applied sciences

MDPI

*Article*

# Classifying Malicious Documents on the Basis of Plain-Text Features: Problem, Solution, and Experiences

**Jiwon Hong** [1], **Dongho Jeong** [2] and **Sang-Wook Kim** [1,*]

[1] Department of Computer Science, Hanyang University, Seoul 04763, Korea; nowiz@hanyang.ac.kr
[2] Department of Artificial Intelligence, Hanyang University, Seoul 04763, Korea; mars9954@hanyang.ac.kr
* Correspondence: wook@hanyang.ac.kr

**Abstract:** Cyberattacks widely occur by using malicious documents. A malicious document is an electronic document containing malicious codes along with some plain-text data that is human-readable. In this paper, we propose a novel framework that takes advantage of such plaintext data to determine whether a given document is malicious. We extracted plaintext features from the corpus of electronic documents and utilized them to train a classification model for detecting malicious documents. Our extensive experimental results with different combinations of three well-known vectorization strategies and three popular classification methods on five types of electronic documents demonstrate that our framework provides high prediction accuracy in detecting malicious documents.

**Keywords:** malware; malicious document; classification; text analysis

check for updates

**Citation:** Hong, J.; Jeong, D.; Kim, S.-W. Classifying Malicious Documents on the Basis of Plain-Text Features: Problem, Solution, and Experiences. *Appl. Sci.* **2022**, *12*, 4088. https://doi.org/10.3390/app12084088

Academic Editor: Shi-Jinn Horng

Received: 28 February 2022
Accepted: 15 April 2022
Published: 18 April 2022

## 1. Introduction

The threat of cyberattacks continues to increase. Various types of new cyberattacks are happening every month. Phishing and ransomware attacks are on a steep rise despite the efforts of governments and companies. Various studies to avoid such risks from cyberattacks were conducted [1–5].

Cyberattacks are in most cases ultimately achieved by executable code called malware. In the past, attackers mainly distributed such executable code directly (e.g., by using the autorun feature of an external storage device). However, recently, most systems have been able to easily detect this malicious executable code [6,7]. Therefore, attackers started to distribute the malicious code indirectly by hiding it inside the electronic document delivered through the web or e-mails [7–9].

The electronic document having the malicious code is called a malicious document. There have been a number of efforts on the detection of malicious documents in the literature [1–3,8,10]. Many of them perform detection by finding signatures for mostly encrypted malicious code in documents. However, in many cases, malicious documents contain plaintext data in addition to the malicious code, as in normal (i.e., benign) electronic documents [8,10].

Generally, plaintext data in a malicious document are used to disguise the document as a benign one [2,8,11]. However, even among plaintext data, in many cases, we could find signatures that are beneficial in determining whether the document is malicious [12]. Such signatures include reusing sentences, techniques, and habits that had been used in previous malicious documents by a group of attackers.

Our contributions in this paper are summarized as follows:

- We propose a malicious-document classification framework that only exploits plaintext data. Our proposed framework extracts plaintext features from given malicious or benign electronic documents and builds vector representations of the documents (i.e., vectorization). After that, it trains a classification model by using the vector representations of all documents. By applying the trained model to an unseen (new) document, our framework predicts whether the target document is malicious.

- We designed our proposed framework to be capable of adopting various vectorization strategies and classification methods. We adopted three well-known vectorization strategies (i.e., bag of words (BoW), term frequency-inverse document frequency (TF-IDF), and Word2Vec) and three popular classification methods (i.e., deep neural networks (DNNs), support vector machines (SVMs), and decision trees) in this paper.
- We show that the proposed framework works well in practice. We carefully evaluate the effectiveness of our framework equipped with different combinations of vectorization strategies and classification methods by conducting extensive experiments on five types of electronic documents. Experimental results demonstrate that our framework provides accuracy higher than 98% in detecting malicious documents.

Furthermore, since our proposed framework successfully detects malicious documents by exploiting only the plaintext data without using the signature of malicious codes, it can be easily combined with those methods that utilize only the signature of malicious codes, thereby utilizing both plaintext data and the signature of malicious codes for detecting malicious documents.

The paper is organized as follows: Section 2 briefly introduces the concept of a malicious document, and Section 3 presents our framework of malicious-document detection using plaintext features. Section 4 evaluates the effectiveness of the proposed framework by using different types of real-world document sets. Lastly, Section 5 summarizes and concludes our paper.

## 2. Malicious Code and Malicious Documents

In general, the word 'malware' indicates both malicious code and malicious documents. Most cyberattacks are ultimately performed by malicious code, which is basically executable code that exists as a portable executable (PE) file [13]. There could also be malicious code that is fileless malware stored in the system configuration file or registry in the form of shellcode [11]. When a system is compromised by malicious code, sensitive information in the system may be stolen or destroyed, or important files may be encrypted (e.g., ransomware). In addition, an attacker can remotely manipulate a compromised system to perform malicious tasks, such as attacking other systems.

Traditionally, approaches most widely used to deliver malware include deceiving a legitimate user to copy it and exploiting a security hole in the system such as the autorun of an external storage device [1,7,13]. Attackers commonly try to use web pages, e-mail messages, and attachment files as attack vectors for malware distribution [8,11,14].

