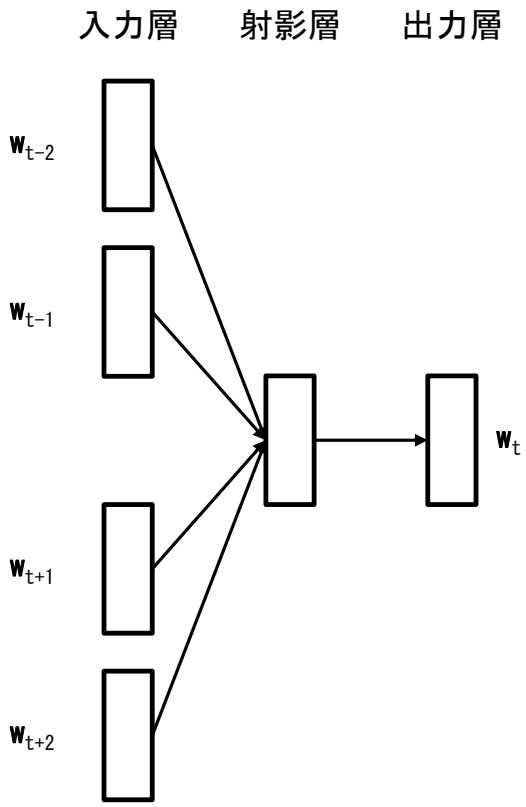


図 2.1: CBoW モデル

CBoW モデルでは, ベクトル化する単語 w_t の前後 2k 語 $(w_{t-k},\cdots,w_{t-1},w_{t+1},\cdots,w_{t+k})$ の単語を文脈と呼び, 文脈の Bag of Words 表現を入力とする. 図 2.1 は $k=2$ の例である. そして, 文脈から予測される中心の単語と実際の単語 w_t が一致するようにニューラルネットワークを訓練する. 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, 出力層へ伝達する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射影層を用いる. 学習の結果得られた射影層の値を単語の Word Vector 表現とする.

2.2.2　skip-gram

skip-gram モデルを図 2.2 に示す.

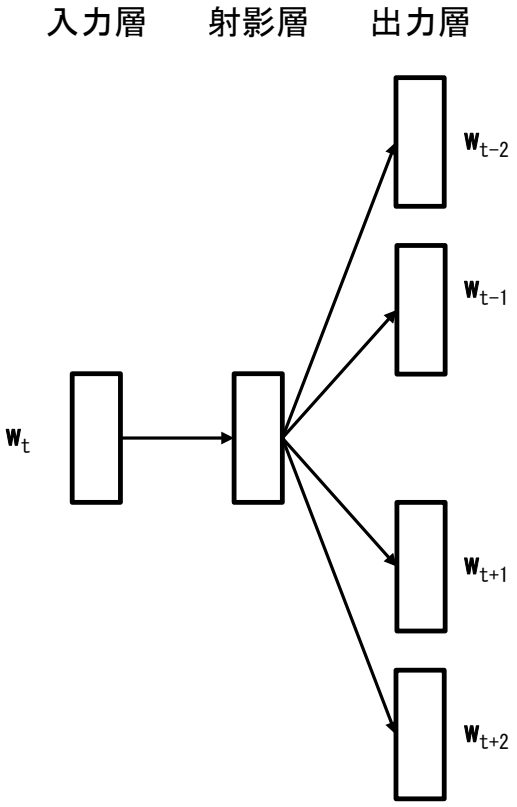


図 2.2: skip-gram モデル

skip-gram モデルは CBoW モデルと逆向きの構造であり, ベクトル化する単語 w_t を入力として, その前後の 2k 語 $(w_{t-k},\cdots,w_{t-1},w_{t+1},\cdots,w_{t+k})$ を出力する. w_t から予測される単語を実際の周辺の単語に近くなるようニューラルネットワークを訓練する. その結果, 得られた射影層の値を単語の Word Vector 表現とする.

2.3　Paragraph Vector の概要

Paragraph Vector は Le [8] によって提案された, 文章のベクトル表現を得る手法である. Word Vector では単語のベクトルを表現していたのに対して, Paragraph Vector は文章全体をベクトルで表現できるよう拡張されている. Paragraph Vector では, ベクトル表現を得るためのアル

ゴリズムとしてPV-DM (Distributed Memory Model of Paragraph Vector) とPV-DBoW (Distributed Bag of Words of Paragraph Vector) の 2 種類が存在する.

2.3.1　PV-DM

PV-DM モデルを図 2.3 に示す.

