


RESEARCH

Open Access



Phishing page detection via learning classifiers from page layout feature

Jian Mao^{1*} , Jingdong Bian^{1,2†}, Wenqian Tian^{1,2†}, Shishi Zhu^{1,2}, Tao Wei³, Aili Li⁴ and Zhenkai Liang⁵

Abstract

The web technology has become the cornerstone of a wide range of platforms, such as mobile services and smart Internet-of-things (IoT) systems. In such platforms, users' data are aggregated to a cloud-based platform, where web applications are used as a key interface to access and configure user data. Securing the web interface requires solutions to deal with threats from both technical vulnerabilities and social factors. Phishing attacks are one of the most commonly exploited vectors in social engineering attacks. The attackers use web pages visually mimicking legitimate web sites, such as banking and government services, to collect users' sensitive information. Existing phishing defense mechanisms based on URLs or page contents are often evaded by attackers. Recent research has demonstrated that visual layout similarity can be used as a robust basis to detect phishing attacks. In particular, features extracted from CSS layout files can be used to measure page similarity. However, it needs human expertise in specifying how to measure page similarity based on such features. In this paper, we aim to enable automated page-layout-based phishing detection techniques using machine learning techniques. We propose a learning-based aggregation analysis mechanism to decide page layout similarity, which is used to detect phishing pages. We prototype our solution and evaluate four popular machine learning classifiers on their accuracy and the factors affecting their results.

Keywords: Anti-phishing, Machine learning, Aggregation analysis

1 Introduction

The web technology has become the cornerstone of a wide range of platforms, such as mobile services and smart Internet-of-things (IoT) systems. In such platforms, users' data are aggregated to a cloud-based platform, where web applications are used as a key interface to access and configure user data. Securing the web interface requires solutions to deal with threats from both technical vulnerabilities and social factors.

Phishing attacks are one of the most common form of social engineering attacks. In a web-based phishing attack, attackers use web pages visually mimicking legitimate web sites, such as banking and government services, to deceive the victims to input their sensitive information (e.g., bank accounts and social security number). Though phishing attacks do not require advanced technical knowledge and these attack techniques are becoming familiar to users,

they are still causing major financial damages. According to the report from the Anti-Phishing Working Group (APWG), there are 1,220,523 phishing attacks reported in 2016, which is a 65% increase over 2015 [1].

Several types of anti-phishing solutions have been developed for web-based phishing solutions. The traditional URL-based anti-phishing solutions [2–5] try to decide whether a page is a phishing page based on its URL. They are limited by the timeliness of malicious URL database update. The solutions based on page contents [6, 7] rely on the context or image processing techniques to detect phishing attacks, which can cause high performance overhead. As the phishing pages usually maintain similar page layouts to their target websites, the similarity of page layouts has been demonstrated as an important metric to detect phishing pages [8, 9]. In particular, features extracted from CSS layout files are used to measure page similarity. However, these measurements heavily rely on human experiences and thus may not be comprehensive to detect new attacks. How to comprehensively evaluate the pages' similarity remains a great challenge.

*Correspondence: maojian@buaa.edu.cn

†Jingdong Bian and Wenqian Tian contributed equally to this work.

¹School of Cyber Science and Technology, Beihang University, Xueyuan Road, Beijing 100083, China

Full list of author information is available at the end of the article

Machine learning has been widely used in many areas to create automated solutions. Researchers also use machine learning to detect phishing attacks based on various features [10–14]. The solutions show the potential of machine learning techniques to detect phishing attacks. In this paper, we aim to explore learning techniques to develop efficient phishing page detection techniques that are difficult to bypass. Our solution is based on the aggregation analysis mechanism to automatically generate rules to determine layout similarity of web pages and then detect phishing pages. Our approach consists of two phases. It first trains a similarity classifier using page layout features, then uses the classifier to detect phishing pages.

1.1 Experimental method

We prototyped our approach and evaluated it based on four learning classifiers, namely, Support Vector Machine (SVM), Decision Tree, AdaBoost, and Random Forest. Our evaluation used more than 490 phishing web pages from *phishtank.com* that mimic 46 target pages, from which we extracted over 20,000 testing samples. Using experiment results, we show the strength and weakness of the classifiers in detecting similar pages and analyzed the effective influences caused by the size of dataset and the sample distributions. It also shows that our approach is effective in creating classifiers and detecting phishing pages via page layout similarity.

In summary, we made the following contributions in this paper:

- We propose a learning-based mechanism to evaluate the similarity of web page layouts and identify phishing pages.
- We define the rules to extract and create effective page layout features and develop a phishing page classifier based on four typical learning algorithms, Supporting Vector Machine, Decision Tree, AdaBoost, and Random Forest.
- We prototyped our approach and evaluated it with real-world web page samples from *phishtank.com*. The experiment results illustrate the efficiency of our approach.

1.2 Paper organization

The rest of this paper is organized as follows. We discuss closely related work in Section 2. Section 3 introduces the background of our work and gives an overview of our approach. Section 4 presents our main algorithm. Section 5 presents the evaluation results. We conclude the paper in Section 6.

2 Related work

In this section, we discuss past research work that is closely related to our approach. We focus on phishing detection techniques that are based on page features intrinsic to page visual appearance, instead of external page features, such as URLs.

2.1 Page-feature-based phishing detection

Eric et al. [15] proposed a scheme that selects text pieces, images, and overall visual appearance as the basic properties to compare the similarity of two pages. Chen et al. [16] presented another algorithm to detect visually similar pages according to Gestalt theory, in which they process the webpage as an indivisible entity. CANTINA [6] detects phishing pages based on “term frequency-inverse document frequency (TF-IDF).” SpoofGuard [17] uses *domain name*, *URL*, *link*, and *image* as the critical features to check suspicious pages. GoldPhish [18] uses optical character recognition from a rendered page to extract page information. It then uses search engines to decide whether the page content is consistent with its domain and thus identifies phishing sites. Zhang et al. [19] used spatial layout characteristics from web pages and used as a basis to decide page similarity. Moghimi et al. [20] discovered a rule-based scheme that used two novel feature sets to detect phishing in internet banking. One feature set is used to evaluate the identity of page resources, and the other is utilized to identify the access protocol. Wardman et al. [21] used file-level similarity between two web pages and to detect phishing web sites. Phishing-Alarm uses CSS layout features that are efficient and robust in detecting phishing web sites [8]. In contrast, to identify new features as a basis for phishing detection, this paper focuses on how to automatically learn classifiers of similar pages from CSS features.

2.2 Learning-based phishing detection

Machine learning has been applied to web page classification in detecting phishing. Pan et al. [10] presented an SVM-based page classifier for detection of phishing sites. Xiang et al. [11] proposed CANTINA+ that takes the 15 features from URL, HTML Document Object Model (DOM), third party services, and search engines. It trains these features using Support Vector Machine (SVM) to detect phishing attacks. Abu-Nimeh et al. [22] compared six machine learning algorithms for phishing detection, including Bayesian Additive Regression Trees, Logical Regression, Support Vector Machine, Random Forest, Neural Network, and Regression Tree. Lee et al. [12] leveraged a linear chain CRF model to understand web browsing behaviors of users on phishing web sites and predicted behavior under the context to detect phishing attacks. Abdelhamid et al. [13] proposed an associative classification method for web site phishing detection based

on multi-label classifiers. Bottazzi et al. [23] proposed a framework in Android mobile devices for phishing detection, which includes a machine learning detection engine for protecting from new phishing activities. Abdelhamid et al. [14] investigated several machine-learning-based phishing detection techniques on their pros and cons, including evaluation with real-world dataset on their performance. In our preliminary work [24], we used two classifiers to detect phishing attacks from page layout features.

We summarize the properties of typical learning-based phishing detection approaches and make a comparison with our scheme in Table 1.

3 Background and overview

In this section, we introduce the background for our solution, define the problem, and describe the overall solution.

3.1 Page layout features

Cascading Style Sheets (CSS) is the commonly used visual layout definition of web pages. Widely supported by browsers, CSS rules specify how different classes of web page components should appear, for example, the font type and the color of the body of a page.

In our previous work [8, 9], we have demonstrated that CSS-based page layout features can be used as the basis to detect phishing pages, where we convert CSS into a normalized representation called *influence vector*. It consists of two parts: a *property*, and one or more *declarations*. Each declaration consists of a *value* and one or more *selectors*. In addition, the selectors can be classified into four categories *tag*, *ID*, *class*, and *others*.

For example, given the CSS rule set of a web page,

$$\{ \dots, [Selector_i \{ \dots; [Property_j : Value_k; \dots], \dots \}], \dots \},$$

its influence vector will be defined as

$$\{ \dots, Property_j : [\dots; \{ Value_k^j : [\dots, Selector_i^{j,k}; \dots] \}], \dots \}.$$

where j means the j th property of one page, k means the k th property value in the j th property of the page, and i means the i th selector that has *Property_j* and *Value_k^j*.