Figure 1 depicts these recent attack vectors. The attacker (1) plants malicious code into electronic documents, (2) delivers them through a webpage, an e-mail, or a message disguised as a benign one to a victim, and convinces the victim to download and open the electronic document file [11]. When the victim (3) opens the malicious document using a legitimate but vulnerable program, the malicious code hidden inside the document (4) is activated and compromises the system. The most common file types used to convey such cyberattacks include hypertext markup language (HTML) [14], extensible markup language (XML) [14], rich text format (RTF) [15], e-mail (EML) [16], Microsoft Word (DOC) [17], Microsoft Excel (XLS) [17], and portable document format (PDF) [12]. Such electronic documents used in attack vectors are called malicious documents [14,18].

Malicious documents are used in attack vectors instead of executable files mainly because they are easier to be disguised than executable files are. Malicious documents contain some or all of the malicious code that performs the actual cyberattack in an encrypted state or downloads actual malicious code via the network (i.e., called drop). However, malicious documents are not executable by themselves, thereby bypassing the traditional signature-based detection of malicious PE files. Such malicious documents may not be detected by antivirus software, firewalls, or intrusion prevention systems (IPS).

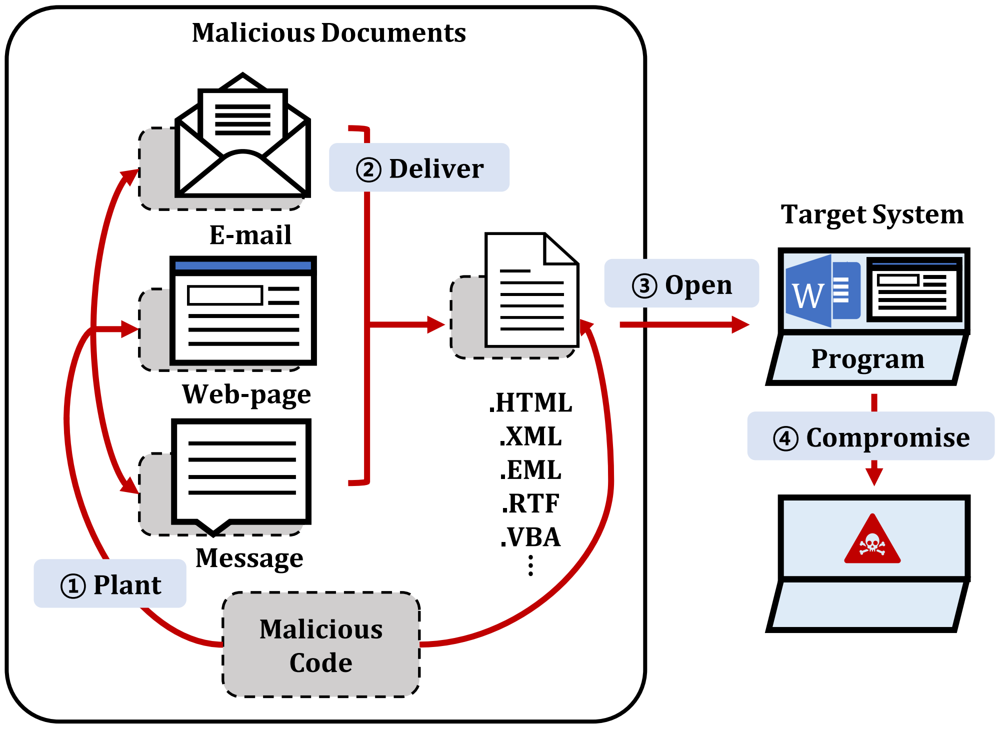

**Figure 1.** Malicious documents used as attack vectors.

Malicious documents contain plaintext data in addition to malicious code for the actual cyberattack. In such plaintext data, there exist specified attributes in the electronic document format (e.g., tags for an HTML file and control keywords for an RTF file) and text contents intended to be readable by humans (e.g., the body of an e-mail message). Format-specific attributes are not intended to be executable but may convey executable malicious code. Attackers may use plaintext contents to disguise the document as legitimate.

In this paper, we address how to detect malicious documents by exploiting these plaintext features. Our scope covers the accurate detection of malicious documents in five types of electronic documents commonly used in attack vectors: HTML, XML (including XHTML), EML (e-mail message), RTF, and Visual Basic for Application (VBA).

## 3. Malicious Document Classification

In this section, we present our proposed framework of detecting malicious documents by using plaintext features in detail.

### 3.1. Overview

We propose a simple but effective two-step classification framework to detect malicious documents containing plaintext features. Figure 2 shows the overview of the proposed framework composed of two steps: (1) feature extraction and (2) classification.

In the feature extraction step, (1) we first preprocess the given set of documents to read their plaintext data properly. Then, (2) we perform tokenization to extract only plaintext features from the electronic document. After that, (3) we construct a vector representation for each document on the basis of extracted plaintext features by applying a text analysis technique. The detailed process is described in Section 3.3.

In the classification step, (4) a classification model is trained with the obtained vector representations of the training samples (i.e., documents with class labels). (5) By using the trained classification model, our framework predicts whether a given (unknown) document is malicious or not. The detailed process is described in Section 3.4.