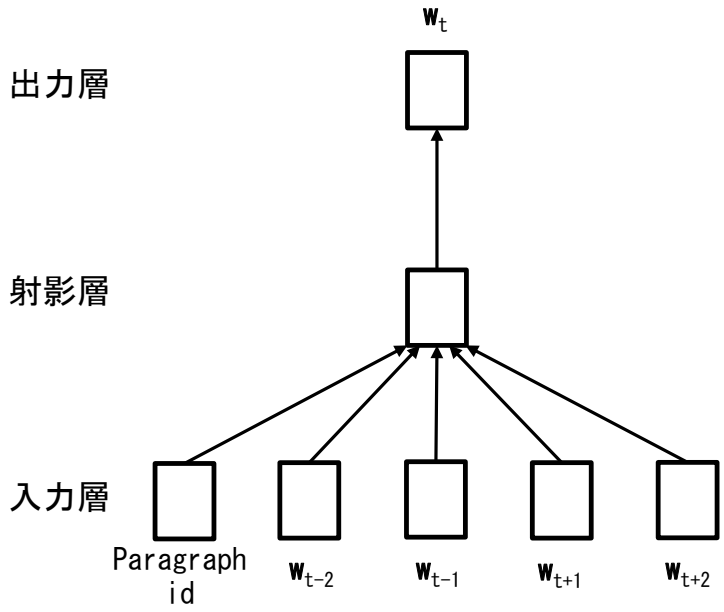


図 2.3: PV-DM モデル

モデルの構造はCBoW と同様であり, 入力にParagraph id を加える点が異なる. ここでParagraph id とは, 学習データの文章につけた番号である. 単語 w_t の前後2k語 $(w_{t-k},\cdots,w_{t-1},w_{t+1},\cdots,w_{t+k})$ と Paragraph id を入力層に与え, 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を訓練することでParagraph Vector を得る.

2.3.2　PV-DBoW

PVBoW モデルを図 2.4 に示す.

モデルの構造は skip-gram と同様である. skip-gram モデルでは入力にベクトル化したい単語 w_t を与えたが, PV-DBoW モデルでは入力にParagraph id を与える.

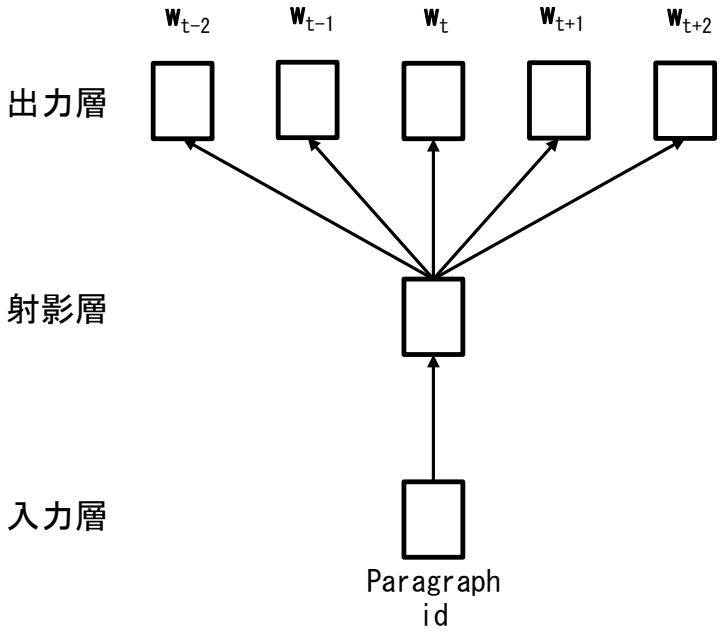


図 2.4: PV-DBoW モデル

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. また, 実装として自然言語用ライブラリである gensim [9] の doc2vec [10] を使用した.

第 3 章

関連研究

悪性 JavaScript を対象とする研究は広範に行われている．ここでは本研究と関連する研究として, 機械学習を用いた悪性 JavaScript 検知に関する既存研究を 1 件, ディープラーニングを用いた研究を 2 件紹介する．

3.1 機械学習を用いた悪性 JavaScript 検知

Likarish [11] は, 難読化が施された悪性 JavaScript から特徴量を選定し, 機械学習を用いて検知する手法を提案している．特徴量として, 1 行辺りの文字数, 空白の割合やコメントの量など, 難読化によく見られる特徴を用いている．機械学習の技法として Naive Bayes, ADTree, SVM, RIPPER の 4 種類を試し, 結果を比較している．

この研究では, 機械学習により結果に特徴が現れることがあると述べられている．例えば, Precision が最も良くなるのは SVM であり, Recall と F 値が最も良くなるのは RIPPER である．また誤検知の例として, 難読化が施された良性 JavaScript が挙げられている．

このような従来の機械学習を用いる手法では, 特徴量を選定するのは人間である．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量の選定が可能である．

3.2　ディープラーニングを用いた研究

ディープラーニングを用いた悪性 JavaScript に対する研究も活発に行われている．ここでは 2 件の研究を紹介する．

三須 [12] は, 悪性 JavaScript を Paragraph Vector で表現することでクラスタリングを行った．クラスタリングの結果, 類似度の高い悪性 JavaScript 同士を比較するとソースコード全体の構造が同じであり, 一部の値が変わっているだけであることがわかった．これにより, Paragraph Vector を用いた悪性 JavaScript のクラスタリングが有効であることを示した．