Table 1 Comparison of learning-based phishing detection

Detection	Input features	Input samples	Classifiers	Precision
Pan et al. [10]	7 features	About 380	1	***
Xiang et al. [11]	15 features	About 8120	6	****
Abdelhamid et al. [22]	16 features	About 1350	6	**
Mao et al. [24]	1 feature	About 2930	2	**
Our work	1 feature	About 26580	4	**

The precision in the table means the degree of correct detection

*The precision is below 90%

**The precision is in 90–95%

***The precision is in 95–97%

****The precision is in 97–100%

More concretely, from the following CSS rules,

```
div {padding : 2px;},
p {padding : 3px;color : #ff0000},
.class1 {padding : 2px;color : #ff0000},
.class2 {padding : 3px},
#id1 {padding : 2px;color : #ff0000},
#id2 {padding : 3px;color : #00ff00}.
```

the corresponding influence vector will be

$$padding : \left[\begin{array}{l} \{ "2px" : ["div", ".class1", "#id1"] \}, \\ \{ "3px" : ["p", ".class2", "#id2"] \}. \end{array} \right],$$

$$color : \left[\begin{array}{l} \{ "#ff0000" : ["p", ".class1", "#id1"] \}, \\ \{ "#00ff00" : [".class2", "#id2", "#id3"] \}. \end{array} \right].$$

Note that this is a basic form to represent the page features from CSS layouts. Considering the influence impacts, our approach includes additional influence factors of a page layout. For example, if the element size does matter to the detection effect, we will include it into the feature representation.

3.2 Learning-based layout similarity detection

The metric we used in our previous work is mainly based on human experiences and may not comprehensively represent all the statistical similarity properties between page layouts of phishing pages and legitimate pages. Especially, the *threshold*, a critical parameter of that approach, is selected based on the similarity score distribution of the collected samples. As a result, its accuracy heavily relies on the completeness of the sample collection and attackers may craft new phishing pages to bypass the detection.

Our goal is to develop methods that can detect the similarity among two page layouts by comprehensively “considering” layout features. Machine learning mechanisms are typically used in such situations, where they are used to infer similarity models according to the statistical properties retrieved from the training samples.

The problem addressed by our paper can be formulated as follows: Taking a set of labeled benign and malicious pages as inputs, we extract CSS features and identify learning algorithms to detect visually similar pages based on these CSS features. The page similarity will help to detect phishing pages.

3.3 Approach overview

As shown in Fig. 1, our approach includes two phases: *similarity classifier training* and *phishing web page detection based on layout similarity*.

3.3.1 Similarity classifier training

We first obtain a classifier to decide page similarity from layout features. This phase consists of the *pre-processing* stage and the *training* stage. The pre-processing stage takes as inputs two categories of pre-prepared web page

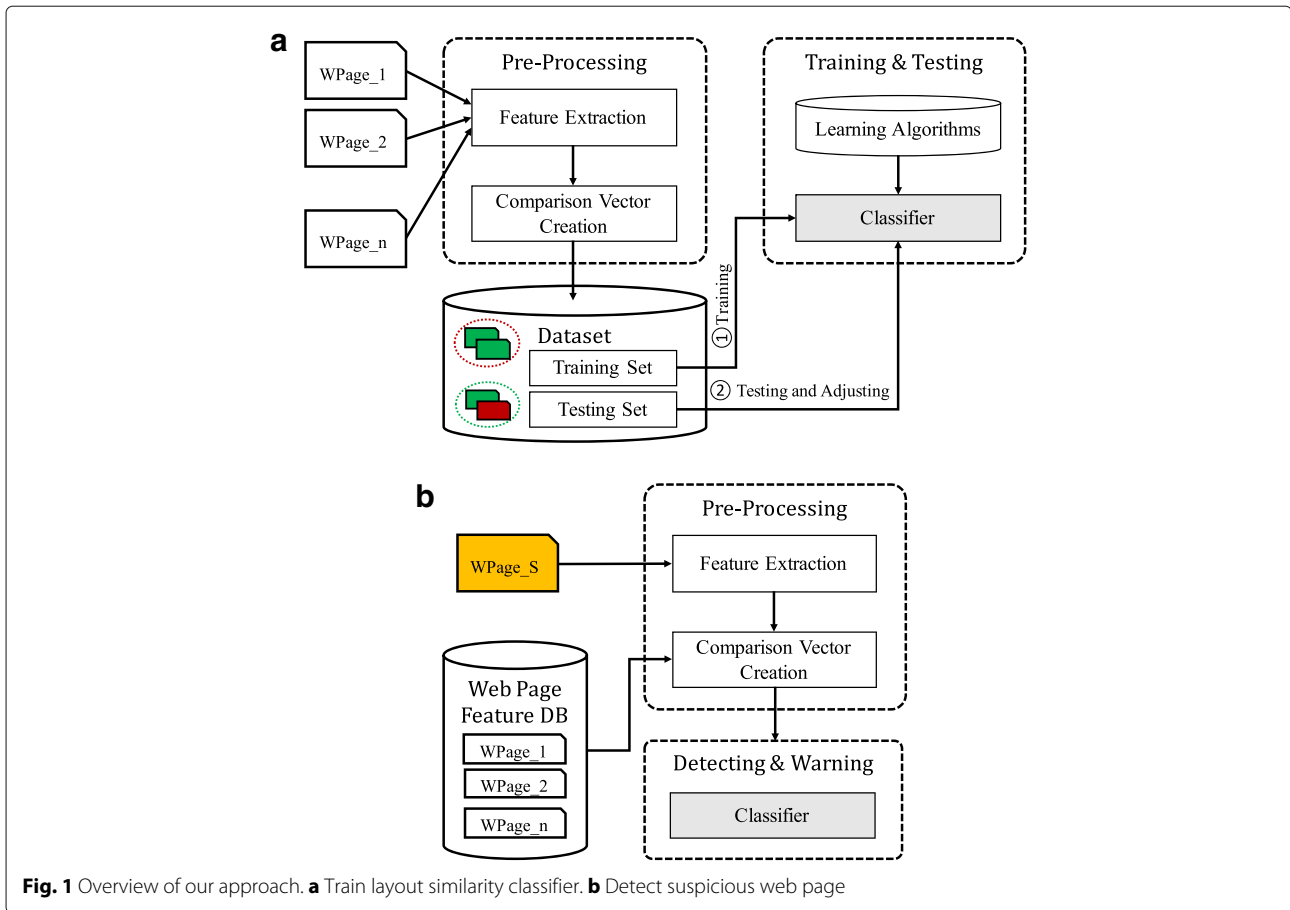


Fig. 1 Overview of our approach. **a** Train layout similarity classifier. **b** Detect suspicious web page

pairs, visually similar web page pairs and visually different web page pairs. Our approach obtains features from web page layouts and creates the *comparison vectors*, which summarize the key similarity features, of every web page pairs accordingly. We label the comparison vector as “ c_1 ” to represent a pair of similar web pages. Correspondingly, the visually different web page pair is labelled as “ c_0 .” The classifier training stage takes as inputs the labelled comparison vectors from the training set. The similar page classifier obtained in this stage can be used to determine whether two web pages are similar according to their comparison vectors.

3.3.2 Phishing web page detection based on layout similarity

The trained classifier can then be used to detect phishing pages. When a user opens a new web page, “WPage_S” (illustrated in a yellow block in Fig. 1b), our detector preprocesses the web page by extracting the layout features of the new page and creating comparison vectors between the “WPage_S” and the pages, “WPage_1, ..., WPage_n” in web page database *Web Page Feature DB*, respectively.

The classifier obtained in phase I takes the comparison vectors as inputs and determines the labels of each vector. If the comparison vector of web page pair “(WPage_S, WPage_i)” is classified as “ c_0 ,” it means “WPage_S” is visually different from “WPage_i” and the system will go to test the next vector. Otherwise, it means “WPage_S” is visually similar as “WPage_i.” Once the classifier outputs a “ c_1 ” labelled vector, the system will send a warning message to alert users.

4 Learning-based similar page layout classification

In this section, we describe the key part of our approach, a learning-based classification module based on CSS features to identify similar web pages. It includes two steps, *property vector extraction* and *classifier building*. We first extract the features of the web pages and combine two pages’ effective CSS features into a comparison property vector. The learning-based classifier training module takes the labelled comparison vectors as inputs, where 1 denotes that two pages are similar and 0 denotes that two pages are visually different. The output of the training

module is a similar page classifier that takes web pages' features as inputs and outputs 1 or 0 to represent similar pages or different pages respectively.

4.1 Property vector extraction

One of the challenges faced by our approach is to extract features from page layouts and represent them into formats that are easily processed by learning techniques. In this step, we present the rules that quantify the CSS elements' impact of a web page and combine CSS features of two pages into one comparison property vector.