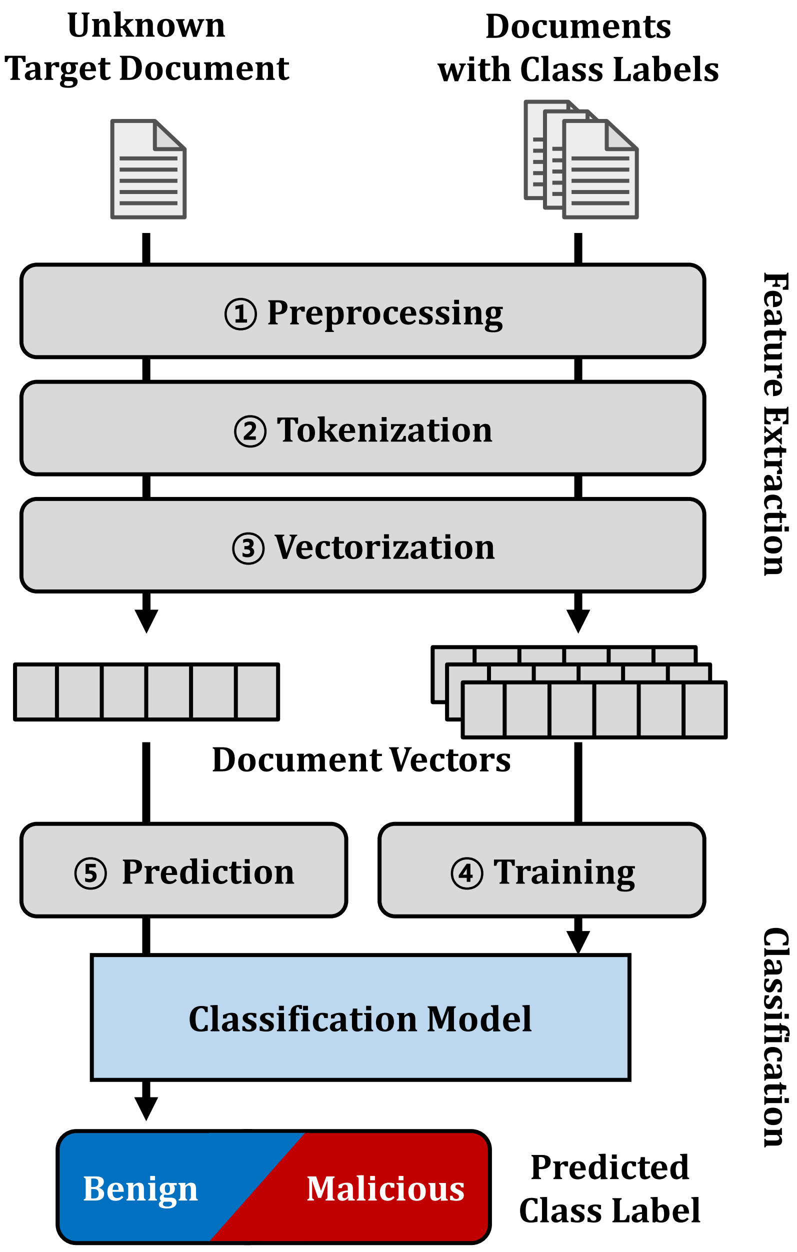

**Figure 2.** Overview of the proposed framework.

### 3.2. Electronic Documents

In the field of natural-language processing (NLP), a document conceptually indicates a group of text that human beings can understand. A document is made up of multiple sentences, each of which subsequently consists of words. Most text analysis techniques simply consider a document as a sequence of words. Text analysis techniques attain the vector representations of words that appear in the entire corpus, aggregating them to obtain the vector representation of the document for its classification [19–22].

An electronic document corresponds to a computer file whose content is a document. An electronic document has a specific structure that allows for itself to be processed by a particular computer program (e.g., word processor, web browser, and/or e-mail client). This structure has two types of elements: plaintext is text that human beings can understand; attributes are those used to specify how the plaintext is displayed in a document or to express information other than plaintext (e.g., figures, tables, and other documents). Common examples of attributes include HTML tags and the markdown formatting syntax.

Plaintext is encoded in various character encodings (e.g., extended ASCII encodings and UTF-8 encoding) and stored as a byte array in an electronic document file. The human-readable text is obtained by decoding these byte strings with an appropriate encoding. Similarly, attributes are also encoded in various ways and stored as a byte array in the file. Generally, attributes surround or are included in the plaintext; so, plaintext and attributes appear alternately in a document.

In the following sections, we briefly describe the structures of the most common file types used to convey cyberattacks.

#### 3.2.1. HTML and XML

HTML is a language for marking up documents to be shown in a web browser. HTML files have the .html extension. Although XML is a more general-purpose markup language than HTML, we mainly cover extensible hypertext markup language (XHTML) files with the .xml extension and consider them to be documents displayed in a web browser similar to HTML. In these two markup languages, attributes constituting the structure of a document are the words surrounded by '<' and '>', which are called tags. These tags surround plaintext meant to be shown to the user.

#### 3.2.2. EML

The .eml extension is used by e-mail clients such as Windows Mail, Microsoft Outlook, and Mozilla Thunderbird to store e-mail messages. In an EML file, header fields store the information such as the sender or receiver of the message and the date of transmission. The message body and attachments are stored together according to the multipurpose Internet mail extensions (MIME) standard. Among the message body and attachments, we decode only those with a MIME type starting with 'text/' with the specified encoding, using them as plaintext data. Data in header fields are treated as attributes.

#### 3.2.3. RTF

RTF is a document format commonly used by various word processors. RTF contains attributes called control keywords that start with a '\'. With the proper encoding, we could decode both control keywords and plaintext data into human-readable texts.

#### 3.2.4. VBA

VBA is a scripting language used in various Microsoft applications; unlike aforementioned file types, it is not generally considered to be electronic documents. However, files with the .vba extension are composed of human-readable scripts and are used in various cyberattacks as a common attack vector. Each VBA file is regarded as a document with keywords and plaintext strings constituting the language syntax of VBA as words.