Wang [13] は, SdA (Stacked denoising auto-encoder) を用いて悪性 JavaScript 検知を行った．SdA を特徴量抽出に用いて, 悪性判定にはロジスティック回帰を用いることで悪性検知に有効であることを示した．また機械学習の他の手法との比較を行っている．

第 4 章

提案手法

4.1 提案手法の概要

本提案手法では, JavaScript のソースコードを Paragraph Vector で表現することで特徴ベクトルを得る. プログラムとしての文法が定まっており, ブラウザが意味を解釈し動作をする, という点で JavaScript のソースコードは自然言語における文章に見なすことができると考えられる. そこで, JavaScript のソースコードを文章と見なすことで, Paragraph Vector を利用して良性・悪性 JavaScript の特徴を表す, というのが本提案手法の狙いである.

本提案手法は 2 段階の手順に分けることができる. 手順 1 では JavaScript のソースコードを整形した上で Paragraph Vector で表現することで特徴ベクトルを作成する. 単語の整形方法については第 4.2 節で述べる. 手順 2 では, 手順 1 で得られた特徴ベクトルに対して SVM を用いて悪性判定を行う.

4.2 手順 1: 特徴ベクトルの作成

最初に, Paragraph Vector に入力するために文章を単語毎に分割する必要がある. 一般的に JavaScript において, ファイルサイズを減らすことを目的に空白を削除する場合がある. また, 悪意を持った難読化が施される場合には空白が取り除かれていることがある. したがって, ソースコードを空白で区切っても単語に分割することはできない. そこで本研究では, JavaScript の構文解析ツールである esprima [14] を用いて単語の分割を行った. esprima を用いた分割の

例を図 4.1 に示す.

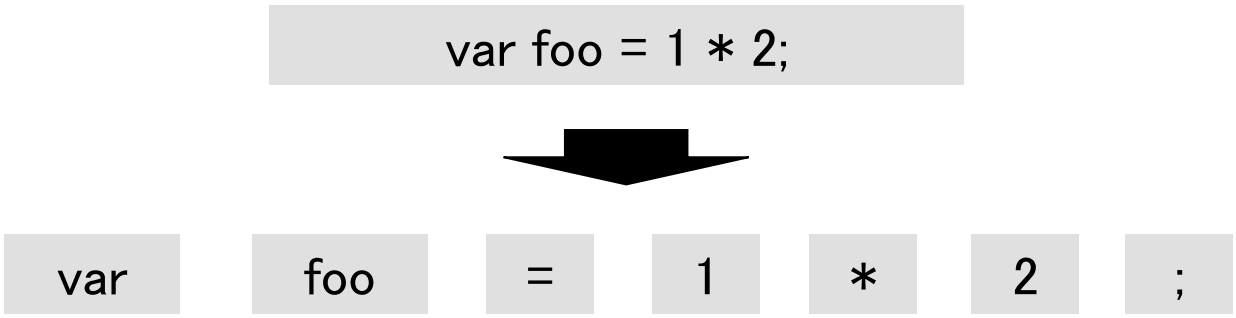


図 4.1: 単語分割の例

次に単語の置換を行う. JavaScript における変数名や関数名は自由に決めることができる. したがって, 変数名や関数名それ自体に意味が無いことも多く, Paragraph Vector を作成する上でこれらがノイズとなってしまう可能性がある. そこで, 変数名と関数名を抽象化することで, Paragraph Vector を作成する際に構文的な特徴をより際立たせる. 具体的には, 変数名を_NAME, 関数名を_FUNC として変換する. 単語の置換の例を図 4.2 に示す.

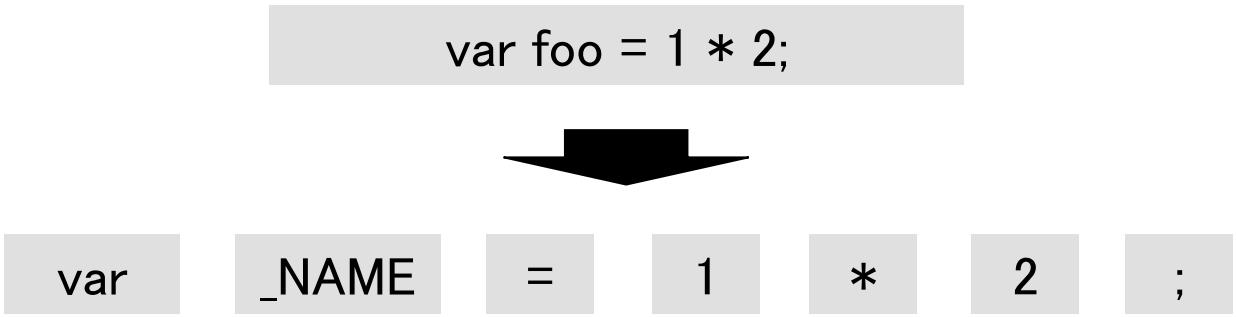


図 4.2: 単語置換の例

最後に, 単語数が 15 単語以下のデータは除外する. 単語数が少なすぎる場合には, 特徴量をうまく抽出できないため除外している. このようにして整形した JavaScript のソースコードを Paragraph Vector の入力とすることで特徴ベクトルを得る.

