UDC 004.056:004.89

Machine Learning Models Stacking in the Malicious Links Detecting

Yevhenii Khukalenko[1], Iryna Stopochkina[1], Mykola Ilin[1]

[1] *National Technical University of Ukraine "Igor Sikorsky KPI", Beresteyskyi ave., 37, Kyiv, 03056, Ukraine Solomyanskyi district,*

Abstract

An analysis of the performance of various classifiers on address and network groups of features was performed. A new classification model is proposed, which is a stacking of 3 models: kNN, XGBoost and Transformer. The best model for stacking was experimentally determined: Logistic Regression, which made it possible to improve the result of the best available model by 3%. The hypothesis that stacking a larger number of worse models has an advantage over stacking a smaller number of more productive models on the used data set was confirmed: regardless of the choice of stacking meta-algorithm, stacking of three models showed better results than stacking two.

Keywords: Malicious URL, Machine Learning, Stacking

Introduction

A large number of attacks are implemented using compromised URLs [1]. Attackers are constantly improving their methods, using a variety of techniques to spoof and mask malicious URLs. Traditional methods of malicious links detecting, such as spam filters and antivirus programs, often fail to detect new attacks or are too sensitive to false positives. As a result, the need to develop for detecting malicious links remains relevant. Compromised URLs used for cyber attacks are called malicious URLs. A large proportion of all websites are potentially malicious in nature [2]. Popular types of attacks using malicious URLs include autoloading, phishing and social engineering, and spam [3].

This paper provides a critical analysis of existing solutions to this problem and proposes a combination of several of the most successful malicious link detection models using stacking to improve accuracy.

1. Characteristics of URLs

1.1. Types of malicious links

According to the classification of Palo-Alto Networks [4], the following types of malicious (clearly or potentially) links can be distinguished:

- command-and-control (C2);
- sites used by malicious software;
- phishing resources;.
- grayware – potentially harmful;
- dynamic-dns – potentially unreliable;
- resources infringing copyright;
- web resources that promote extremism;
- anonymizers – potentially harmful;
- registered by individuals (parked) – potentially harmful.

1.2. Malicious URLs detecting

The most common method used by many antivirus groups to detect malicious URLs is the blacklist method. Blacklists are essentially a database of URLs that have been confirmed to be malicious in the past. This database is collected over time (often through crowdsourcing, e.g. PhishTank[5]) when a URL is known to be malicious.

The advantages of the approach are speed and the absence of the need for additional calculations. ease of implementation and use and low false-positive rate (although there is evidence that blacklisting sometimes suffers from non-trivial false-positive responses [6]).

The disadvantage of the approach is the impossibility of maintaining an exhaustive list of malicious URLs, as new URLs are generated every day. Attackers use special techniques to avoid blacklists by changing the URL to appear legitimate through obfuscation. Four types of obfuscations are provided in [7]:

• obfuscation of the host by IP address;

• obfuscation of the host by another domain;

• obfuscation of the host with large host names;

• masquerading as legitimate domains (writing with errors).

Attackers use many other methods to avoid blacklists, including: fast-flux, which automatically generates proxies to host a web page, algorithmic generation of new URLs, etc. [8]. Additionally, attackers can launch more than one attack at a time to change the attack signature, making such an attack invisible to techniques that focus on specific signatures.

To effectively address this problem, specific machine learning algorithms have also been developed that either use properties exhibited by the training data of malicious URLs or recognize some associated features.

In [9], instead of using traditional blacklist-based filters or landing page content analysis for web URLs, the authors study the behavioral factors of both the URL publisher and the URL clicker. Link posting behavior was measured via the Twitter API, while clicker behavior was measured via the Bitly API. Examples of used features: number of publications, intensity of publications, number of links, difference between peak values of links, etc. The basic idea is that these behavioral signals may be more difficult to manipulate than traditional signals. Authors offer and rate features based on clicks and posts. The final result of this approach: precision (0.86), recall (0.86), and AUC (0.92), using only behavioral factors. The authors suggest not to consider this approach separately, but instead emphasize that the integration of similar features into existing algorithms and methods should significantly improve the reliability of the final solution.