Table 1 shows the sample plaintext contents of each file type above. In many cases, a cyberattack using an electronic document tries to hide the actual cyberattack code in attributes to exploit the vulnerability when the program used to open the file renders a particular attribute [15]. We equally regarded words from plaintext and attributes as words, so that the arrangement of these words represents an electronic document.

### *3.3. Feature Extraction*

This subsection discusses how to generate a representation vector from each electronic document. We considered each electronic document as a list of words (both plaintext and attributes), as mentioned in Section 3.2.

**Table 1.** Examples of document contents for HTML (XML), EML, RTF, and VBA.

| File Type | Plaintext Examples |
|---|---|
| | . . . |
| | <div style="border: 1 px solid #f90; margin: 0–15 px; |
| | padding: 10 px;"> |
| HTML & XML | <h2>Big Size Clothing for Men</h2> |
| | <strong>Clothes sizes from 2XL to 8XL</strong>with |
| | <strong>chest and waist sizes</strong>up to 72 inches. |
| | </div><section class='home_page_links'> |
| | ... |
| | . . . |
| | Content-Type: text/plain; charset="utf-8" |
| EML | Content-Transfer-Encoding: quoted-printable |
| | Content-Disposition: inline |
| | Hallo. Im Anhang finden Sie eine Zahlungskopie. |
| | . . . |
| | . . . |
| | {\f0\fswiss\fcharset0 Arial;} |
| | {\f1\fmodern Courier New;} |
| RTF | {\colortbl\red0\green0\blue0;\red0\green0\blue255;} |
| | \uc1\pard\plain\deftab360 \f0\fs20 Invio file invoice.xml, |
| | con identificativo 123456789. In allegato il file contenente |
| | la fattura ed il file contenente i metadati.\par |
| | . . . |
| | . . . |
| | Set SomeDoc = App.OpenDocumentByCode("DOC") |
| | Set rsSomeDoc = SomeDoc.DataSets("MAIN") |
| | For i = 0 To rsSomeDoc.Fields.Count − 1 |
| VBA |     Fld_Name = rsSomeDoc.Fields.Item(i).Name |
| |     If Left(Fld_Name, 1) = "A" Then |
| |         FldVal(Fld_Name) = rsSomeDoc.FldVal(Fld_Name) |
| |     End if |
| | Next |
| | . . . |

#### 3.3.1. Preprocessing

We extracted human-readable text from a target document by using its appropriate encoding on the basis of its file type. First, we established the encoding of a file by using the description in the file or encoding predictor. Then, we decoded the file to convert it into Unicode text. We applied this process recursively to decode MIME data for EML files. We read each file and stored the extracted plaintext into Unicode byte arrays with predicted encoding.

#### 3.3.2. Tokenization

To extract and build the list of words from a plaintext chunk, we performed tokenization [23,24]. After preprocessing, we extracted the words from these Unicode byte arrays by using regular expressions [25]. In most target file types, both words for attributes and plaintext are separated by whitespace characters and nonalphanumeric characters (e.g., '\',

‘"’, ‘>’, and ‘<’). We performed tokenization by writing a regular expression suitable for each file type.

Lastly, we removed too-long tokens and tokens with appearances less than a certain number from the entire corpus [26]. We could thus safely remove nonplaintext tokens such as binary code or images.

#### 3.3.3. Vectorization

In order to apply a classification model to detect malicious documents, we need to represent the documents as vectors. To this end, existing text analysis techniques first build a vector representation of each word in the corpus, aggregating vectors of words included in a document to build a vector representation of the document. We use the three following well-known techniques in our framework for this document vectorization.

Bag of words (BoW) is a document representation technique widely used in NLP and information retrieval fields. In BoW vectorization, each word included in the corpus is encoded as a *one-hot vector* with a length equal to the number of words included in the corpus [27]. Each document becomes a sum of vectors of words included in the document. In the document vector obtained with the BoW technique, the value corresponding to a word appearing in the document is set as 1, and the value corresponding to a word not appearing in the document is set as 0. The time complexity of BoW vectorization is $O(Nl + NV)$, where $N$ is the size of the dataset, $l$ is the number of tokens in each sample, and $V$ is the size of the global vocabulary (e.g., the size of feature vector dimension).

Term Frequency-Inverse Document Frequency (TF-IDF) is another vectorization technique that considers the importance of each word within the corpus [28,29]. In TF-IDF, a document is represented as a vector with the length same as the number of words included in the corpus. An element of the document vector has a value multiplied by $TF_{w,d}$, which indicates the frequency with which the corresponding word appears in the document, and $IDF_{w,D}$, which indicates the importance of the word in the entire corpus. $TF_{w,d}$ is computed by counting the occurrences of word $w$ in the document $d$, and $IDF_{w,D}$ is done by counting the documents containing $w$, which is scaled logarithmically. The time complexity of TF-IDF vectorization is $O(2Nl + NV)$.

Word2Vec is a technique for learning the vector representation of each word in the corpus by using a neural network [30,31]. The vectors of words learned from Word2Vec are located more closely if they have similar meanings within the corpus. Word2Vec extracts a continuous skip-gram [32] from the corpus and then uses it to train a shallow neural network to predict the surrounding words of a given word [30,31]. A vector corresponding to each word is extracted from the projection layer of the neural network learned in this way. Since Word2Vec is a technique for obtaining a vector representation of a word, an additional aggregation step is required to obtain the final document vector. In this paper, we use the sum and average of the word vectors included in a document as the vector representation of the document. The time complexity of vectorization using Word2Vec could be represented as $O(NC(D + D\log_2 V))$, where $C$ is the size of the sliding window, $D$ represents the size of the resulting vectors.