4.3　手順 2: JavaScript の悪性判定

手順 1 で JavaScript ソースコードごとの特徴ベクトルを得た. 得られた特徴ベクトルに, 良性・悪性のラベル付けを行う. ラベル付けした特徴ベクトルを, 教師あり学習器である SVM の入力として学習させる. この学習済みの SVM を用いて, 未知のデータの悪性判定を行う.

悪性判定を行う JavaScript のソースコードは, 手順 1 と同様にして整形をする. その後, 手順 1 で Paragraph Vector を作成するのに用いたものと同じモデルに適用することでベクトル化を行う. ベクトル化したものを学習済みの SVM に与えることで, 悪性かどうかの判定をする.

第 5 章

実証実験

5.1 実験に用いるデータ

5.1.1 悪性データセット

本実験では悪性 JavaScript の収集を, malware-traffic-analysis [15], Zerophage Malware [16], Phish Tank [17] から行った. malware-traffic-analysis と Zerophage Malware では, 閲覧者をマルウェアに感染させる Web サイトへアクセスした際の通信データを pcap として公開している. これら 2 つの Web サイトから pcap を収集し, pcap に含まれる JavaScript のデータを抽出する. 抽出した JavaScript の中で悪性であるものを本実験の悪性データとして扱う.

Phish Tank は, フィッシングサイトの情報を共有する Web サイトである. ここに掲載されたフィッシングサイトにアクセスし, 実行される JavaScript の収集を行った. 収集した JavaScript の中で悪性であるものを本実験の悪性データとして扱う.

JavaScript が悪性であるかどうかの判定には VirusTotal [18] を用いた. VirusTotal は, 複数のアンチウイルス製品を用いてアップロードしたファイルの検査を行う Web サービスである. これにより, 1 つ以上のアンチウイルス製品により悪意があると判定された JavaScript を本実験の悪性データセットとして用いる. 以上の手順により用意した, 490 個の悪性 JavaScript を実験に用いた.

5.1.2　良性データセット

本実験では良性 JavaScript の収集を, Alexa [19] のランキングに掲載されている Web サイトから行った. Alexa では, 世界 186 カ国それぞれにおける Web サイトのアクセス数ランキングを掲載している. そこで, 各国のランキング上位 50 サイトを巡回することで JavaScript を収集した.

なお, このランキングに掲載されている Web サイトに悪性なサイトが含まれている可能性を考慮し, 収集した JavaScript は VirusTotal で検査をした. 検査の結果, アンチウイルス製品により 1 つも悪性と判定されなかった JavaScript を本実験の良性データセットとして扱う. 以上の手順により用意した, 3017 個の良性 JavaScript を実験に用いた.

5.2　評価指標

本実験で利用する評価指標について述べる. 一般的に機械学習の精度評価の指標の 1 つとして混同行列 (confusion matrix) がある. 混同行列は, 分類したデータと真の結果を比較し表にまとめたものである. これを表 5.1 に示す.

表 5.1: 混同行列

		真の結果	
		悪性	良性
推定結果	悪性	True Positive(TP)	False Positive(FP)
	良性	False Negative(FN)	True Negative(TN)

True Positive (TP) は悪性 JavaScript を正しく悪性と判定できた数であり, False Positive (FP) は良性 JavaScript を誤って悪性と判定した数である. また, True Negative (TN) は良性 JavaScript を正しく良性と判定できた数であり, False Negative (FN) は悪性 JavaScript を誤って良性と判定した数である.

この混同行列から算出できる指標として, 適合率 (Precision), 再現率 (Recall), F 値 (F-measure) の 3 つの指標がある. 本実験ではこれら 3 つの指標を算出した. 各値の定義を以下に示す.

適合率 (Precision) は, 悪性であると推定した JavaScript の内, 真に悪性であるものの割合である. これは式 5.1 のように表せる.

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

再現率 (Recall) は, 悪性 JavaScript の内, 悪性であると判定できたものの割合である. これは式 5.2 のように表せる.

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

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値である. 式で表すと以下の式 5.3 のようになる.

$$F - measure = \frac{2Recall \cdot Precision}{Recall + Precision} \tag{5.3}$$

5.3　実験手順

提案手法を適用するにあたり, Paragraph Vector のパラメータを決定する必要がある. Paragraph Vector で用いるパラメータの内容を以下に示す.

epoch 学習回数

window 入力層のサイズ

size 作成するベクトルの次元数

min_count 出現回数がこの値より小さい単語を除外する

alpha 学習率

本研究では, これら 5 つのパラメータのうち, epoch, window, size の 3 つのパラメータについて選定を行った. そこで, これらのパラメータを 1 つずつ変化させながら第 5.2 節で挙げた評価指標を算出していき, 最も良い指標となった場合のパラメータを採用していく. 今回の実験では, 評価指標の良し悪しが判別しにくい場合には, 再現率の値を重視した判定をする.