In [10], the authors used domain information about links to recognize phishing pages. One of the advantages of their method was the use of domain information to detect phishing sites, as this important characteristic was not taken into account in other anti-phishing methods. In this method, first of all, all direct and indirect links to the selected site were extracted. Next, using the source code of the page, all domains associated with direct links were extracted and placed in the S_1 set. Then the indirect links were extracted and placed in the S_2 set, after that both sets were combined and only common domains were extracted, from which the IP addresses of these domains were subsequently extracted using a DNS lookup. The result of the study shows that the accuracy, FP, FN, TN and TR of the proposed method are 99.62, 0.32, 0.5, 99.5 and 99.67 percent. Despite its advantages such as accuracy, the dependence of this method on various external methods such as DNS lookups and search engines can affect its effectiveness.

In [11], the authors use the design of the Transformer classifier model to detect malicious URLs. The authors compare their model with a number of other models (Decision Tree, Random Forest, Multilayer perceptron, XGBoost, SVM, and Auto Encoder) and show that their model is better in precision, recall, and accuracy than all other models on their dataset. The final result of this approach: precision (0.98), recall (0.96), and accuracy (0.97). The peculiarity of the approach is that only the URL address is used without pre-processing. This work can be considered a continuation of work [12], as it develops the idea of using sequential models for the classification of malicious URLs. However, among the problems of the work, one can single out too small a set of data for training, testing and validation, especially considering that the proposed model is much more computationally complex than the others considered.

An interesting approach to the problem is also demonstrated in [13]. This paper presents a reinforcement learning-based model for automatic URL-based phishing detection. This implementation of the deep learning algorithm is an additional approach to existing phishing detection methodologies, the advantage of the system is its greater dynamism and the possibility of continuous learning. At the same time, the authors themselves note that the work is rather a basis for the direction of reinforcement learning in this field, since the authors use the basic algorithm of Deep Q Learning and a limited set of features, namely only URL lexical features. The final result of this approach: precision (0.86), recall (0.88), and accuracy (0.90). The result is worse than other considered approaches, but the work raises an important topic of dynamic retraining of models and their more flexible adaptation to changes in the behavior of attackers.

An alternative view of the problem is raised in work [14], where the authors investigate the vulnerabilities of existing solutions based on machine learning and implement an evasion attack on existing systems. The authors first themselves create several different solutions based on machine learning, and then implement the evasion attack developed in the work. The work shows that methods based exclusively on lexical features of URLs have a rather low reliability against this type of attack, moreover, even external solutions such as VirusTotal and PhishTank can be affected by such an attack.

Thus, the existing solutions contain both positive features and leave a possibilities for improvement.

1.3. Malicious URL features

To obtain an informative representation of the URL, the researcher needs to create a set of features based on domain knowledge and heuristics. These features can be lexical features (URL string statistical properties, n-grams, etc.), host-based features (WHOIS information, host geolocation properties), etc.

Problems that arise. A very large number of URLs can be a problem if the researcher has a large sample at his disposal and new data is constantly coming in. As a result, the training time for traditional models may be too high to be practical. To overcome this problem, it is possible, for example, to apply the online learning approach.

Another problem may be caused by the specificity of certain representations of features. For example, high sparsity of data in the case of representing it using a bag of words (BoW) approach. This data representation can produce a huge number of features, most of which will always be zeros, so researchers must either optimize feature extraction approaches or use algorithms that can adequately handle the sparse representation..

Stages of features formation. The first stage is transformation of URL-address u to the vector of the features x, where it is possible to consider several types of information and to use defferent methods accordingly. The features can vary from lexical information (URL word length, URL general length) to the host-based information (WHOIS, IP-addres, location etc). The second stage is information processing and formation of feature vector x. For example, numerical information can be used as it is, and Bag-of-Words, which is used to represent textual or lexical content, must be converted to numerical data, for example, using word2vec.

After obtaining the feature vector x for the training data to find the prediction function $f: \mathbb{R}^d \rightarrow \mathbb{R}$, it is usually formulated as an optimization problem such that detection accuracy is maximized. Function f is parametrized by d-dimentional vector with weight w usually, so that $f(x) = (w_{\mathrm{T}} x)$.

Let $\widehat{y_t} = sign\big(f(x_t)\big)$ denotes class label forecast, made by f function. The number of mistakes, made by forecast model for all initial data, is determined as: $\sum_{t=1}^{T} \mathbb{I}_{\widehat{y_t}=y_t}$, where $\mathbb{I}$ – an indicator that evaluates to 1 if the condition is true and 0 otherwise. Since the indicator function is not convex, the optimization can be difficult to solve.. So often convex loss function is determined, that is denoted as $\ell(f(x), y)$. General optimization problem can be formulated as: $\min_w \sum_{t=1}^{T} \ell(f(x_t), y_t)$.