4.1.1 Property vector generation

As in our previous work [8, 9], we use the area of elements in a page to demonstrate their impacts on page layout. The larger the area is, the more impact it has on the page layout. As our main goal is to learn classifiers *without* human expertises, instead of manually decide how to use the area information, we extract area properties as a part of the influence vectors extracted from CSS layouts, which we call *property vector* in this paper.

To avoid the inaccuracy of detection in different page window sizes, in this paper, we use the relative area, i.e., the proportion of an element's area to the whole page window size. Because page visual appearance is affected by CSS selectors' properties and values, which are not CSS names of selectors, we associate the area information with properties in the representation. We extract and express CSS features to the pattern shown as follows:

$$[\dots, Property_j \{ \dots; Value^k : AreaInfo_j^k, \dots \}, \dots]$$

where j denotes the j th property in a page and k denotes the k th value in $Property_j$.

Different from the representation used in our past work, we incorporate the relative area size of page elements into the features. We rank the CSS objects in the decreasing order by area proportion in a $Property_j$, i.e., $AreaInfo_j^1 > AreaInfo_j^2 > \dots > AreaInfo_j^n$.

For example, assuming pages have common three properties: $Property_1$ ="height," $Property_2$ ="width," $Property_3$ ="color." There are target Page1 and suspicious Page2. Here is an illustrative example of the vector representation.

```
Page1 :[ "height" { 16px : 0.26, 20px : 0.2 },
        "width" { 344px : 0.2 }, "color" { #ffffff0 : 0.1 } ]
Page2 :[ "height" { 14px : 0.28 },
        "width" { 320px : 0.2 }, "color" { #ffffff : 0.15 } ]
```

One practical challenge is that different pages have different numbers of CSS selectors and declarations. If we want to merge two pages, we should unify the dimension of properties of different pages and then they can be combined. To understand the effective CSS properties used in web page CSS files, we collect all the properties from all the web pages for training and testing and made a statistics, shown in Fig. 2. We make a union set of the properties, denoted by Σ , where $\Sigma = \{Property_1, Property_2, \dots, Property_k\}$. We make the length of the union set $|\Sigma| = k$ as the dimension of the property vector. So, we can unify effective CSS features of one page into the following pattern:

$$Page_i [Property_1 \{ \dots \}, Property_2 \{ \dots \}, \dots, Property_k \{ \dots \}]$$

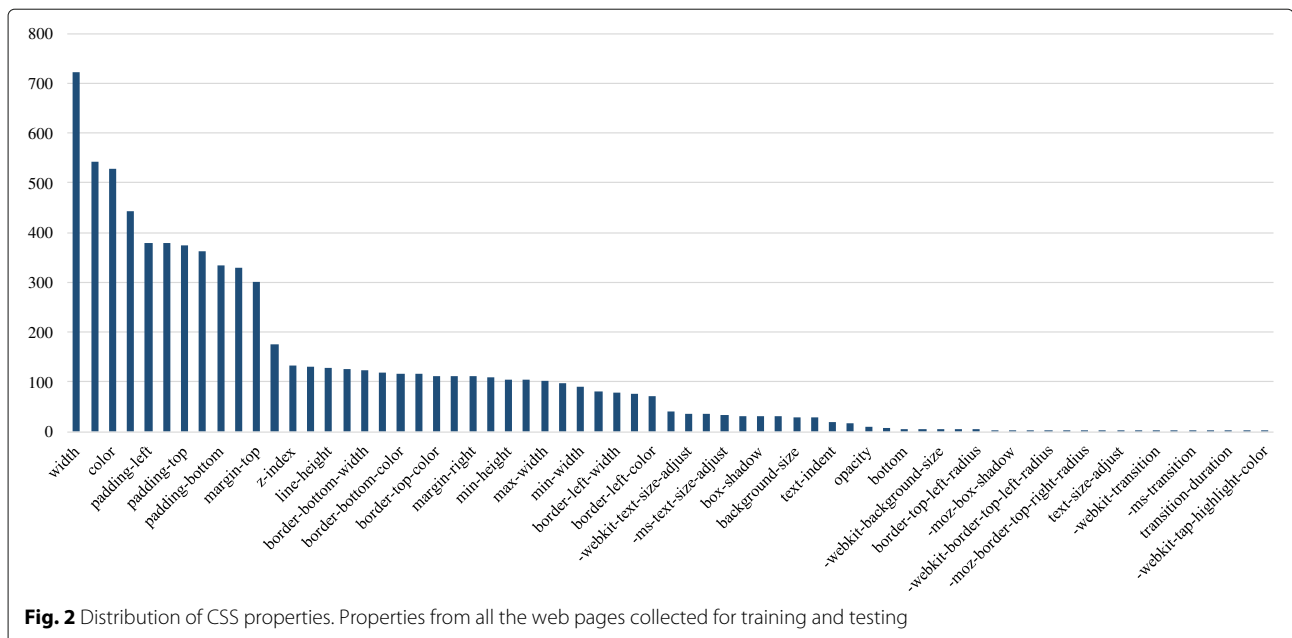


Fig. 2 Distribution of CSS properties. Properties from all the web pages collected for training and testing

Different pages have the same k property expressed as above. For simplicity, we can represent the pattern as follows:

$$\begin{aligned} & \text{Page1} \left[P_1 \left\{ V_1^1 : A_1^1, \dots, V_1^{m_1} : A_1^{m_1} \right\}, \right. \\ & \quad P_2 \left\{ V_2^1 : A_2^1, \dots, V_2^{m_2} : A_2^{m_2} \right\}, \dots, \\ & \quad \left. P_k \left\{ V_k^1 : A_k^1, \dots, V_k^{m_k} : A_k^{m_k} \right\} \right] \end{aligned}$$

where P denotes the *property*, V denotes the *value*, A denotes the *area proportion*. Different *properties* may have different numbers of *values*.

In order to quantify CSS features, we should transfer their property values into computable type. So we do some simplified encoding in property values. For examples, we transfer *color:#ffffff* to *color:(255,255,255)* and take away units of *width:16px* to get *width:16*.

4.1.2 Comparison vector generation

Given the two pages, we quantify their common CSS features into a comparison vector. The procedure is as follows:

- We first unify the property values of the same property into the same dimension. For the same property P_k , if Page1 has m_1 values $\{V_k^1 : A_k^1, \dots, V_k^{m_1} : A_k^{m_1}\}$ and Page2 has m_2 values $\{V_k^1 : A_k^1, \dots, V_k^{m_2} : A_k^{m_2}\}$. We choose the larger value of m_1 and m_2 , denoted by m , and extend the page property of the smaller one to the length of m by adding zeros. The outputs in this step are Page1 $\{V_k^1 : A_k^1, \dots, V_k^m : A_k^m\}$ and Page2 $\{V_k^1 : A_k^1, \dots, V_k^m : A_k^m\}$ with the same dimension.
- We compute the difference between Page1 : V_k^i and Page2 : V_k^i where $i \in m$ and use the maximum value of Page1 : A_k^i and Page2 : A_k^i to multiply the difference value. The result is denoted by ε_k^i . $\varepsilon_k^i = |\text{Page1} : V_k^i - \text{Page2} : V_k^i| \times \max(\text{Page1} : A_k^i, \text{Page2} : A_k^i)$. Then, we get a value in i th $\{V_k : A_k\}$ as the i th dimension of their comparison property vector.
- We calculate all the ε_k^i of P_k and obtain $\varepsilon_k = \text{sum}(\varepsilon_k^i, \text{where } i = 1, 2, \dots, m)$.
- After repeating the previous steps k times, we finally get the comparison property vector of Page1 and Page2 denoted as $[\varepsilon_1, \varepsilon_2, \dots, \varepsilon_k]$.

For the above example, after apply simplified encoding, the features become:

$$\begin{aligned} & \text{Page1} : [16 : 0.26, 20 : 0.2, 344 : 0.2, (255, 255, 240) : 0.1] \\ & \text{Page2} : [14 : 0.28, 0 : 0, 320 : 0.2, (255, 255, 255) : 0.15] \\ & \varepsilon_1 = |16 - 14| \times 0.28 + |20 - 0| \times 0.2 = 4.56 \\ & \varepsilon_2 = |344 - 320| \times 0.2 = 4.8 \\ & \varepsilon_3 = (|255 - 255| + |255 - 255| + |240 - 255|) \times 0.15 = 2.25 \end{aligned}$$

So, the common property vector is: $[4.56, 4.8, 2.25]$. After representing all page features into comparison vectors, they are ready to be processed by the learning algorithms.