なお, SVM に関してもパラメータを設定する必要がある. こちらに関しては事前にグリッドサーチを行い, 最も良かったものを採用した上で各実験を行う. また, 各実験の際は 5 分割交差検定を行う.

5.4　実験結果

5.4.1　SVM のパラメータ

SVM で必要なパラメータはグリッドサーチにより決定した. パラメータ C はコストパラメータであり, この値が小さいほど誤分類を許容し, 大きいほど誤分類を許容しない. gamma は RBF カーネルで用いるパラメータであり, この値が小さいほど単純な境界となり, 大きいほど複雑な境界となる. その値を以下の表 5.2 に示す.

表 5.2: SVM パラメータ

カーネル	RBF
C	10
gamma	0.1

Paragraph Vector のパラメータを選定する際は, SVM のパラメータはすべて表 5.2 の値を用いて実験を行った.

5.4.2　epoch

本節では, 最適な epoch 数 (学習回数) の選定を行う. 選定の際の epoch 以外のパラメータを表 5.3 に示す. なお, SVM のパラメータは第 5.4.1 節で示したものを用いる.

表 5.3: epoch 選定に用いるパラメータ

size	1400
window	6
min_count	0
alpha	0.003

epoch 数は, 最小値を 1 として, 5,10,15... と 5 ずつ増やしていき, 最大値を 100 として実験を行った. このときの結果をグラフにまとめたものを図 5.1 に示す.

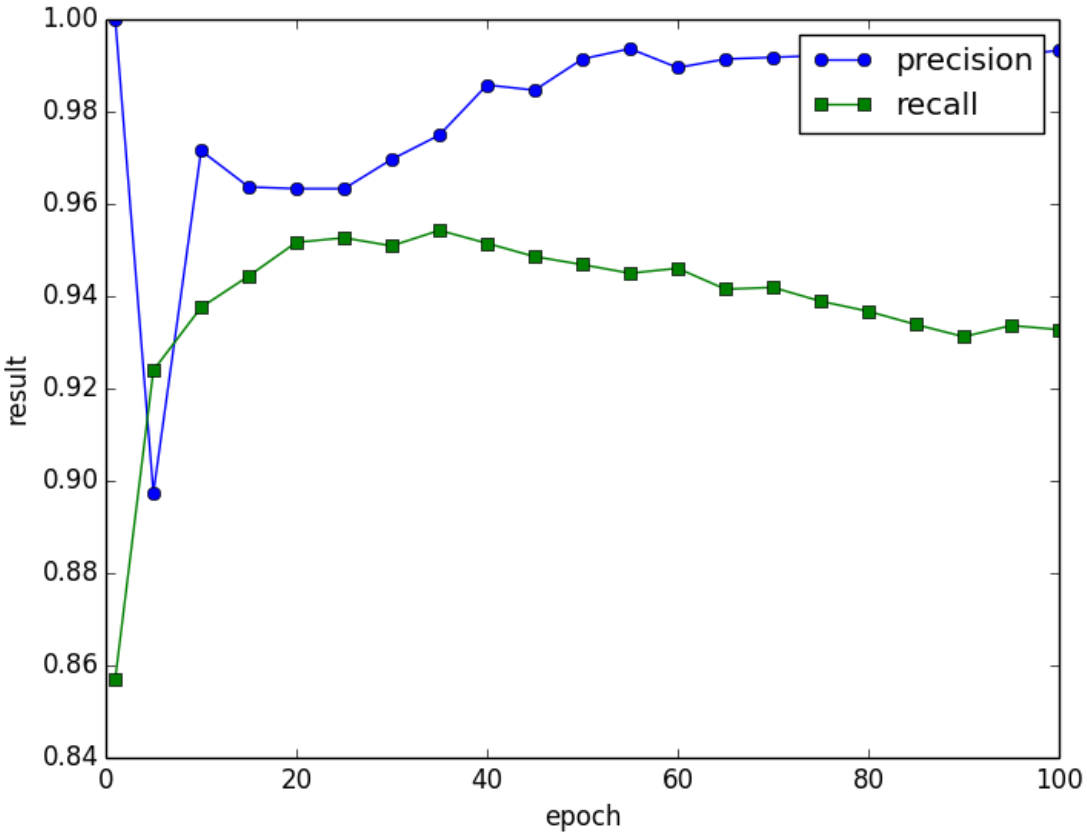


図 5.1: epoch 数の選定

epoch 数を増やしていくと, Precision は上がり, Recall は下がっていく傾向が見てとれる. Recall の値が一番高くなるのは, epoch が 35 のときであったため, epoch=35 を最適なパラメータとした.

5.4.3　window

本節では, 最適な window の選定を行う. 第 5.3 節で述べたように, window は入力層のサイズである. 選定の際の window 以外のパラメータを表 5.4 に示す. なお, SVM のパラメータは第 5.4.1 節で示したものを用いる. epoch は第 5.4.2 節で求めたものを用いる.

