機械学習を用いた悪性Pythonパッケージの検出

近藤 徳彦　大山 恵弘

Python 言語は他人の作ったコードを容易に再利用できる，パッケージという仕組みを持っている．パッケージはオープンソースで誰でも公開・利用ができる便利な機能である一方で，近年悪性のコードを含むものが増加している．そのため，悪性のパッケージの素早い検出が，利用者の安全のために必要になってきている．しかし，悪性コードを含んだ Python パッケージの検出に関する研究やツールは少なく，多数のユーザがダウンロードする危険性がある．そこで本研究では，機械学習により悪性の Python パッケージを検出する手法を提案する．この手法では，まずパッケージ内の Python コードから予め定めたルールに基づき特徴を抽出する．この操作を悪性・良性の両サンプルに適用し，両方の特徴を基に機械学習で悪性と良性に分類するためのモデルを作成する．この手法を良性と悪性を含むデータセットを使用して評価した結果，約 92%の精度で正しく分類することができた．

1　はじめに

Python への注目が近年高まっているとともに，悪性のコードを含む Python パッケージも増加傾向にある [4]．パッケージは自分以外が作成したコードを容易に再利用するための仕組みであり，多くのユーザが利用している．しかしこの仕組みを使い，悪意を持ったパッケージ開発者が悪性のパッケージを作成し，頒布している [12]．例えば，Python のパッケージはインストール時に，依存関係などのパッケージのメタデータを処理する必要がある．この処理は様々な手法で実現されるが，しばしば setup.py というファイルによって処理する手法が利用される．この手法で処理を行うには，setup.py 自体を実行する必要があるため，標準的なパッケージ管理ツールである pip でパッケージをインストールする場合には，setup.py がインストール時に自動実行されるようになっている．この機能を悪用し，setup.py に悪性のコードを書いておくことで，インストール時にその処理を気づかれずに実行する悪性パッケージが多く存在する [6,7,9]．

以上の状況から，悪性 Python パッケージを検出する必要性が極めて高くなっている．しかし，バイナリコードをはじめとするマルウェアを検出する研究は数多くあり，検出ツールやそれに使われるルールも多く作成されている一方で [1,2]，悪性の Python コードの検出に関する研究やツールは少なく，ましてや Python パッケージに関する研究やツールはさらに少ない．その結果，PyPI などのパッケージ管理レポジトリ内に悪性のパッケージが登録されてから，発見・削除までに多くのユーザがダウンロードした事例がある [8]．

そこで本研究では，機械学習を用いて悪性 Python パッケージを自動で検出するための手法を提案する．この手法では，パッケージから抽出した特徴を機械学習に使用した，パッケージが悪性か良性かを予測するプログラムを作成した．このプログラムを使用して，用意したデータセットを交差検証で評価する実験を行った．

提案手法を用いたプログラムは以下のことを達成できている．

- 実験では，既存のツールよりも高い検出精度を出すことができている

Detection of Malicious Python Packages Using Machine Learning

Norihiko Kondo, Yoshihiro Oyama, 筑波大学, University of Tsukuba.

- 予測モデルの作成に使用する特徴として，パッケージ内のファイルから静的に得られる情報のみを使用しているため，パッケージのダウンロード時などに，短時間で簡易的に調査する目的に活用が可能

本論文の構成は以下のようになっている．2 節で関連研究について述べた後，3 節で提案手法を説明し，4 節で実験の結果と評価を述べる．5 節で結論を述べる．

2 関連研究

2.1 悪性 Python コードの検出

Fang らの研究 [5] では，以下のような特徴を使って機械学習によって予測モデルを作成し，分類する手法を提案した．

- ファイル内に事前に定めた悪性と疑われるコードが存在するか
- 情報エントロピーや最長文字列などに関する，コードの難読化がなされているかを示唆する数値
- Python ファイルをコンパイルして得られるバイナリのオペコードを FastText というプログラムや，TF-IDF という手法で処理して得られた値

この研究では以上の手法により，高い精度で悪性コードを含んだ Python ファイルを検出できることが示されているが，通常複数のファイルで構成されるパッケージでは，この手法が有効か否かは未知数である．

2.2 検出ツールの精度比較

Vu らの研究 [14] では，PyPI Malware Checks[†1] [15]，OSS Detect Backdoor [11]，Bandit4Mal [13] の 3 つの悪性パッケージ検出ツールについて，対象の Python パッケージデータセットから悪性パッケージを検出する処理の検出率と誤検出率を調査した．調査では各ツールを用いて，1 種類の悪性データセットと 2 種類の良性データセットをそれぞれ 2 通りの方法でスキャンした．この調査では，検出率が最も高い場合でも約 90%であり，その他の場合はさらに低い．また最も高い検出率を出した場合では，誤検出率が 80%を超えてしまっている．