Table 2 Comparison of the four classification algorithms

Classifier	Robustness	Efficiency	Dataset scale
SVM	ooo	oo	*
DT	o	ooo	*
AB	oo	o	**
RF	ooo	oo	**

"o" represents the performance level

**" represents the scale of affordable dataset

4.2 Classifier building

We consider our approach as a two-category classification problem. We set the output of the classifier as a binary output, 1 or 0, and make the comparison property vectors in the dataset as inputs. We divide the dataset into two parts. One is used to train the classifier, and the other is the testing set used to evaluate the performance. Let $\Gamma_1 = \{x_i\}_{i=1}^m$ be a set of M training vectors, where x_i is a k -dimension vector labelled by $y_i \in \{\pm 1\}$, with $y_i = 1$ and $y_i = -1$ indicating x_i to the class 1 and class 2 respectively. And $\Gamma_2 = \{x_i\}_{i=1}^n$ be a set of N testing vectors.

We use the following four classifiers in our approach, including Support Vector Machine (SVM) [25, 26], Decision Tree (DT) [27, 28], AdaBoost (AB) [29, 30], and Random Forest (RF) [31, 32]. The property comparison of the four classification algorithms is summarized in Table 2, with detailed explanation as follows.

- *Support Vector Machine (SVM)*. SVM aims to maximize the margin between classes closest points to find an optimal separating hyperplane between them. The minority of support vectors (SV) produced after training determines the result of classifiers, which avoids dimension disaster and offers a good performance in robustness.
- *Decision tree (DT)*. DT classifies items by making decisions at each branch to obtain as much as entropy gain as possible. A decision tree consists of a root node, several internal nodes, and leaf nodes. Leaf nodes denote the result of the classifier, and other nodes denote each attribute. Every route from the root node to a leaf node corresponds a determining test sequence. It follows the rule of divide-and-conquer.
- *AdaBoost (AB)*. Boosting is a kind of ensemble learning algorithms that promote weak learner to

Table 3 Dataset for classifier

Source	PhishTank	
Dataset	Positive samples	Negative samples
Training set	3719	17926
Testing set	414	1992

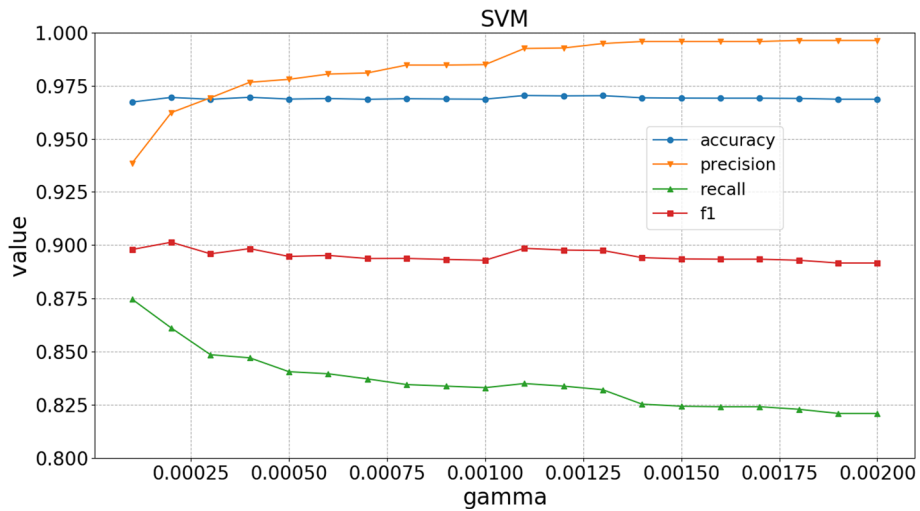


Fig. 3 Result of SVM with regard to gamma. According to the experiment results shown in this figure, the accuracy is around 96%, while the rest of the three metrics are mostly above 80%. With the rising of gamma, the value of accuracy and F1 almost remain constant, while recall falls down a little and precision goes up a little. When gamma is about 0.0002, the four metrics get close to the best performances

strong learner. AB is a representative of this kind of boosting. Its training starts with a base learner and adjusts the distribution of samples based on the performance of the base learner. Then, it trains the next base learner based on the adjusted distribution of samples iteratively. Their outputs are given different weights that contribute to the final output of the boosted classifier. It is a kind of serial ensemble algorithm.

- *Random Forest (RF)*. Different from boosting, Bagging is a parallel ensemble learning algorithm. It

samples different sets form the training set, trains base learners based on these different sample sets, and combines the base learners to produce a good result. RF is an expansion of Bagging technique that builds lots of decision trees for training and outputs the most-voting class. It introduces the random attribute selective to make stronger generalization.

In our approach, we use Γ_1 to train a classifier model and use Γ_2 to test its performance. When the input of a comparison property vector gets output 1, it means

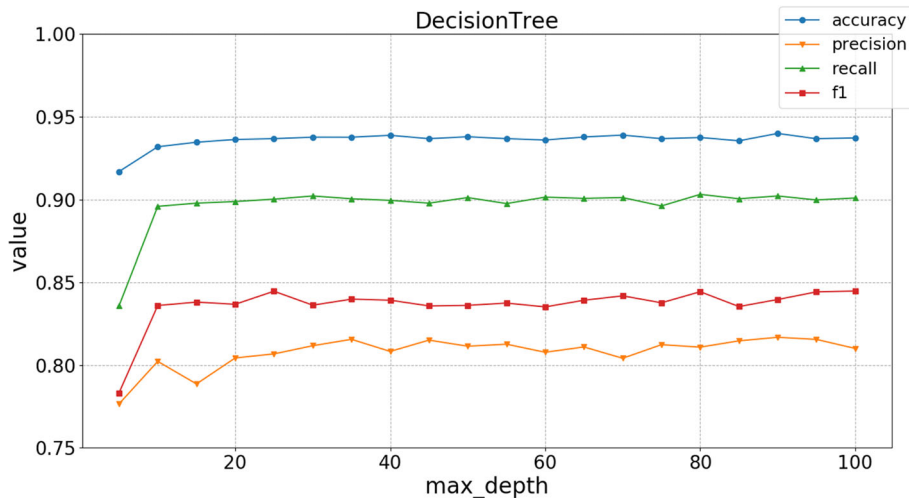


Fig. 4 Result of Decision Tree with regard to depth of tree. According to the experiment results, the four metrics remain constant when *max_depth* is above 20, and their values may fluctuate a little. The accuracy is about 93%, while the precision is the lowest, which is around 80%. When *max_depth* is about 25, the four metrics achieve the best

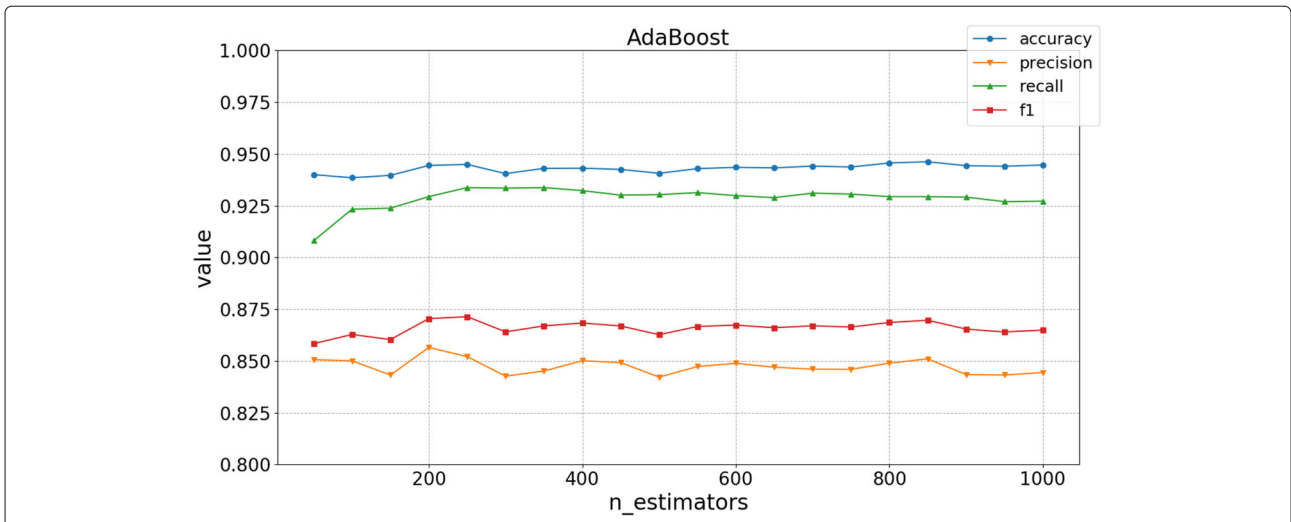


Fig. 5 Result of AdaBoost with regard to $n_estimators$. The four metrics displayed in this figure are above 82.5%, and their values increase slightly when the value of $n_estimators$ increases. The accuracy is close to 94%. When $n_estimators$ is about 250, the system obtains a relatively optimal performance

the two pages are similar. The suspicious page will be determined to be malicious. When the input of a comparison property vector gets output 0, it means the two pages are not similar. The suspicious page will be determined to be benign. We evaluate four classifiers in the next section.