表 5.4: window 選定に用いるパラメータ

epoch	35
size	1400
min_count	0
alpha	0.003

window は, 最小値を 1, 最大値を 30 として 1 ずつ増やしながら実験を行った. このときの結果をグラフにまとめたものを図 5.2 に示す.

window の値と, Precision, Recall の値の間には相関が見られなかった. window が 12 と 17 のときに Recall の値は多少高くなっている. 今回は window=12 のときを最適なパラメータとした.

5.4.4　size

本節では, 最適な size (ベクトルの次元数) の設定を行う. 選定の際の size 以外のパラメータを表 5.5 に示す. なお, SVM のパラメータは第 5.4.1 節で示したものを用いる. epoch, window はそれぞれ, 第 5.4.2 節, 第 5.4.3 節で求めたものを用いる.

表 5.5: size 選定に用いるパラメータ

epoch	35
window	12
min_count	0
alpha	0.003

size は, 最小値を 100, 最大値を 3000 として 100 ずつ増やしながら実験を行った. このときの結果をグラフにまとめたものを図 5.3 に示す.

size が 900 までは Precision が高く Recall が低い傾向があり, 1000 を超えると Precision が下

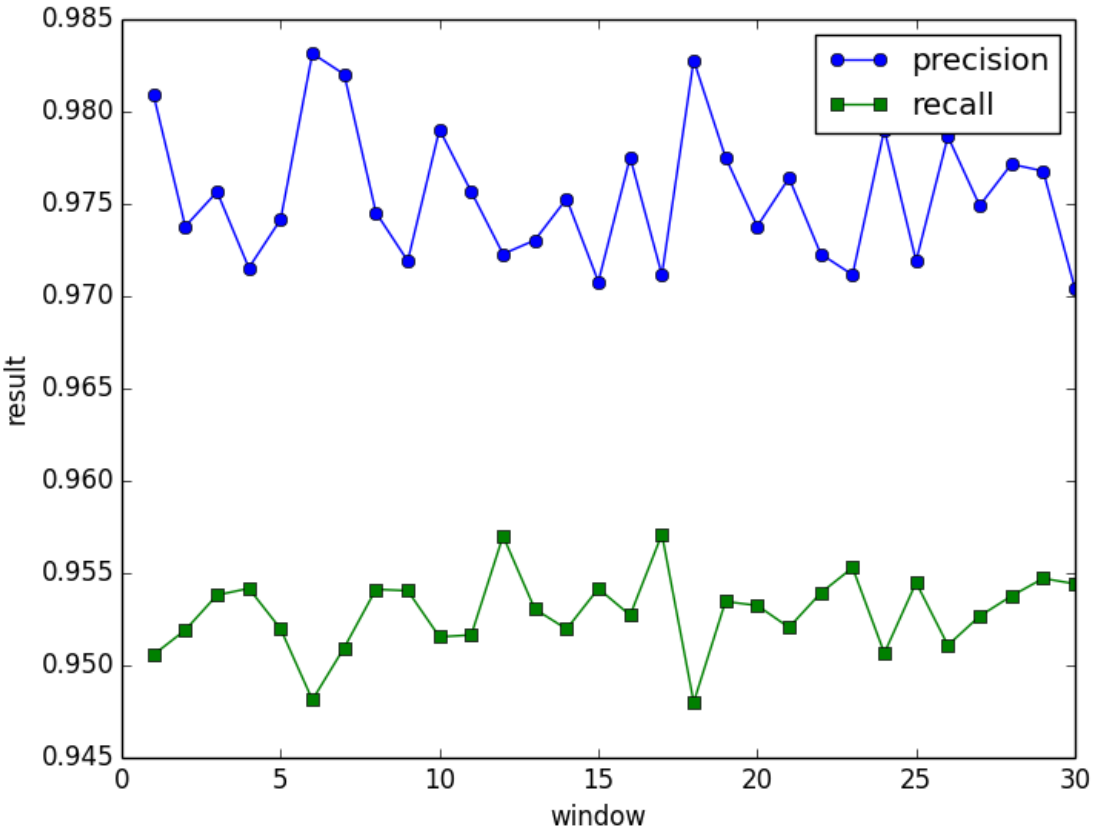


図 5.2: window の選定

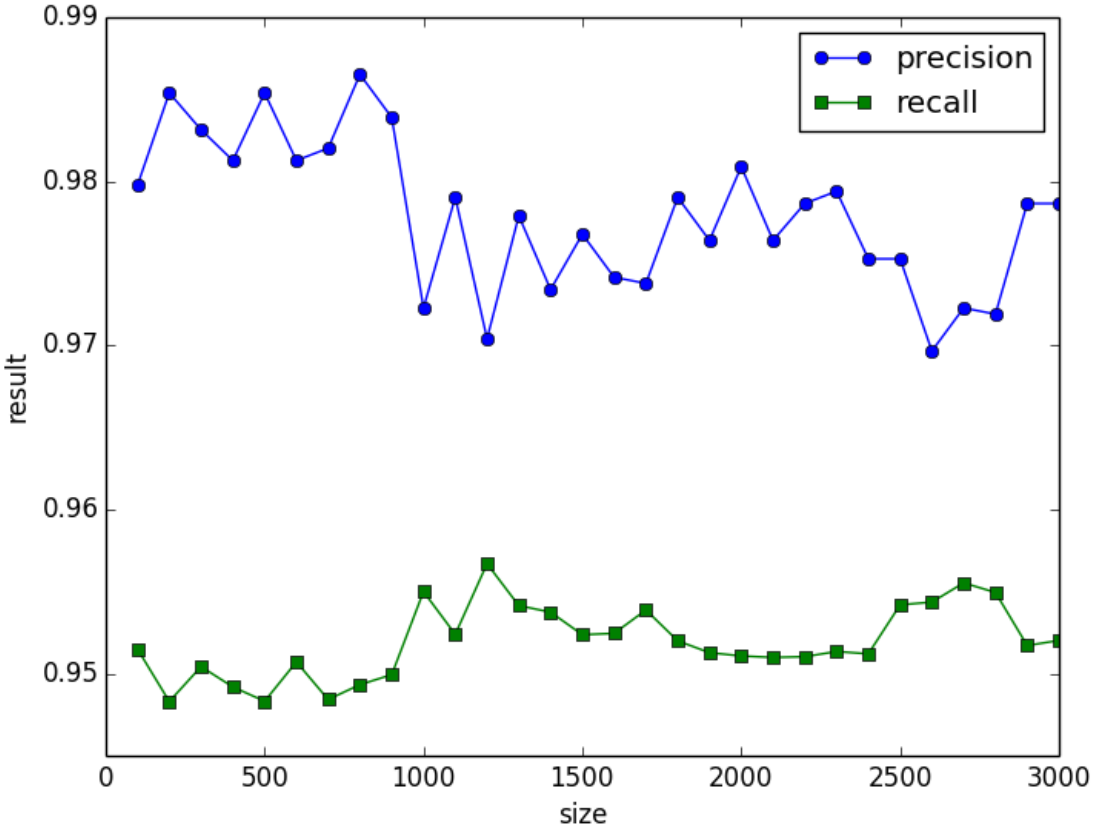

0.99
0.98
0.97
0.96
0.95
result
0
500
1000
1500
2000
2500
3000
size
precision
recall

図 5.3: size の選定

がり Recall が上がっている. 1000 以上では size と結果に相関は見られなかった. size=1200 のときに Recall が最も高くなっているため, 1200 を最適なパラメータとした.

5.4.5　結果

以上より, 最適な Paragraph Vector のパラメータが求まった. 結果を表 5.6 にまとめる.

表 5.6: 最適な Pragraph Vector のパラメータ

epoch	35
window	12
size	1200
min_count	0
alpha	0.003