### *3.4. Model Construction*

The proposed framework uses classification methods for malicious document detection. As described in Section 3.3, we obtained the document vector of a document in a training corpus. We built a classification model with training documents and used the model to predict whether a given target document is malicious. In this paper, we use three well-known classification models: decision trees, support vector machines (SVMs), and deep neural networks (DNNs).

#### 3.4.1. Decision Tree

The decision tree is a traditional method for classifying a given sample by using a treelike structure. A tree-shaped classification model is generated by using the decision tree induction where it assesses all features in a dataset [33–37]. The class of a given sample can be determined by using this classification model thus built. The decision tree is one of the most intuitive classifiers; looking at a tree helps us in seeing which features most affect a sample's classification. So, its explainability is considered to be high. The time complexity of the decision tree is $O(DN \log N)$.

#### 3.4.2. SVM

SVM is another classical classification model. SVM can achieve a high level of classification accuracy in a variety of fields and build sophisticated classification models [38,39]. It seeks an ideal hyperplane or a group of hyperplanes that distinguishes the class of data in a given dataset correctly and clearly. The ideal hyperplane is the one with the widest margin between classes, which helps the classifier in reducing future errors. Classification using SVM predicts the class of a given sample by determining the side of the found hyperplane to which the given sample belongs. The training complexity of nonlinear SVM is generally between $O(N^2)$ and $O(N^3)$.

#### 3.4.3. DNN

Recently, thanks to the power of deep learning, various efforts on neural-network-based classification have been undertaken [20,21,40,41]. DNN performs classification by training an artificial neural network with a multilayer structure. Each layer of DNN learns to turn the data it receives into a more abstract and more composite representation. After the neural network is trained, it can be used to predict the class of a given sample in testing. The time complexity of DNN is $O(NTAB)$, where $T$ represents the number of iterations, $A$ and $B$ represent the input and output dimensions of a hidden layer, respectively.

In summary, we first preprocessed each electronic document into a Unicode text; second, we tokenized each Unicode text into a set of words; third, we built a vector representation of each document with BoW, TF-IDF, or Word2Vec methods; then, we built a classification model with the vectorized set of documents (i.e., the training set) by using the decision tree, SVM, or DNN methods; lastly, we predicted whether each of the given set of unknown documents (i.e., the test set) is malicious by using the classification model.

## 4. Evaluation

In this section, we evaluate and discuss the effectiveness of the proposed framework. Section 4.1 describes the experimental setup of the evaluation, and Section 4.2 presents and analyzes the results.

### *4.1. Experimental Setup*

#### 4.1.1. Datasets

For evaluation, we used a dataset built by cybersecurity domain experts. For each target file type, the experts crawled 1000 samples from VirusTotal [42] and adjusted the numbers of malicious or benign samples for each file type to be the same (i.e., 500 malicious samples and 500 benign samples). Each dataset was split into five folds by using stratified sampling [43] to perform cross-validation [43]. Then, in each run of cross-validation, words with less than 2 occurrences and those with lengths higher than 30 characters were removed from the corpus of our training set. In the vectorization using BoW and TF-IDF, the length of a document vector (i.e., dimensionality) was set as the number of words remaining in the final corpus of the training set; in vectorization using Word2Vec, it was set as 600, following [30].

#### 4.1.2. Model Parameters in Classification

We conducted our experiments by using various parameters and improvement tactics as follows to build each classification model:

- Decision tree: we used the classification and regression trees (CART) algorithm [44] and built a decision tree on the basis of Gini impurity. We employ multiple class weight settings of (100:1, 10:1, 1:1, 1:10, and 1:100).
- SVM: The model was built by using the radial basis function (RBF) kernel and the linear kernel [38,39].
- DNN: the model in our case uses five layers having 64 units for the document vector obtained from BoW and TF-IDF and 4096 units for the document vector obtained from Word2Vec. The activation function in the intermediate layer uses a rectified linear unit (ReLU) [45], and the sigmoid function was used in the last layer. We used AdaGrad and Adam optimizers [46,47]. We added the batch normalization, drop out, and early stop as improvement tactics [48–50]. We set patience parameters to 20, 30, and 40 for early stop.

After performing experiments on the various combinations of parameters above, we show only the result with the best score in each accuracy metric (described later).

#### 4.1.3. Evaluation Metrics

As metrics for evaluating classification model performance, we used accuracy, precision, and recall [51] defined as follows:

$$Accuracy = \frac{tp+tn}{tp+tn+fp+fn}, \tag{1}$$

$$Precision = \frac{tp}{tp+fp}, \tag{2}$$

$$Recall = \frac{tp}{tp+fn} \tag{3}$$

where $tp$, $fp$, $tn$, and $fn$ are numbers of true-positive, false-positive, true-negative, and false-negative predictions, respectively [51]. For example, $tp$ indicates the number of malicious samples correctly predicted by a model as malicious.

### *4.2. Results*

Table 2 shows the results for five datasets (i.e., HTML, XML, EML, RTF and VBA): each part of the table shows the average accuracy, precision, and recall of all five folds for every combination of a feature extraction method and a classification model. The boldfaced results indicates the best performance for each file type.