5 Evaluation

In this section, we evaluate our approach. In order to evaluate the effectiveness of our solution, we deploy several machine learning classifiers to evaluate the performance.

We use four metrics *accuracy*, *precision*, *recall*, and *F1 score*, to analyze the results of our approach. Accuracy equals to the proportion of the number of web pages that are correctly detected as phishing pages or normal pages to the number of total sample web pages. Precision equals to the proportion of the number of web pages that are correctly detected as phishing pages to the number of total detected web pages. Recall equals to the proportion of the number of web pages that are correctly detected as phishing pages to the number of total phishing samples.

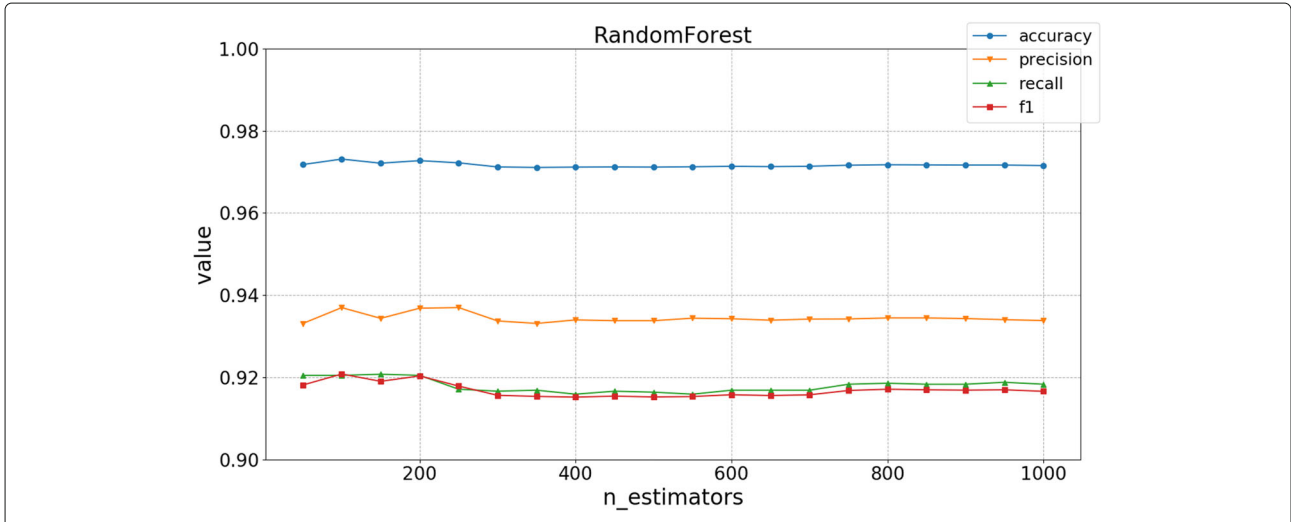
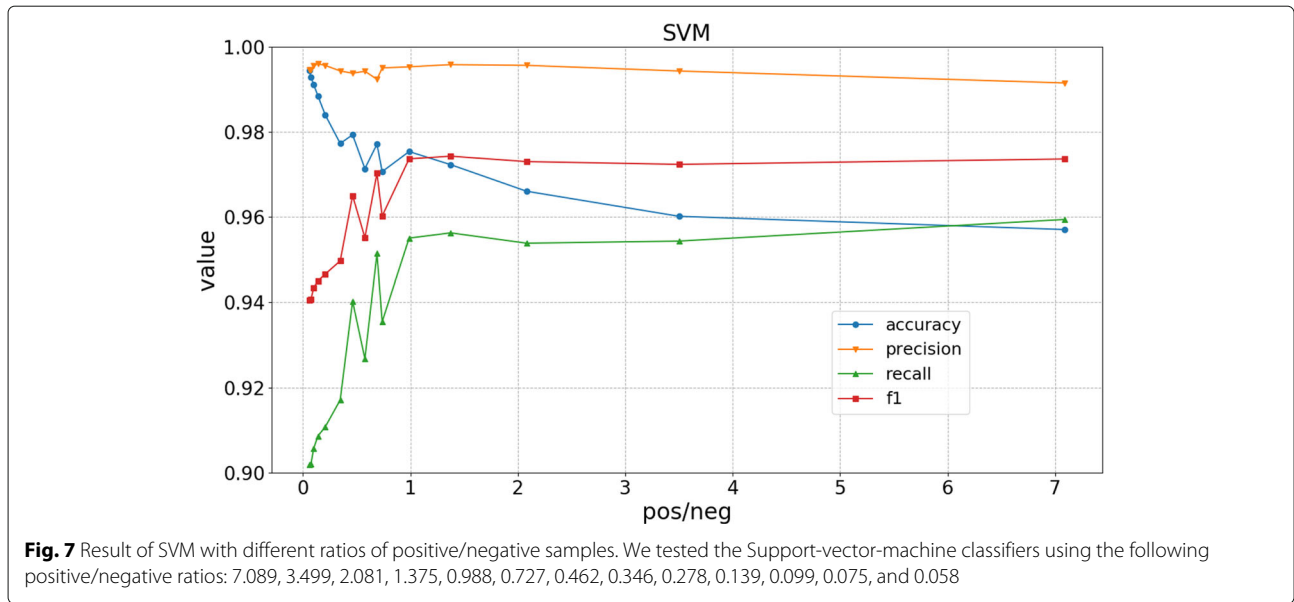


Fig. 6 Result of Random Forest with regard to $n_estimators$. The accuracy is above 96%, and the rest of the three metrics are above 90% and their values keep nearly stable over different values of $n_estimators$. The system gets a better performance, when $n_estimators$ is about 100



The accuracy, precision, and recall (as shown in Eqs. (1), (2), and (3)) are calculated the same as in [9].

$$\text{Precision} = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

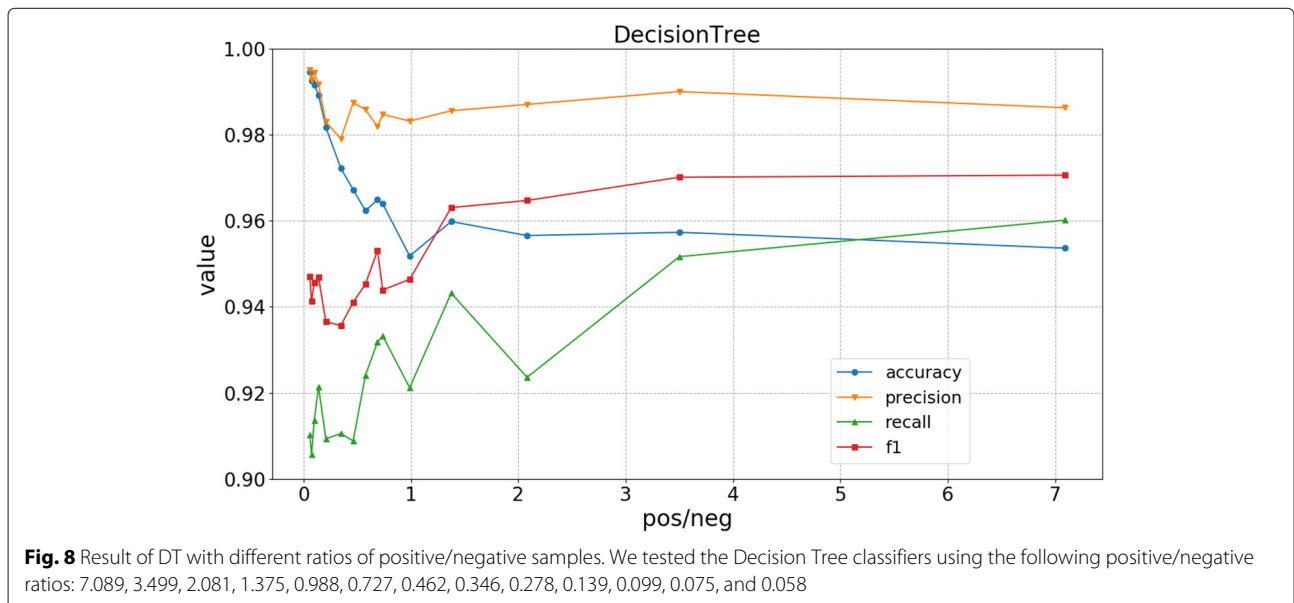
$$\text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

True positive (TP) is the number of correctly classified phishing pages. True negative (TN) is the number

of correctly classified legitimate pages. False negative (FN) is the number of phishing pages misclassified as legitimate pages. False positive (FP) is the number of legitimate pages misclassified as phishing pages. Besides, we use F1 score (Eq. (4)) as a metric to evaluate our approach.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