また, このときの評価指標を表 5.7 に示す. ここでは比較のため, Likarish らの SVM による手法 [11] を SVM, Wang らの SdA を用いた手法 [13] を SdA-LR として, 本提案手法を doc2vec-SVM として示す.

5.5　考察

実験結果より, 本提案手法は既存手法に比べても十分な性能を発揮していることがわかる. したがって, JavaScript のソースコードを自然言語に見立てて求めた Paragraph Vector は, 悪性,

表 5.7: 実験結果

手法	Precision	Recall	F-measure
doc2vec-SVM	0.970	0.957	0.963
SdA-LR	0.949	0.948	0.948
SVM	0.920	0.742	0.764

良性の特徴を表すことに成功していると言える.

ただし, 語検知をしてしまったデータも存在する. 以下の図 5.4 にその例を示す. なお, わかりやすくするため適宜インデント, 改行を挿入している.

```
eval(function(a, b, c, d, e, f) {
    if (e = function(a) {
            return a
        }, !"".replace(/^/, String)) {
        for (; c--;) f[c] = d[c] || c;
        d = [function(a) {
            return f[a]
        }], e = function() {
            return "\\w+"
        }, c = 1
    }
    for (; c--;) d[c] && (a = a.replace(new RegExp("\\b" + e(c) + "\\b", "g"), d[c]));
    return a
}('5 13=["\\23\\24\\25\\26\\27\\28\\24","\\29\\27\\30\\30\\24\\23","\\32\\33\\34\\28\\35",
JuicyCodes.Run("ZXZhbChmdW5jdGlvbihwLGEsYyxrLGUsZCl7ZT1mdW5jdGlvbihjKXtyZXR1cm4gYy50b1N0cm
```

図 5.4: 誤検知の例

これは, 良性の JavaScript を悪性と誤検知した例である. この JavaScript の動作としては, 現在の Web ページのドメインが一定の値ではないとき, 特定のページへ遷移させるものである. それを難読化した結果, このようなソースコードとなっている.