In the case of HTML files, the best accuracy and precision were obtained with the combination of DNN with Word2Vec$_{\text{Avg}}$. SVM with BoW showed the best recall, but lower accuracy and precision than DNN with Word2Vec$_{\text{Avg}}$. For XML files, DNN with BoW achieved 100% accuracy as shown in Table 2. SVM with BoW also showed very high accuracy. In the case of the EML files, DNN with BoW showed the best accuracy and recall. DNN with Word2Vec$_{\text{Avg}}$ showed 100% precision, but provided unreasonably low recall. In the case of RTF files, BoW presented remarkably high accuracy with both the SVM and the DNN model. Results for VBA files showed reasonably high accuracy both DNN with Word2Vec$_{\text{Avg}}$ and SVM with BoW.

Results showed mostly reasonably high accuracy for all file types and with most combinations of a feature extraction method and a classification model. As shown in Table 2, lower accuracy was obtained with EML than that with other datasets except for DNN with BoW. A possible reason is that some EML files did not include enough plaintext data for the classification. Word2Vec, a more sophisticated method for vectorization, was not more accurate than simple BoW and TF-IDF. This could have been due to the use of a naive aggregation strategy. Overall, DNN with BoW showed the highest accuracy. Experimental results validated that the proposed framework effectively detects malicious documents in a practical sense.

**Table 2.** Classification accuracy, precision, and recall.

| | | **Model** | **HTML** | | | **XML** | | | **EML** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **Acc.** | **Prec.** | **Rec.** | **Acc.** | **Prec.** | **Rec.** | **Acc.** | **Prec.** | **Rec.** |
| **Vectorization** | **BoW** | **SVM** | 0.979 | 0.965 | **0.994** | 0.997 | **1.000** | 0.994 | 0.900 | 0.899 | 0.886 |
| | | **DT** | 0.975 | 0.986 | 0.970 | 0.996 | 0.992 | **1.000** | 0.836 | 0.845 | 0.829 |
| | | **DNN** | 0.987 | 0.998 | 0.980 | **1.000** | **1.000** | **1.000** | **0.965** | 0.964 | **1.000** |
| | **TF-IDF** | **SVM** | 0.934 | 0.952 | 0.914 | 0.991 | 0.990 | 0.992 | 0.875 | 0.889 | 0.838 |
| | | **DT** | 0.973 | 0.982 | 0.964 | 0.995 | 0.996 | 0.998 | 0.829 | 0.858 | 0.831 |
| | | **DNN** | 0.982 | 0.994 | 0.970 | 0.997 | **1.000** | 0.998 | 0.946 | 0.975 | 0.920 |
| | **W2V$_{Sum}$** | **SVM** | 0.936 | 0.954 | 0.916 | 0.990 | 0.988 | 0.992 | 0.831 | 0.828 | 0.826 |
| | | **DT** | 0.920 | 0.920 | 0.920 | 0.988 | 0.992 | 0.992 | 0.798 | 0.791 | 0.810 |
| | | **DNN** | 0.973 | 0.996 | 0.950 | 0.980 | **1.000** | 0.992 | 0.770 | 0.919 | 0.962 |
| | **W2V$_{Avg}$** | **SVM** | 0.936 | 0.950 | 0.948 | 0.990 | 0.988 | 0.992 | 0.842 | 0.834 | 0.826 |
| | | **DT** | 0.915 | 0.912 | 0.920 | 0.988 | 0.984 | 0.992 | 0.809 | 0.791 | 0.773 |
| | | **DNN** | **0.995** | **1.000** | 0.992 | **1.000** | **1.000** | **1.000** | 0.885 | **1.000** | 0.784 |

| | | **Model** | **RTF** | | | **VBA** | | |
|---|---|---|---|---|---|---|---|---|
| | | | **Acc.** | **Prec.** | **Rec.** | **Acc.** | **Prec.** | **Rec.** |
| **Vectorization** | **BoW** | **SVM** | **0.998** | 0.998 | 0.998 | 0.996 | 0.996 | 0.996 |
| | | **DT** | 0.997 | 0.998 | 0.996 | 0.991 | 0.992 | 0.992 |
| | | **DNN** | **0.998** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** |
| | **TF-IDF** | **SVM** | 0.991 | 0.990 | 0.992 | 0.985 | 0.974 | 0.996 |
| | | **DT** | 0.996 | 0.996 | 0.996 | 0.984 | 0.984 | 0.984 |
| | | **DNN** | 0.993 | **1.000** | 0.992 | 0.993 | **1.000** | 0.992 |
| | **W2V$_{Sum}$** | **SVM** | 0.966 | 0.972 | 0.960 | 0.982 | 0.984 | 0.980 |
| | | **DT** | 0.966 | 0.970 | 0.974 | 0.979 | 0.973 | 0.986 |
| | | **DNN** | 0.952 | 0.952 | 0.958 | 0.924 | **1.000** | 0.968 |
| | **W2V$_{Avg}$** | **SVM** | 0.955 | 0.976 | 0.984 | 0.990 | 0.990 | 0.990 |
| | | **DT** | 0.974 | 0.966 | 0.960 | 0.987 | 0.980 | 0.998 |
| | | **DNN** | 0.985 | 0.990 | 0.982 | 0.996 | 0.996 | 0.996 |

## 5. Conclusions and Further Studies

In this paper, we proposed a framework that effectively detects malicious documents by using only plaintext features of electronic documents via text analysis techniques. The proposed framework extracts only human-readable plaintext features from a given document, excluding executable binary, and applies it to classification methods to determine whether it is malicious. To this end, we used three feature extraction strategies and three classification methods in this paper. In addition, we verified the effectiveness of our proposed framework through a series of experiments, which showed that it achieved more than 98% accuracy in detecting malicious documents thanks to the appropriate combination of a feature extraction strategy and a classification method.