Dataset preparation. We collect phishing websites from *phishtank.com*. We first check and filter those invalid



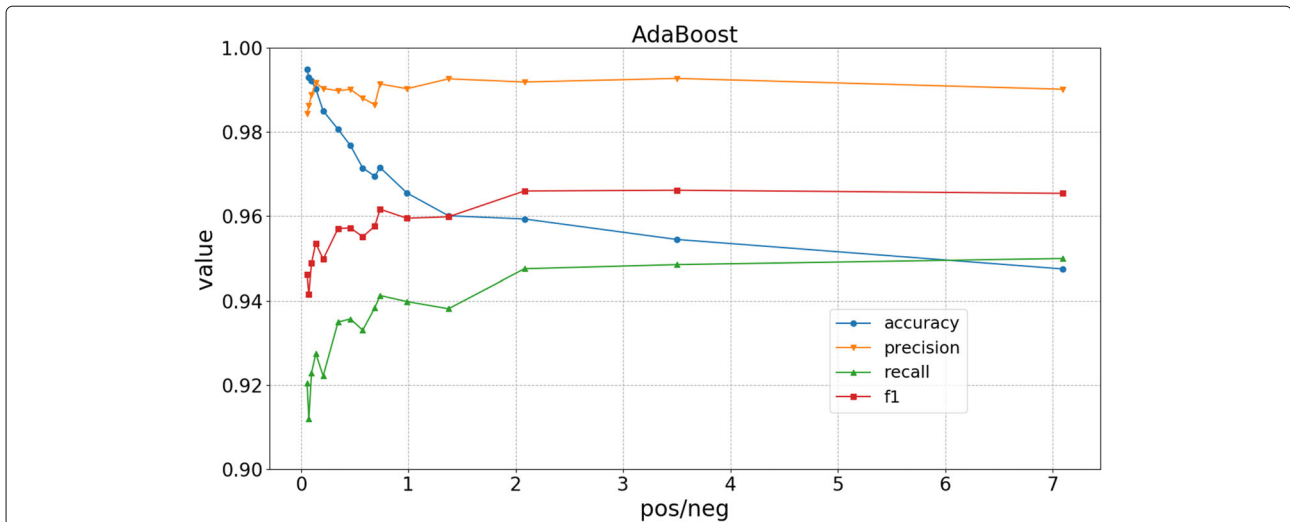


Fig. 9 Result of AB with different ratios of positive/negative samples. We tested the AdaBoost classifiers using the following positive/negative ratios: 7.089, 3.499, 2.081, 1.375, 0.988, 0.727, 0.462, 0.346, 0.278, 0.139, 0.099, 0.075, and 0.058

pages manually. We then exclude the pages whose layout elements are too small and whose layout appearance is totally different from their target. We select 46 target pages, 448 suspicious pages, and 40 normal pages different from target pages to test our approach. In property vector extraction, we obtain 4133 comparison vectors as positive samples whose label is set to 1 and 19918 comparison vectors as negative samples whose label is set to 0. Positive samples consist of pairs of target pages and corresponding similar suspicious pages. Negative samples consist of pairs of target pages and corresponding dissimilar suspicious pages, pairs of

normal pages and suspicious pages, and pairs of normal pages and target pages. There are 24051 samples in total to evaluate our four classifiers, shown in Table 3.

5.1 Classifier effectiveness

We first evaluate the classifiers’ effectiveness under different parameters. In these experiments, we use all of our effective 24051 samples mentioned above to evaluate and ignore the unbalance of positive and negative samples, which we will analyze the impact in the next experiment. The results are as follows:

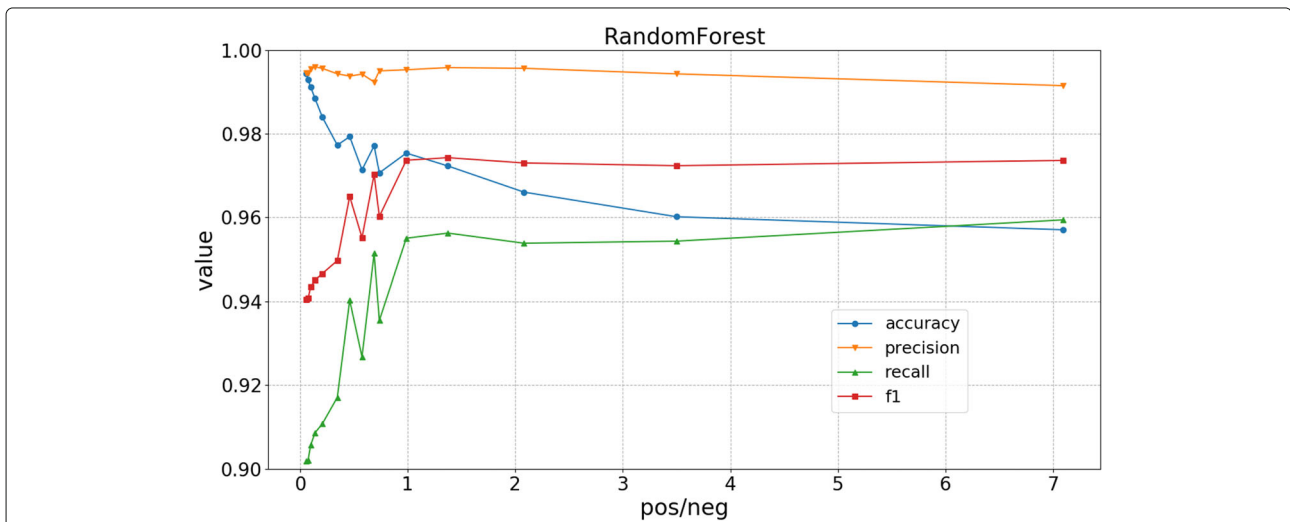


Fig. 10 Result of RF with different ratios of positive/negative samples. We tested the Random Forest classifiers using the following positive/negative ratios: 7.089, 3.499, 2.081, 1.375, 0.988, 0.727, 0.462, 0.346, 0.278, 0.139, 0.099, 0.075, and 0.058

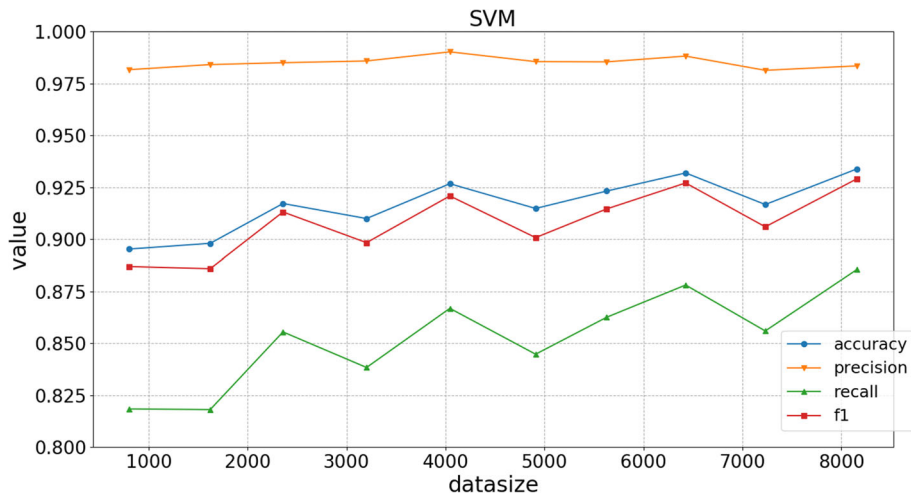


Fig. 11 Result of SVM with different training set size. We test the SVM classifier using subsets with the following sizes: 803, 1622, 2357, 3201, 4043, 4909, 5628, 6425, 7230, and 8155, where the ratio of positive/negative is close to 1. The accuracy is high, which is above 95%. With the increase of data size, the results get better

- Support Vector Machine (SVM). We employ SVM as the classifier and test four metrics regarding the parameter γ in the SVM algorithm. According to the experiment results shown in Fig. 3, the accuracy is around 96%, while the rest three metrics are mostly above 80%. With the rising of γ , the value of accuracy and F1 almost remain constant, while recall falls down a little and precision goes up a little. When γ is about 0.0002, the four metrics get close to their best performance.
- Decision Tree (DT). We employ DT as the classifier and test four metrics regarding the parameter

max_depth in the DT algorithm. The results are shown in Fig. 4, where the four metrics remain constant when max_depth is above 20, and their values may fluctuate a little. The accuracy is about 93%, while the precision is the lowest, which is around 80%. When max_depth is about 25, the four metrics achieve the best.