†1 以前 PyPI で使用されていた悪性パッケージ検出ツール．現在はリンク切れなどのため，入手できないようである．

3 提案手法

本節では，提案手法について説明する．提案手法では，パッケージのファイルに対し，設定されたルールにのっとってマッチングを行うことで特徴を抽出し，この特徴を使って機械学習で予測モデルを作成する．具体的には，まずサンプルのパッケージ内の Python ファイル内のコードが，既存ツールである PyPI Malware Checks を基に作成したルール群 (表 1) にマッチするか調べる．表 1 で示されるルール群は，対象の Python コードが悪性コードに多く見られる特徴を持っているかを調べることを目的として作成されている．直感的には，コードがマッチするルールが多ければ多いほど，そのコードが悪性である可能性が高い．

次に，マッチしたルールに相当する要素を 1，マッチしなかったルールに相当する要素を 0 とする特徴ベクタを作成する (図 1)．ここで，パッケージ内にある `setup.py` と `__init__.py` はパッケージ特有のファイルであり，通常のコードとは違うコードが書かれることが多いため，このマッチング処理を `setup.py`，`__init__.py`，それ以外の 3 種類に分けて行う．もし `__init__.py` やその他の Python ファイルがパッケージ内にない場合は，その部分の特徴は全て −1 とする．

最後に，作成した 3 種類の特徴ベクタを連結し，そのパッケージの特徴ベクタとする (図 2)．図 2 の例においては，`setup.py` ではルール 2 とルール 3 にマッチするため，特徴ベクタのその部分に相当する要素を 1 とし，`__init__.py` が存在しないため，特徴ベクタの相当する部分を全て −1 としている．特徴ベクタのその他の Python ファイルに相当する要素は，分類されるファイルが 1 つでもマッチすれば 1 とする．例ではルール 1 とルール n がマッチしているため，相当する要素を 1 としている．

表 1　使用したルール

ルール	ライブラリ	関数
ネットワーク通信	socket/ftplib/requests/ socketserver/http/ssl/ xmlrpc/urllib	指定なし
デシリアライズ	base64/binhex/pickle/	pytransform/pyarmor_runtime/ pytransform_bootstrap
プロセス生成 1	os	system/exec/spawn/popen/ posix_spawn
プロセス生成 2	subprocess	run/call/check_call/ check_output/Popen
プロセス生成 3	multiprocessing	Pool/Processapply/apply_async/start
プロセス生成 4	threading	run/Thread/start/
メタプログラミング系	importlib/compileall/ py_compile/inspect	__dir__/compile/dir/ __import__/eval/exec/ord/ getattr/__dict__/vars/ globals/locals/chr/
Win32 API	ctypes/win32com/ pypi2in32	指定なし

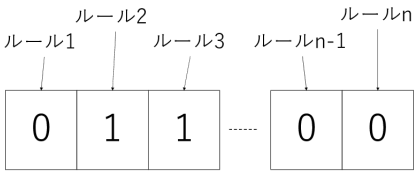

図 1　ファイルの特徴例: ルール 2 とルール 3 がマッチ

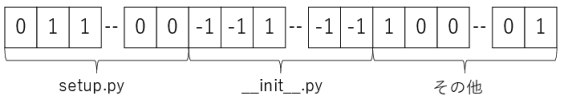

図 2　パッケージの特徴例

以上の処理をデータセット内の全てのパッケージに対して行い，集めた特徴を機械学習で使うデータセットとする．この作成した特徴の集合を基に機械学習を行い，予測モデルを作成する．機械学習のアルゴリズムには，実行速度が速く精度も高いランダムフォレストを使用する．

4　実験

4.1　実験環境

本研究の実験環境は表 2 のとおりである．また，実験で使用した Python ライブラリは表 3 のとおりである．

4.2　データセット

本研究ではデータセットとして，悪性データセット約 1380 個，良性データセット約 760 個の合計約 2140 個を使用した．悪性データセットは MalOSS というツールの GitHub レポジトリ [10] から約 120 個，DataDog というセキュリティ会社の GitHub レポジトリ [3] から約 1270 個入手した．一方で良性のサンプルはデータセットとしてまとまったものがなかったため，2.2 節で紹介した Vu らの研究 [14] を参考に，PyPI でダウンロード数が上位 1000 個のパッケージをダウンロードした．ここからパッケージ内に Python ファイルが存在しないものや，Python2.x 系のコードを含んでいるためなど，特徴を抽出するプログラムでエラーになるものを除いてまとめた約 760 個を良性データセットとして使用した．

これらのデータセットは全て `setup.py` というファイルを含んだ，source distribution と呼ばれる形式のパッケージで構成されている．これは，悪性データセットのほとんどがこの形式であり，これ以外の形式の悪性サンプルがほとんどないためである．また，良性データセット内のパッケージは source distribution ではない形式も持っているものがあるが，悪性サ