As further studies, instead of simply using plaintext features, we improved detection performance by combining our framework with an existing signature-based detection technique for executable code. Adopting the ensemble methods [52,53] could also improve the accuracy of our proposed framework even further. In addition, we also plan to consider and exploit the order of words to obtain more accurate classification.

Another future study is to address real-world data imbalances where the number of malicious documents is significantly lower than that of benign documents.

**Author Contributions:** Conceptualization, J.H.; data curation, D.J.; funding acquisition, S.-W.K.; methodology, J.H.; project administration, J.H.; software, J.H. and D.J.; supervision, S.-W.K.; validation, J.H. and D.J.; visualization, J.H.; writing—original draft, J.H. and D.J.; writing—review and editing, S.-W.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work is the result of commissioned research project supported by the affiliated institute of ETRI[2021-015]. This work was also supported by the National Research Foundation of Korea (NRF) under Project Number 2020R1A2B5B03001960 and Institute of Information & Communications Technology Planning & Evaluation (IITP) under Project Number 2020-0-01373.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Ye, Y.; Li, T.; Adjeroh, D.; Iyengar, S.S. A Survey on Malware Detection Using Data Mining Techniques. *ACM Comput. Surv.* **2017**, *50*, 1–40. [CrossRef]
2. Or-Meir, O.; Nissim, N.; Elovici, Y.; Rokach, L. Dynamic Malware Analysis in the Modern Era—A State of the Art Survey. *ACM Comput. Surv.* **2019**, *52*, 1–48. [CrossRef]
3. Sihwail, R.; Omar, K.; Ariffin, K.A.Z. A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis. *Int. J. Adv. Sci. Eng. Inf. Technol.* **2018**, *8*, 1662. [CrossRef]
4. Kim, E.; Park, S.J.; Choi, S.; Chae, D.K.; Kim, S.W. MANIAC: A man-machine collaborative system for classifying malware author groups. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS), Virtual, 15–19 November 2021; pp. 2441–2443.
5. Pinhero, A.; Anupama, M.; Vinod, P.; Visaggio, C.A.; Aneesh, N.; Abhijith, S.; AnanthaKrishnan, S. Malware Detection Employed by Visualization and Deep Neural Network. *Comput. Secur.* **2021**, *105*, 102247. [CrossRef]
6. Bott, E. *Introducing Windows 10 for IT Professionals*; Microsoft Press: Redmond, WA, USA, 2016.
7. Singh, J.; Singh, J. A Survey on Machine Learning-Based Malware Detection in Executable Files. *J. Syst. Archit.* **2021**, *112*, 101861. [CrossRef]
8. Sudhakar; Kumar, S. An Emerging Threat Fileless Malware: A Survey and Research Challenges. *Cybersecurity* **2020**, *3*, 1–12. [CrossRef]
9. Mimura, M.; Tajiri, Y. Static Detection of Malicious PowerShell Based on Word Embeddings. *Internet Things* **2021**, *15*, 100404. [CrossRef]
10. Afreen, A.; Aslam, M.; Ahmed, S. Analysis of fileless malware and its evasive behavior. In Proceedings of the 2020 International Conference on Cyber Warfare and Security (ICCWS), Islamabad, Pakistan, 20–21 October 2020; pp. 1–8.
11. Mansfield-Devine, S. Fileless Attacks: Compromising Targets without Malware. *Netw. Secur.* **2017**, *2017*, 7–11. [CrossRef]
12. Smutz, C.; Stavrou, A. Malicious PDF detection using metadata and structural features. In Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC), Orlando, FL, USA, 3–7 December 2012; pp. 239–248.
13. Ye, Y.; Wang, D.; Li, T.; Ye, D. IMDS: Intelligent malware detection system. In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Jose, CA, USA, 12–15 August 2007; pp. 1043–1047.
14. Hou, Y.T.; Chang, Y.; Chen, T.; Laih, C.S.; Chen, C.M. Malicious Web Content Detection by Machine Learning. *Expert Syst. Appl.* **2010**, *37*, 55–60. [CrossRef]
15. Saad, G.; Raggi, M.A. Attribution is in the object: Using RTF object dimensions to track APT phishing weaponizers. *Virus Bull.* **2020**, *12*, 1–2.
16. Yadav, N.; Panda, S.P. Feature selection for email phishing detection using machine learning. In Proceedings of the International Conference on Innovative Computing and Communications (ICICC), New Delhi, India, 19–20 February 2022; pp. 365–378.
17. Yang, S.; Chen, W.; Li, S.; Xu, Q. Approach using transforming structural data into image for detection of malicious MS-DOC files based on deep learning models. In Proceedings of the 2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), Lanzhou, China, 18–21 November 2019; pp. 28–32.
18. Tzermias, Z.; Sykiotakis, G.; Polychronakis, M.; Markatos, E.P. Combining static and dynamic analysis for the detection of malicious documents. In Proceedings of the Fourth European Workshop on System Security (EUROSEC), Salzburg, Austria, 10 April 2011; pp. 1–6.
19. Kowsari, K.; Jafari Meimandi, K.; Heidarysafa, M.; Mendu, S.; Barnes, L.; Brown, D. Text Classification Algorithms: A Survey. *Information* **2019**, *10*, 150. [CrossRef]
20. Kowsari, K.; Brown, D.E.; Heidarysafa, M.; Meimandi, K.J.; Gerber, M.S.; Barnes, L.E. Hdltex: Hierarchical deep learning for text classification. In Proceedings of the 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), Cancun, Mexico, 18–21 December 2017; pp. 364–371.