It is possible to use several types of loss functions including the losses in form:

$$\ell(f(x), y) = \frac{1}{2} \max(1 - yf(x), 0)\,,$$

or function of quadratic losses:

$$\ell(f(x), y) = \tfrac{1}{2}\big(y - f(x)\big)^2.$$

Sometimes a regularization term is also added to prevent overfitting or to train models with a minimum number of parameters, in some cases the loss function is modified based on the specific data available, for example in the case of class imbalance, or if the study predicts different loss estimates for different types of threats.

2. Features engineering and representation

2.1. Features types

To detect malicious URLs, researchers have proposed a large number of types of features that can be used to provide useful information about the URL. Conventionally, all attributes can be classified into: blacklist-based attributes, lexical URL attributes, host-based attributes, content-based attributes, and others (context and popularity).

Traditional lexical features are: the most commonly used lexical features include statistical properties of the URL string, such as URL length, length of each component of the URL (hostname, top-level domain, root domain), number of special characters, etc. In [15], the authors were one of the first to suggest choosing words from the URL string. The string was processed so that each segment separated by a special character (eg "/", ".", "?", "=", etc.) made up a word. Using all words from all available URLs, a dictionary was created, meaning each word became a feature. If the word was present in the URL, the value of the function would be 1, otherwise it would be 0. This text processing model is known as BoW.

Using the BoW model directly results in the loss of word order information in the URL. In [16], the authors distinguished the membership of tokens to the hostname, path, top-level domain, and root domain by creating separate dictionaries for each component. This separation made it possible to partially preserve information about the order of words. For example, it made it possible to distinguish the presence of "com" in the top-level domain from other parts of the URL. In [17], the authors enriched the lexical features by considering the use of bigrams, that is, they build a dictionary where, in addition to ordinary words, a combination of two words in the URL is also a feature.

In general, the BoW approach can be seen as a fuzzy, machine learning-compatible blacklist implementation. Instead of focusing on the entire URL string, it assigns URL points based on the smaller components of the URL string. Although this approach offers us a large number of features, it can become problematic when running complex algorithms on them. For example, in [16] dataset of 2 million URLs was collected that have almost the same number of lexical features. This number can grow even more if we take into account the characteristics of bigrams. Paper [13] considered the features of n-grams (same as bigram, but n can be greater than 2), and developed a feature selection scheme based on relative entropy for dimensionality reduction. A similar method of distinguishing lexical features was used in work [18], where the weight of a feature was calculated according to how often it occurs in one class than in another.

To avoid blacklisting, hackers can generate malicious URLs algorithmically. Using BoW for such URLs is likely to result in poor performance, as algorithmically generated URLs may produce words that the model has not seen before (and thus new features). To detect such algorithmically generated malicious URLs, you can try parsing the strings at the character level to get the signatures. According to research [20], algorithmically generated domain names and names generated by people would have a significantly different alphanumeric distribution.

Advanced lexical features: While traditional approaches to obtaining lexical features are not domain-specific and are obtained directly from the URL string, there are also more sophisticated lexical features for more informativeness. One of the goals of such approaches can be, for example, the derivation of features resistant to obfuscation, as in [21]. Based on the types of obfuscation defined in [7], five categories of features are proposed:

- attributes related to the URL (keywords, length, etc.);
- domain characteristics (length of the domain name, whether an IP address is used as a domain name, etc.);
- attributes related to the directory (length of the directory, number of subdirectory markers, etc.);
- attributes of the file name (length of the file name, number of separators, etc.);
- argument attributes (argument length, number of variables, etc.).

The success of a machine learning model critically depends on the quality of the training data and the quality of the feature representation. Let URL$\in \mathbb{U}$, where $\mathbb{U}$ denotes the set of x valid input URLs, the aim of represention the features is to find a reflection $g : \mathbb{U} \rightarrow \mathbb{R}^d$, such that $g(u) \rightarrow x$, where $x \in \mathbb{R}^d \epsilon$ d-dimentional feature vector, which is given to machine learning model input.