- AdaBoost (AB). We employ AB as the classifier and test four metrics regarding the parameter $n_estimators$ in the AB algorithm. The four metrics displayed in Fig. 5 are above 82.5%, and their values increase slightly when the value of $n_estimators$

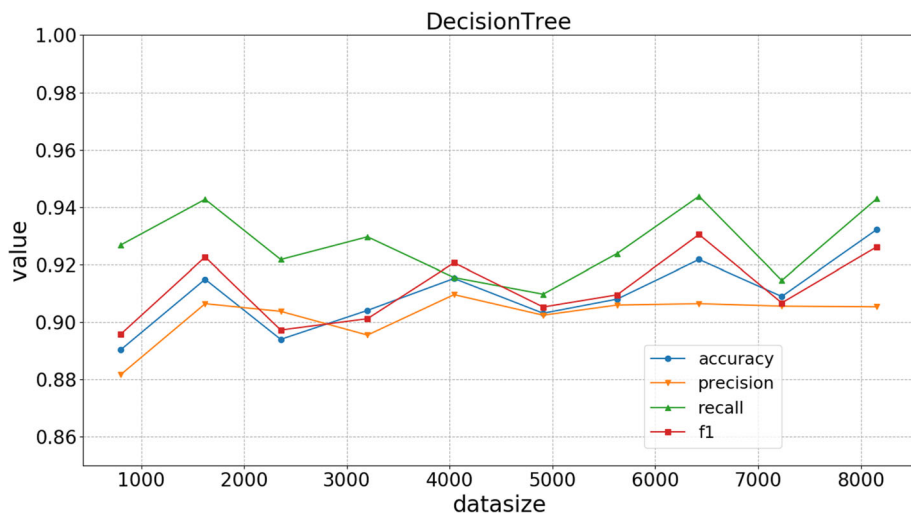


Fig. 12 Result of DT with different training set size. We test the Decision Tree classifier using subsets with the following sizes: 803, 1622, 2357, 3201, 4043, 4909, 5628, 6425, 7230, and 8155, where the ratio of positive/negative is close to 1. The accuracy is high, which is above 95%. With the increase of the data size, the results get better

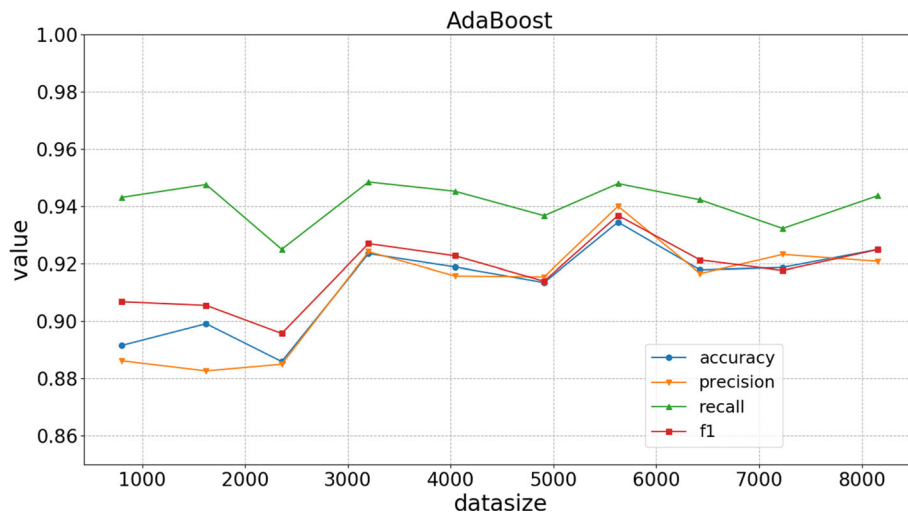


Fig. 13 Result of AB with different training set size. We test the AdaBoost Decision Tree classifier using subsets with the following sizes: 803, 1622, 2357, 3201, 4043, 4909, 5628, 6425, 7230, and 8155, where the ratio of positive/negative is close to 1. The accuracy is high, which is above 95%. With the increase of the data size, the results get better

increases. The accuracy is close to 94%. When $n_estimators$ is about 250, the system obtains a relatively optimal performance.

- Random Forest (RF). We employ RF as the classifier and test four metrics regarding the parameter $n_estimators$ in the RF algorithm. Figure 6 gives the experiment results, and we can see that the accuracy is above 96%, and the rest of the three metrics are above 90% and their values keep nearly stable over different values of $n_estimators$. The system gets a better performance, when $n_estimators$ is about 100.

5.2 Effectiveness of positive-negative sample distributions

Here, we evaluate the effect of the ratio of positive/negative samples. We change the number of negative samples to control the ratio. We tested the classifiers using the following positive/negative ratios: 7.089, 3.499, 2.081, 1.375, 0.988, 0.727, 0.462, 0.346, 0.278, 0.139, 0.099, 0.075, and 0.058. The results are shown in Figs. 7, 8, 9, and 10. The accuracy decreases with the increase of the ratio, while all three other metrics increase. A ratio of 1 to 2 is recommended.

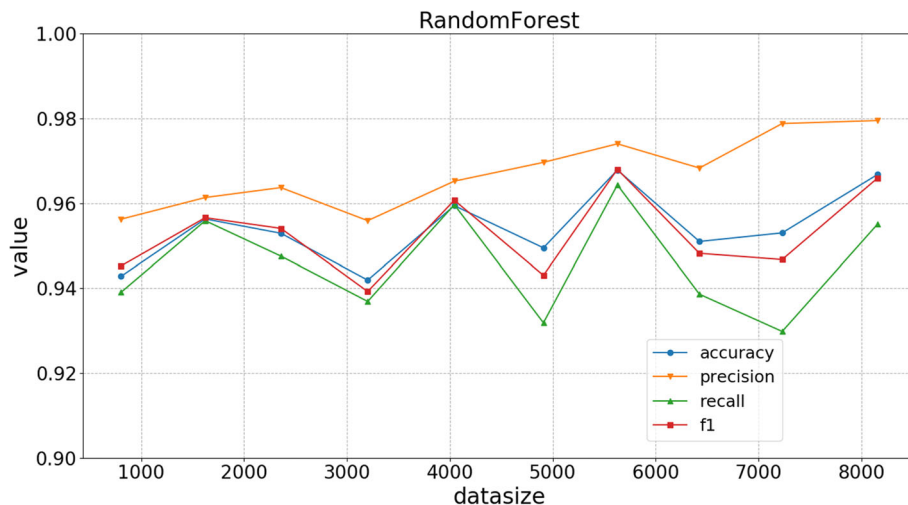


Fig. 14 Result of RF with different training set size. We test the Random Forest classifier using subsets with the following sizes: 803, 1622, 2357, 3201, 4043, 4909, 5628, 6425, 7230, and 8155, where the ratio of positive/negative is close to 1. The accuracy is high, which is above 95%. With the increase of the data size, the results get better

Table 4 Test results of the four classifiers

Classifier	Accuracy	Precision	Recall	F1
SVM	0.96948	0.96235	0.86115	0.90134
DT	0.93676	0.80674	0.90015	0.84467
AB	0.94500	0.85218	0.93378	0.87145
RF	0.97310	0.93695	0.92046	0.92078

5.3 Sensitivity to size of training set

Finally, we evaluate how the size of the training set affects the detection results. Here, we test classifiers using subsets with the following sizes: 803, 1622, 2357, 3201, 4043, 4909, 5628, 6425, 7230, and 8155, where the ratio of positive/negative is close to 1. The results are shown in Figs. 11, 12, 13, and 14. The accuracy is above 95%. With the increase of data size, SVM and Random Forest explicitly improve their performance, while Decision Tree and AdaBoost have implicit tendency under the distribution of the testing samples.

5.4 Results and discussion

According to the experiment results, we present the best performance values of each classifier in Table 4. Among these four classifiers, Random Forest performs the best by considering all the four metrics. All the classifiers show more than 93% accuracy and more than 84% F1, which demonstrates that our approach can make an effective detection in phishing websites.

Table 5 illustrates three metrics of our work and four other approaches (CANTINA [6], CANTINA+ [11], Corbetta et al. [33], and Zhang et al. [19]).

Although the metrics of our approach is not the best, it still performs better than Corbetta et al. [33] and Zhang et al. [19]. However, with respect to other approaches, our method is light-weight as it only takes one class of features, CSS structure, as the input to identify the similarity of web pages and detect phishing attacks. Moreover, our method is independent of the language of web pages. In addition, according to the evaluation conducted in Sections 5.2 and 5.3, the accuracy and robustness of such learning-based solutions are greatly influenced/limited by the size of the dataset and the distribution of the testing

Table 5 The precision, recall, and F1 score of our work and other approaches

Approaches	Precision (%)	Recall (%)	F1
CANTINA [6]	94.2	97.0	0.956
CANTINA+ [11]	97.5	93.47	0.963
Corbetta et al. [33]	95.3	73.08	0.827
Zhang et al. [19]	91.0	91.90	0.915
Our work	93.7	92.05	0.921

samples. More testing samples and the adjustment of classifier parameters will promote our results.