21. Kowsari, K.; Heidarysafa, M.; Brown, D.E.; Meimandi, K.J.; Barnes, L.E. Rmdl: Random multimodel deep learning for classification. In Proceedings of the 2nd International Conference on Information System and Data Mining (ICISDM), Lakeland, FL, USA, 9–11 April 2018; pp. 19–28.
22. Aggarwal, C.C.; Zhai, C. A Survey of Text Classification Algorithms. In *Mining Text Data*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 163–222.
23. Gupta, G.; Malhotra, S. Text Document Tokenization for Word Frequency Count Using Rapid Miner. *Int. J. Comput. Appl.* **2015**, *975*, 8887.
24. Verma, T.; Renu, R.; Gaur, D. Tokenization and Filtering Process in RapidMiner. *Int. J. Appl. Inf. Syst.* **2014**, *7*, 16–18. [CrossRef]
25. Friedl, J.E. *Mastering Regular Expressions*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2006.
26. Pahwa, B.; Taruna, S.; Kasliwal, N. Sentiment Analysis-Strategy for Text Pre-Processing. *Int. J. Comput. Appl.* **2018**, *180*, 15–18. [CrossRef]
27. Harris, Z.S. Distributional Structure. *Word* **1954**, *10*, 146–162. [CrossRef]
28. Salton, G.; Buckley, C. Term-Weighting Approaches in Automatic Text Retrieval. *Inf. Process. Manag.* **1988**, *24*, 513–523. [CrossRef]
29. Jones, K.S. A Statistical Interpretation of Term Specificity and Its Application in Retrieval. *J. Doc.* **1972**, *28*, 11–21. [CrossRef]
30. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. In Proceedings of the International Conference on Learning Representations Workshop Track (ICLR Workshop), Scottsdale, AZ, USA, 2–4 May 2013; pp. 1301–3781.
31. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrases and their compositionality. In Proceedings of the Advances in Neural Information Processing Systems (NIPS), Lake Tahoe, NV, USA, 5–8 December 2013; pp. 3111–3119.
32. Goldberg, Y.; Levy, O. Word2vec explained: Deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv* **2014**, arXiv:1402.3722.
33. Morgan, J.N.; Sonquist, J.A. Problems in the Analysis of Survey Data, and a Proposal. *J. Am. Stat. Assoc.* **1963**, *58*, 415–434. [CrossRef]
34. Safavian, S.R.; Landgrebe, D. A Survey of Decision Tree Classifier Methodology. *IEEE Trans. Syst. Man Cybern. Syst.* **1991**, *21*, 660–674. [CrossRef]
35. Magerman, D.M. Statistical decision-tree models for parsing. In Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL), Cambridge, MA, USA, 26–30 June 1995; pp. 276–283.
36. Quinlan, J.R. Induction of Decision Trees. *Mach. Learn.* **1986**, *1*, 81–106. [CrossRef]
37. De Mántaras, R.L. A Distance-Based Attribute Selection Measure for Decision Tree Induction. *Mach. Learn.* **1991**, *6*, 81–92. [CrossRef]
38. Manevitz, L.M.; Yousef, M. One-Class SVMs for Document Classification. *J. Mach. Learn. Res.* **2001**, *2*, 139–154.
39. Han, E.H.S.; Karypis, G. Centroid-based document classification: Analysis and experimental results. In Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery (PKDD), Lyon, France, 13–16 September 2000; pp. 424–431.
40. Bengio, Y.; Courville, A.; Vincent, P. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *35*, 1798–1828. [CrossRef] [PubMed]
41. Schmidhuber, J. Deep Learning in Neural Networks: An Overview. *Neural Netw.* **2015**, *61*, 85–117. [CrossRef] [PubMed]
42. Virustotal. Available online: https://www.virustotal.com/ (accessed on 9 January 2019).
43. Kohavi, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI), Montreal, QC, Canada, 20–25 August 1995; Volume 2, pp. 1137–1143.
44. Hastie, T.; Tibshirani, R.; Friedman, J.H.; Friedman, J.H. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*; Springer: Berlin/Heidelberg, Germany, 2009; Volume 2.
45. Brownlee, J. A Gentle Introduction to the Rectified Linear Unit (ReLU). 2019. Available online: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ (accessed on 9 January 2019).
46. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. In Proceedings of the International Conference on Learning Representations (ICLR), San Diego, CA, USA, 7–9 May 2015.
47. Freund, Y.; Schapire, R.E. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. Syst. Sci.* **1997**, *55*, 119–139. [CrossRef]
48. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the International Conference on Machine Learning (PMLR), Lille, France, 7–9 July 2015; pp. 448–456.
49. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
50. Yao, Y.; Rosasco, L.; Caponnetto, A. On Early Stopping in Gradient Descent Learning. *Constr. Approx.* **2007**, *26*, 289–315. [CrossRef]
51. Powers, D. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. *J. Mach. Learn. Technol.* **2011**, *2*, 37–63.

52. Breiman, L. Bagging Predictors. *Mach. Learn.* **1996**, *24*, 123–140. [CrossRef]
53. Freund, Y.; Schapire, R.E. Experiments with a new boosting algorithm. In Proceedings of the International Conference on Machine Learning (ICML), Bari, Italy, 3–6 July 1996; Volume 96, pp. 148–156.