2.2. Solutions using host-based features

Host-based tags are obtained from the hostname properties of the URL. They allow us to know the location, identity, ownership type and properties of malicious hosts. Article [22] investigated the effect of several host-based features on the detection of malicious URLs. One of the key observations was that attackers were using URL shortening services and many were using botnets to host their sites on multiple machines in multiple countries. Hence, host-based signatures have become an important element in detecting malicious URLs. In article [23] is proposed the use of several host-based features including: IP address properties, WHOIS information, location, domain name properties, and connection speed. IP address properties contain attributes derived from the IP address prefix and the autonomous system number. WHOIS information includes domain name registration dates, registrars and registrants. Location information contains physical geographic location data - e.g. the country/city to which the IP address belongs. Domain name properties include the TTL value, the presence of certain keywords such as "client" and "server", whether or not the IP address is in the hostname, and whether the PTR record is one of the host's IP addresses. Since many features are identity-related information, a BoW-like approach is needed to store them in a numeric vector where each word corresponds to a specific concept. As in the case of lexical features, this approach leads to a large number of features. IP address signatures are probably more stable, as it will be difficult for attackers to constantly obtain new IP addresses for their URLs.

In [24], the authors determine the age of the domain and the "confidence" of the domain (depending on the similarity with whitelisted domains), which helps to determine the variability of the URL (for example, malicious URLs using fast flux will have a lower domain age). It is also possible to use the headers of the HTTP responses, for example it is possible to use the age obtained from the timestamp of the last modified header. It is also possible to use network layer features in combination with application layer to build a multi-layer malicious URL detection mechanism..

2.3. Solutions using content-based features

Content-based features are obtained after the entire web page is loaded. Compared to features based solely on URLs, are "difficult" because a lot of information needs to be retrieved and at the same time there may be security issues. However, with more information about the content of a web page, it is natural to assume that this will lead to a better prediction model. Further, if URL-based features cannot detect a malicious URL, a more thorough analysis of content-based features can help in early detection of threats [25]. Content-based web page features can be derived primarily from HTML content and the use of JavaScript. Content-based web page attributes can be tentatively classified into 5 broad categories: lexical attributes, HTML document attributes, JavaScript attributes, ActiveX objects, and attribute relationships. The authors of [26] propose the approach for detecting phishing websites using a comprehensive approach based on machine learning, using various features of the HTML Document Object Model (DOM), search engines, and third-party services. Next, we'll discuss some of these categories, focusing primarily on HTML document-level features and JavaScript features.

HTML tags. The first type of features that can be extracted from an HTML document are lexical features, they are relatively easy to extract and preprocess. The next level of complexity of HTML features are document-level features. Such signs are various statistical features of the HTML document, as well as the use of various functionality. In paper [27] is suggested the use of such features as: document length, average word length, number of words, number of unique words, number of words in a line, number of NULL characters, use of string concatenation, asymmetric HTML tags, links to remote source scripts, and invisible objects. Malicious code is often encoded in HTML, which involves long word lengths or heavy use of concatenation, and so these features can help detect malicious activity. Similar features with minor variations have been used by many subsequent researchers, including [28] (number of iframes, number of zero-sized iframes, number of lines, number of hyperlinks, etc.). In paper [29] is also used the similar features and additionally proposed to use several more descriptive features that were aimed at secondary statistical properties of the page. These include the following features such as the number of elements with small area, the number of elements with suspicious content (suspiciousness is determined by the content length between the start and end tag), the presence of duplicate documents, etc. In paper [29] is developed a delta method where the delta represented the changes in different versions of a website. They analyzed whether the change was malicious or safe.

JavaScript features. Typically, attackers use JavaScript functions to encrypt or simply hide malicious code. For example, extensive use of the eval() and unescape() functions may indicate the execution of hidden code in HTML. The authors of [27] use native JavaScript functions to build a BoW model as features for detecting malicious URLs. Among all the native functions of JavaScript, the researchers single out those that are most often used when performing XSS attacks or when spreading malicious software, they include: escape(), eval(), link(), unescape(), exec() and search( ). The authors of [29] propose additional heuristic features based on JavaScript code. In [30], the authors try to detect JavaScript obfuscation by analyzing JavaScript codes using n-grams, entropy, and word size. n-grams and word size are commonly used to study the statistical distribution of words and symbols. Regarding the use of entropy, the authors note that obfuscated strings usually have a lower entropy compared to normal code. It is also possible to apply deep learning methods to create a feature representation from JavaScript code [31].

Visual features. There have also been attempts to use images of web pages to determine the malicious nature of a URL. Most of them focus on calculating the visual similarity of the analyzed sites to conditionally safe sites. A very high level of visual similarity may indicate that the site is masquerading as another known site and is a phishing attempt. With recent advances in deep learning and image recognition, more effective visual features can be obtained.