表 2　実行環境

CPU	AMD Ryzen 7 2700X Eight-Core Processor 3.70 GHz
メモリ	32 GB
OS	Windows 10 Pro
仮想マシン	Oracle VM VirtualBox 6.1.38
仮想マシン上のメモリ	4 GB
仮想マシン上の OS	Windows 10 Pro
使用プログラミング言語	Python 3.11

表 3　使用した Python ライブラリ

ライブラリ名	バージョン
`numpy`	1.25.0
`scikit-learn`	1.2.2

ンプルとパッケージの形式をそろえるため，source distribution のみを収集して使用している．

4.3　既存手法との比較実験

入手した悪性サンプルと良性サンプル約 2140 個に提案手法を適用し，与えられたパッケージが良性か悪性かを推定する実験を行った．作成されたモデルの検出精度の評価には Stratified k-fold を使用し，k=5 で交差検証を行った．Stratified k-fold は，通常の k-fold 交差検証がデータセットを無作為に k 個に分割するのに対し，データセットのラベルの比率を保ったまま k 個に分割するというものである．つまり，良性と悪性の比率が 2:1 のデータセットをこの方法で分割した場合，分割後の k 個のブロック 1 つ 1 つも良性と悪性の比率が 2:1 になる．

この実験の結果を表 4 に示す．表 4 では，提案手法の結果と並べて，2.2 節で示した Vu らの研究で使われたツールの実験結果も示している．Vu らの研究では検出率と誤検出率のみ出されていたため，その部分のみを表示している．またこの実験では，良性のデータセットを 2 種類使用していたが，表には結果の良かった方のデータセットの値を示している．

表 4 の提案手法の結果では全ての指標が 9 割を超えており，既存のツールと比較して再現率・特異度ともに全て上回っている．また，どれか 1 つの指標だけ特段低いといったことがなく，特異度や適合率が高いことから，良性を間違えて悪性とするようなことが防ぎやすく，良性が圧倒的に多い現実の環境に適したモデルになっていると考えられる．

4.4　アルゴリズムの比較実験

これまでの実験では，機械学習のアルゴリズムは速度などの面からランダムフォレストを使用したが，特徴の傾向によってはこれが最適でない可能性がある．そこで他のアルゴリズムと精度や実行速度を比較する実験を行った．この実験では，提案手法で使用したランダムフォレストの他に，サポートベクターマシン，多層パーセプトロン，k 近傍法の 3 つのアルゴリズムを使用した．サポートベクターマシンは，データセットを分割するための線を，分割後の集団からできるだけ離そうとするアルゴリズムで，分類でよく使われる．多層パーセプトロンは，パーセプトロンという複数の入力データから 1 つの出力を返す関数を組み合わせることで，複雑なデータを解釈するアルゴリズムである．k 近傍法は，与えられたデータのラベルを一番近い k 個のデータのラベルの多数決で予測する，クラスタリングでよく使われるアルゴリズムである．これらのアルゴリズムを表 3 で示した scikit-learn を使用して実装した．また，これらのアルゴリズムを使用するときのハイパーパラメータは全てデフォルトとし，何も入力せずに実装した．

この比較実験の結果を表 5 に示す．表 5 の学習時間は，k-fold で分割した 1 回あたりの学習時間である．

この結果をみると，この中ではランダムフォレストが今回のような用途に対するアルゴリズムとしては最適と思われる．まず検出精度に関する指標をみると，

表 4　提案手法と既存手法の結果 (%)

	精度	適合率	再現率	F 値	特異度
提案手法	92.3	96.3	91.6	93.9	93.6
PyPI Malware Checks (手法 1)			58.9		85.1
PyPI Malware Checks (手法 2)			85.7		32.2
OSS Detect Backdoor (手法 1)			50.6		66.9
OSS Detect Backdoor (手法 2)			85.1		22.8
Bandit4Mal (手法 1)			66.7		29.8
Bandit4Mal (手法 2)			90.5		15.4

ランダムフォレストと多層パーセプトロンの 2 つが，バランス良く高い値を出している．サポートベクターマシンは特異度や適合率が他のアルゴリズムよりも高いが，再現率がそれ以上に落ちてている．残った k 近傍法は，ほとんどの指標が他のアルゴリズムよりも大きく下がっている．

次に速度に関する指標を見る．k 近傍法は，学習時間が他のアルゴリズムよりも大幅に短く，多層パーセプトロンが圧倒的に長い．残りの 2 つに関しては，サポートベクターマシンの方がランダムフォレストと比べて 5 倍以上速いが，0.1 秒と 0.02 秒では作業効率に影響が出るほどの差はないと考えられる．また，予測時間に関しては多層パーセプトロンを除き微差であり，多層パーセプトロンにおいても他のアルゴリズムの数十倍速いものの，学習時間の遅さを埋めるほど速くないと判断した．