6 Conclusion

In phishing web site detection, comprehensively evaluating page similarity remains a great challenge. In this paper, we propose a learning-based aggregation analysis mechanism to determine similarity of page layouts and detect phishing pages. Our approach automatically trains classifiers to determine web page similarity from CSS layout features, which does not require human expertise. We prototyped our approach and evaluated it using a large amount of phishing web pages. The experiment results demonstrate that our approach is accurate and effective in determining similarity from page layouts. Our approach can effectively enhance the performance of existing anti-phishing mechanisms.

Abbreviations

AB: AdaBoost; CSS: Cascading style sheets; DT: Decision tree; IoT: Internet of things; RF: Random forest; SVM: Support vector machine

Acknowledgements

Not applicable.

Funding

This work was supported in part by the National Key R&D Program of China (No. 2017YFB0802400), the National Natural Science Foundation of China (No. 61402029, No. 61471028, No. U11733115), the Funding Project of Shanghai Key Laboratory of Integrated Administration Technologies for Information Security (No. AGK201708), the Singapore Ministry of Education under National University of Singapore (NUS) Grant R-252-000-666-114.

Availability of data and materials

Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Authors' contributions

The authors have contributed jointly to the manuscript. All authors have read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details

¹School of Cyber Science and Technology, Beihang University, Xueyuan Road, Beijing 100083, China. ²School of Electronic and Information Engineering, Beihang University, Xueyuan Road, Beijing 100083, China. ³Baidu USA LLC, Bordeaux Drive, Sunnyvale 94089, USA. ⁴Information Technology Service Center, China National Petroleum Corporation, Beijing, China. ⁵School of Computing, National University of Singapore, Lower Kent Ridge Road, Singapore 117417, Singapore.

Received: 13 September 2018 Accepted: 29 January 2019

Published online: 20 February 2019

References

1. APWG, Statistical highlights for 4th quarter 2016. http://docs.apwg.org/reports/apwg_trends_report_q4_2016.pdf. Accessed 26 July 2018
2. P. Likarish, E. Jung, D. Dunbar, T. E. Hansen, J. P. Hourcade, in *Proceedings of IEEE International Conference on Communications, ICC'08*. B-apt: Bayesian anti-phishing toolbar (IEEE, Beijing, 2008), pp. 1745–1749

3. Cloudmark Inc., Cloudmark toolbar (2018). <https://www.cloudmark.com/en/products/cloudmark-safe-messaging-cloud>. Accessed 26 July 2018
4. T. Ronda, S. Saroiu, A. Wolman, itrustpage: a user-assisted anti-phishing tool. *ACM SIGOPS Oper. Syst. Rev.* **42**(4), 261–272 (2008)
5. I. Fette, N. Sadeh, A. Tomasic, in *Proceedings of the 16th International Conference on World Wide Web*. Learning to detect phishing emails (ACM, Banff, Alberta, Canada, 2007), pp. 649–656
6. Y. Zhang, J. I. Hong, L. F. Cranor, in *Proceedings of the 16th International Conference on World Wide Web*. Cantina: a content-based approach to detecting phishing web sites (ACM, Banff, Alberta, Canada, 2007), pp. 639–648
7. A. Nourian, S. Ishtiaq, M. Maheswaran, in *Proceedings of 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing, Washington, DC, USA*. Castle: A social framework for collaborative anti-phishing databases (IEEE, 2009), pp. 1–10
8. J. Mao, W. Tian, P. Li, T. Wei, Z. Liang, in *Proceedings of the 12th International Conference on Wireless Algorithms, Systems, and Applications*. Phishing website detection based on effective css features of web pages (Springer, Guilin, 2017), pp. 804–815
9. J. Mao, W. Tian, P. Li, T. Wei, Z. Liang, Phishing-alarm: robust and efficient phishing detection via page component similarity. *IEEE Access.* **5**, 17020–17030 (2017)
10. Y. Pan, X. Ding, in *Proceedings of the 22nd Computer Security Applications Conference(ACSAC)*. Anomaly based web phishing page detection (IEEE, Miami Beach, Florida, 2006), pp. 381–392
11. G. Xiang, J. Hong, C. P. Rose, L. Cranor, Cantina+: a feature-rich machine learning framework for detecting phishing web sites. *ACM Trans. Inf. Syst. (TISSEC)*. **14**(2), 1–28 (2011)
12. L. Lee, K. Lee, Y. Juan, H. Chen, Y. Tseng, in *Proceedings of the 23rd International Conference on World Wide Web*. Users' behavioral prediction for phishing detection (ACM, Seoul, 2014), pp. 337–338
13. N. Abdelhamid, A. Ayesh, F. Thabtah, Phishing detection based associative classification data mining. *Expert Syst. Appl.* **41**(13), 5948–5959 (2014)
14. N. Abdelhamid, F. Thabtah, H. Abdel-Jaber, in *Proceedings of IEEE International Conference on Intelligence and Security Informatics*. Phishing detection: a recent intelligent machine learning comparison based on models content and features (IEEE, Beijing, China, 2017), pp. 72–77
15. E. Medvet, E. Kirda, C. Kruegel, in *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks*. Visual-similarity-based phishing detection (ACM, Istanbul, 2008), pp. 1–6
16. T.-C. Chen, S. Dick, J. Miller, Detecting visually similar web pages: application to phishing detection. *ACM Trans. Internet Technol.* **10**(2), 1–38 (2010)
17. N. Chou, R. Ledesma, Y. Teraguchi, D. Boneh, J. C. Mitchell, in *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS)*. Client-side defense against web-based identity theft (Internet Society, San Diego, 2004), pp. 1–16
18. M. Dunlop, S. Groat, D. Shelly, in *Proceedings of the 5th International Conference on Internet Monitoring and Protection (ICIMP)*. Goldphish: using images for content-based phishing analysis (IEEE, Barcelona, 2010), pp. 123–128
19. W. Zhang, H. Lu, B. Xu, H. Yang, Web phishing detection based on page spatial layout similarity. *Informatica.* **37**(3), 231–244 (2013)
20. M. Moghimi, A. Y. Varjani, New rule-based phishing detection method. *Expert Syst. Appl.* **53**, 231–242 (2016)
21. B. Wardman, T. Stallings, G. Warner, A. Skjellum, in *eCrime Researchers Summit*. High-performance content-based phishing attack detection (IEEE, San Diego, California, 2011), pp. 1–9
22. S. Abu-Nimeh, D. Nappa, X. Wang, S. Nair, in *Proceedings of the Anti-phishing Working Groups 2nd Annual eCrime Researchers Summit*. A comparison of machine learning techniques for phishing detection (ACM, Pittsburgh, Pennsylvania, 2007), pp. 60–69
23. G. Bottazzi, E. Casalicchio, D. Cingolani, F. Marturana, M. Piu, in *Proceedings of the 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM)*. MP-Shield: a framework for phishing detection in mobile devices (IEEE, Liverpool, 2015), pp. 1977–1983
24. J. Mao, J. Bian, W. Tian, S. Zhu, T. Wei, A. Li, Z. Liang, in *Proceedings of International Conference On Identification, Information and Knowledge in the Internet of Things (IIKI)*. Detecting phishing websites via aggregation analysis of page layouts (Elsevier, Qufu, China, 2017), pp. 224–230
25. C. Cortes, V. Vapnik, Support-vector networks. *Mach. Learn.* **20**(3), 273–297 (1995)
26. Wikipedia, Support vector machine (2018). https://en.wikipedia.org/wiki/Support_vector_machine/, [Online]. Accessed 26 July 2018
27. L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, *Classification and Regression Trees*. (Wadsworth and Brooks/Cole Advanced Books and Software, Monterey, CA, 1984)
28. Wikipedia, Decision tree learning (2018). https://en.wikipedia.org/wiki/Decision_tree_learning/, [Online]. Accessed 26 July 2018
29. Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting. *Comput. J., Syst. Sci.* **55**(1), 119–139 (1997)
30. Wikipedia, AdaBoost (2018). <https://en.wikipedia.org/wiki/AdaBoost/>, [Online]. Accessed 26 July 2018
31. L. Breiman, Random forests. *Mach. Learn.* **45**(1), 5–32 (2001)
32. Wikipedia, Random forest (2018). https://en.wikipedia.org/wiki/Random_forest/, [Online]. Accessed 26 July 2018
33. J. Corbetta, L. Invernizzi, C. Kruegel, G. Vigna, in *Proceedings of International Symposium on Research in Attacks, Intrusions, and Defenses*. Eyes of a human, eyes of a program: leveraging different views of the web for analysis and detection (Springer, Gothenburg, 2014), pp. 130–149

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- Convenient online submission
- Rigorous peer review
- Open access: articles freely available online
- High visibility within the field
- Retaining the copyright to your article

Submit your next manuscript at ► [springeropen.com](https://www.springeropen.com)