2.4. Other features

Context features. In recent years, there has been an increase in the number of short URL service providers that allow the original URL to be represented as a shorter string. This allows URLs to be shared on social media platforms like Twitter, where originally long URLs would not fit within the 140 character limit of a tweet. Unfortunately, this has also become a popular obfuscation technique for malicious URLs. Although URL shortening providers try their best to avoid generating URL shortenings for attackers, it is difficult for them to do an effective job. As a result, a research direction has recently appeared that takes into account the contextual features of a URL, that is, the features of the environment in which a given URL was distributed. In [32] the contextual information is used, which was obtained from the tweets in which the URL was sent. In [33] a click and traffic data were used to classify short URLs as malicious or not. Paper [9] proposes a different direction of malicious identification tools - they also focus on URLs that are shared in social networks and direct their attention to detecting the malicious nature of a URL by analyzing the behavior of users who shared it and users who clicked on the address These features are officially called "Posting-based" and "Click-based" features.

Signs of popularity. Some other features have been developed as heuristic approaches to measure the popularity of a URL. One of the earliest approaches using statistical methods to detect malicious URLs [7] was aimed at probabilistic identification of the importance of manually selected features. These include page-based attributes (page rank, quality, etc.), domain-based attributes (presence in domain whitelisting), type-based attributes (types of obfuscation), and word-based attributes (presence of keywords such as "confirm" , "banking", etc.). Also, an important category of research on this topic is the study of link graphs between different addresses.

All considered before features have their advantages and disadvantages, and while some of them are very informative, the methods of obtaining them can be difficult or unprofitable to implement. Similarly, different features have different preprocessing challenges.

3. Stacking method and models

3.1. The main ideas

This work proposes to combine the strengths of existing algorithms and approaches to feature extraction from URLs by combining models using the stacking method.

Stacking involves training an algorithm to combine the predictions of several other machine learning algorithms. First, all other algorithms are trained using the available data, then the combinator algorithm is trained to make a final prediction using all the predictions of the other algorithms as additional input. In practice, the logistic regression model is often used as a combinator. An illustration of what staking looks like in practice can be seen in Figure 1.

Stacking usually provides better performance than any one of the trained models [34]. It is successfully used both for learning with a teacher and for learning without a teacher.

In contrast to the more traditional methods of ensemble boosting and bagging (boosting, bagging), it is better to use algorithms of different "nature" for stacking, i.e., those that have different assumptions about the data model.

As can be seen from the existing works, there are quite a lot of signs that can be obtained from a single URL using third-party services such as whois. However, due to the computational and engineering complexity of obtaining some features, their number should not be too large.

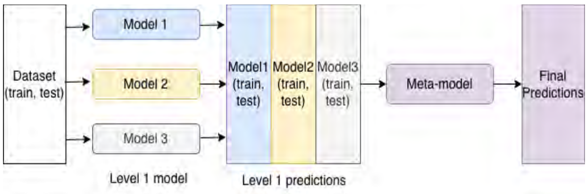

Figure 1: Visualization of models stacking

Also, from examples of existing approaches, we can see that with the help of neural networks, we can transmit the URL itself as an independent set of features.

3.2. Features and dataset

Tables 1,2 provide a description of the characteristics used in the development of the algorithm.

Table 1
Address-based features

Features	Description
Using the IP Address	The IP address is used instead of the domain name (sometimes converted to 16-bit form)
Dot Count	Number of subdomains
Digit Count	Number of digits in URL
Special Character Count	Using the symbols from set (';', '+=', '_', '?', '=', '&', '[', ']')
Hyphen Count	Hyphen usage (rarely used in benign URLs)
Double Slash Count	The marker of redirection
Single Slash Count	The number of single slashes in the address
URL Length	URL length (a long URL may be needed to hide the suspicious part)

Table 2.
Network features

Features	Description
Resolved IP count	The number of IP addresses that can be resolved for the URL domain
Name server count	The number of name servers serving the domain
Name server IP count	The number of IP addresses these name servers are associated with.
Registered Date	Registration date (whois)
Expiration Date	Validity (whois)
Update Date	Date last updated (whois)

A dataset containing 96,018 instances was used for the work, of which 48,009 were reliable and 48,009 were phishing. The data are obtained from the Aalto University Datasets service. The dataset itself is presented with additional characteristics, but the work only uses URLs in their pure form and the class label, other characteristics have been removed.

All further processing, analysis, training and evaluation of models took place in the Python.