5　結論

本研究では，機械学習を使用して悪性 Python パッケージを検出する手法を提案した．提案手法ではパッケージのファイルを 3 種類に分け，それぞれルールごとにコードのマッチングを行い特徴を抽出した．抽出した特徴を使用し，Python パッケージを悪性と良性に予測するモデルをランダムフォレストで作成し，実験によって評価したところ，約 92%の精度を得ることができ，既存ツールの精度を超えることができた．

また，今後の課題としては次のことが挙げられる．まず，提案手法ではシグネチャマッチングのような悪性パッケージの特徴となるルールを予め作成する手法をとっている．そのため，全く新しいタイプの悪性パッケージではマッチするルールが存在しない場合が出てくる恐れがある．また，今回使用した悪性サンプルのパッケージでは似たようなパッケージが多くなっている．これらのことから今回のモデルでは十分な汎化性能が得られていない可能性がある．今後はこの部分を解決する手法の開発が必要であると考える．

謝辞 本研究の一部は JSPS 科研費 23K11096 の助成を受けている．

参 考 文 献

[1] Alahmadi, A., Alkhraan, N., and BinSaeedan, W.: MPSAutodetect: A Malicious Powershell Script Detection Model Based on Stacked Denoising Auto-Encoder, *Computers & Security*, Vol. 116, No. 102658,(2022).

[2] Belaoued, M. and Mazouzi, S.: A Real-Time PE-Malware Detection System Based on CHI-Square Test and PE-File Features, CIIA 2015, 2015, pp. 416–425.

[3] DataDog: malicious-software-packages-dataset, `https://github.com/DataDog/malicious-software-packages-dataset`. (Accessed on 04/08/2022).

[4] Digmi, I.: The rising trend of malicious packages in open source ecosystems, `https://snyk.io/blog/malicious-packages-open-source-ecosystems/`, 2023. (Accessed on 20/07/2023).

[5] Fang, Y., Xie, M., and Huang, C.: PBDT: Python Backdoor Detection Model Based on Combined Features, *Security and Communication Networks*, Vol. 2021(2021). DOI: 10.1155/2021/9923234.

[6] Hai, S. B.: PyPI に Windows ユーザーを狙う 6 つの Python パッケージを発見, `https://unit42.paloaltonetworks.jp/malicious-packages-in-pypi/`, 2023. (Accessed on 04/08/2023).

[7] Lakshmanan, R.: Warning: PyPI Feature Executes Code Automatically After Python Package

表 5　アルゴリズムごとの結果

	(%)					(秒)	
	精度	適合率	再現率	F 値	特異度	学習時間	予測時間
ランダムフォレスト	92.3	96.3	91.6	93.9	93.6	0.1287	0.0100
サポートベクターマシン	91.8	96.7	90.3	93.4	94.3	0.0284	0.0139
多層パーセプトロン	92.3	96.2	91.6	93.9	93.4	1.5014	0.0002
k 近傍法	91.0	94.3	91.7	92.9	89.6	0.0004	0.0208

Download, https://thehackernews.com/2022/09/warning-pypi-feature-executes-code.html, 2022. (Accessed on 04/08/2023).

[8] Lakshmanan, R.: Researchers Uncover Obfuscated Malicious Code in PyPI Python Packages, https://thehackernews.com/2023/02/researchers-uncover-obfuscated.html, 2023. (Accessed on 20/07/2023).

[9] Lee, J.: More Supply Chain Attacks via New Malicious Python Packages in PyPi, https://www.fortinet.com/blog/threat-research/more-supply-chain-attacks-via-new-malicious-python-packages-in-pypi, 2023. (Accessed on 04/08/2023).

[10] MalOSS: ossanitizer, https://github.com/osssanitizer/maloss. (Accessed on 31/03/2023).

[11] Microsoft: OSS Detect Backdoor, https://github.com/microsoft/OSSGadget/wiki/OSS-Detect-Backdoor. (Accessed on 30/12/2022).

[12] Team, P. R.: Malicious Actors Use Unicode Support in Python to Evade Detection, https://blog.phylum.io/malicious-actors-use-unicode-support-in-python-to-evade-detection/, 2023. (Accessed on 31/05/2023).

[13] Vu, D. L.: Bandit4Mal, https://github.com/lyvd/bandit4mal. (Accessed on 30/12/2022).

[14] Vu, D.-L., Newman, Z., and Meyers, J. S.: A Benchmark Comparison of Python Malware Detection Approaches, *arXiv*, (2022). https://arxiv.org/abs/2209.13288.

[15] Warehouse: Malware Checks, https://github.com/pypi/warehouse/blob/main/.github/ISSUE_TEMPLATE/malware-check.md. (Accessed on 09/08/2022).