動作内容や難読化を考慮すると悪性であるように見えるため, 本提案手法でも悪性と誤検知をしたものと思われる. しかし, 遷移先のページが一般の Web ページであるため VirutTotal による判定結果は良性であったと考えられる. このように, 良性 JavaScript であるにも関わらず, 一見すると悪性 JavaScript であるかのような難読化, あるいは動作内容の誤検知が見受けられた.

第 6 章

結論

6.1 まとめ

本論文は, JavaScript のソースコードを自然言語の文章に見立て, Paragraph Vector を用いて特徴ベクトルを作成する手法を提案した. その特徴ベクトルを SVM に学習させることで, 悪性 JavaScript を検知可能であることを示した. Precision は 97.0%, Recall は 95.7%となった.

本手法は, ソースコードを元に Paragraph Vector を作成することで判定を行うため, 人手を介さず自動で行うことが可能である. また, ソースコードに難読化が施されていても, 解除することなくそのまま判定を行うことができるという利点がある. そのため, ソースコードを読み解いていく静的解析に比べると短時間で検査を行うことができる.

6.2 今後の課題

6.2.1 ソースコードの整形方法の工夫

本研究では, Paragraph Vector に与えるソースコードの下準備として, 変数名と関数名の抽象化を行った上で, 単語数の少ないデータは取り除いた. しかし, 少ない単語数からなる JavaScript も存在する. 例えば, 複数のファイルにまたがるソースコードの場合は 1 つのファイルだけを見ると単語数が少ない場合がある. この場合には, 外部ファイルで宣言された変数や関数は本提案手法ではうまく抽象化が行えない.

そこで, ソースコードが複数ファイルにまたがる場合でも問題なく動作する整形方法が今後の課題として挙げられる. また, 変数名と関数名の他にも抽象化すべき単語の発見など, ソースコードの整形方法には様々な工夫の余地がある. 整形方法を改善することで精度向上にもつながると考えられるため, 今後の課題として挙げられる.

6.2.2　さまざまなデータでの検証

本研究では, 悪性データセットとして, malware-traffic-analysis[15], Zerophage Malware[16], Phish Tank[17] からデータを収集した. しかし, 個々の JavaScript がどのような動作をするものかは調べていない. 今後は, より様々な動作をする悪性 JavaScript を収集する必要がある.

また, 悪性データセットは良性データセットに比べ少ない数であった. そこで, 様々な種類かつ良性データと同程度の悪性 JavaScript を収集し, 検証を行う必要がある.

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. また, 本研究を進めるにあたり, 後藤研究室の皆様には様々なアドバイスとご協力をいただきましたことを心より感謝いたします.

参考文献

[1] IPA, “ドライブ・バイ・ダウンロード攻撃とは,” https://www.ipa.go.jp/files/000008801.pdf.

[2] TREND MICRO, “2017 年第 1 四半期セキュリティラウンドアップ,” TREND MICRO, https://www.trendmicro.com/ja_jp/security-intelligence/research-reports/sr/sr-2017q1.html.

[3] TREND MICRO, “2017 年上半期セキュリティラウンドアップ,” TREND MICRO, https://www.trendmicro.com/ja_jp/security-intelligence/research-reports/sr/sr-2017h1.html.

[4] TREND MICRO, “サービスとしてのエクスプロイトキット,” TREND MICRO, https://www.trendmicro.com/vinfo/jp/threat-encyclopedia/web-attack/3192/exploits-as-a-service.

[5] Ryo Kiire, Shigeki Goto, “Detecting Drive-by-Download Attacks based on HTTP Context-Types,” Proceedings of the APAN - Research Workshop 2016, pp.1–7, August 2016.

[6] 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.

[7] 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.

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

[9] “gensim topic modelling for humans,” https://radimrehurek.com/gensim://radimrehurek.com/gensim/.

[10] “models.dic2vec - Deep learning with paragraph2vec”, https://radimrehurek.com/gensim/models/doc2vec.html.

[11] Peter Likarish, Eunjin Jung, and Insoon Jo, “Obfuscated Malicious Javascript Detection using Classification Techiniques,” 2009 4th International Conference on Malicious and Unwanted Software, pp.47–54, October 2014.

[12] 三須 剛史, 巻島 和雄, 岡田 晃市郎, 岩本 一樹, “ソースコードの類似度に基づく悪性 JavaScript の分類に関する一検討,” Computer Security Symposium 2017, October 2017.

[13] Yao Wang, Wan-dong Cai, and Peng-cheng Wei, “A deep learning approach for detecting malicious JavaScript code,” SECURITY AND COMMUNICATION NETWORKS 2016, pp.1520–1534, February 2016.

[14] “ECMAScript parsing infrastructure for multipurpose analysis,” http://esprima.org/.

[15] “MALWARE-TRAFFIC-ANALYSIS.NET,” http://www.malware-traffic-analysis.net/

[16] “Zerophage Malware,” https://zerophagemalware.com/

[17] “Phish Tank,” https://www.phishtank.com/

[18] “Virus Total,” https://www.virustotal.com/

[19] “Alexa,” https://www.alexa.com/topsites/countries