For each of the sets of characteristics, a number of experiments were conducted with different models. Each model was evaluated according to several criteria, after which the best model was selected at each step by averaging.

3.3. Models assessment

Models are evaluated on three dimensions: Precision (1), Recall (2) and AUC. The evaluation procedure is cross-validation with the parameter k=5.

$$\text{Precision} = TP/(TP + FP) \qquad (1)$$

$$\text{Recall} = TP/(TP + FN) \qquad (2)$$

The basis of the method is the division of the existing training set of data into k approximately equal blocks, for example k=5. Then on k−1, that is, on 4 blocks, the model is trained, and the 5th block is used for testing. The procedure is repeated k times, while on each pass a new block is selected for testing, and training is performed on the others. Cross-validation has an important advantage over using one set for training and one for testing the model: if you estimate the original error of the model at each pass and agree it across all passes, the resulting estimate will be more reliable.

For all types of features the best results have XGBoost і Random Forest, які дорівнюють відповідно 0,88 і 0,87. The worst result has SVM - 0,80 (Figure 2).

For lexical features of the address string the best estimations for all three parameters have kNN and Random Forest with AUC (0,813, 0,810), the worst is SVM with 0.705 (Figure 3).

For address string network features the results show that the XGBoost and Random Forest algorithms have better scores for all three parameters, the AUC of these algorithms are 0.872 and 0.860, respectively. And the worst result shows the SVM algorithm with a result of 0.789 (Figure 4).

Thus, we choose two first-level algorithms for stacking - kNN and XGBoost for use on address characteristics and network characteristics, respectively. The final goal of this work is to build a combined classifier using the stacking method to improve the metrics of existing simple classifiers. Based on the results of testing on different sets of characteristics, the best intermediate classifiers - kNN and XGBoost - were determined. As the third classifier in the ensemble, we will use a completely different classifier based on the neural network architecture of the transformer described in [11].

Let's set a conditional initial precision from which we will start: the Tranfsormer model on our data set showed the following results: precision (0.91), recall (0.89), and AUC (0.9). Accordingly, this is the result that we will try to improve by stacking models.

Also we test the assumption that a larger number of specialized models can be better than a smaller number of large models, even in spite of worse intermediate results.

The next step is to choose a model that will be a stacking model. Three models participate in this stack: the transformer model trained only on the text of URL addresses, the kNN model for lexical features, and the XGBoost model for network features. The results of stacking three models by different top-level models are shown in Figure 5.

During the experiment, it was established that thanks to the new classification scheme, there is an improvement in such parameters as AUC, precision, and recall. A fixed metric boost is available for most second-level algorithms. The Logistic Regression algorithm showed the best result with an AUC increase of 3%.

Let's also compare these results with the stacking of two models: Transformer model and XGBoost classifier trained on all available features to test whether it is better to use more small models for stacking.

The results of stacking two models with different top-level models are shown in Figure 6.

During the experiment, we saw that, on average, stacking 3 smaller models turned out to be more effective on our data than stacking two larger ones, although it should be noted that this result is not guaranteed to be preserved for other data sets and other subject areas (Figure 7).

As a result of the experiment, it was found that the developed stacking model shows better results than the Transformer model on all types of malicious links, but slightly worse results on safe URLs, that is, the developed model can potentially give more false positive answers. It is also interesting that the models work best on phishing links, which can be explained by the fact that the training data set mostly consisted of such links (Figure 8).

The AUC comparison of the models is shown in Figure 9, all model names are abbreviated to first letters, column names are in the format "[lexical feature model]_[network feature model]" .

As you can see from the results, the "choose the best model at each step" approach is a good starting point for stacking, but not the best. 9 out of 42 different combinations of models showed a better result. The combination of XGBoost + Logistic Regression showed the greatest increase, namely +1.7% to AUC.

Conclusions

Analysis of the performance of different classifiers on two groups of features - lexical and network features, allowed choosing the best model of the first level for each set of features: for lexical features - kNN with AUC=0.813; for network - XGBoost with AUC=0.872. A general model was built on the entire set of features: the best was XGBoost with AUC= 0.884. A new classification model was built, which is a stacking of 3 models: two pre-trained kNN and XGBoost models, and the Transormer model, which showed the best result on the available data.

The best model for stacking was determined, it turned out to be the Logistic Regression model with a result of 0.927.

Validation of the proposed model was performed on an independent data set with a different distribution and the performance of the model on different types of malicious links was investigated. The advantage of the developed model is preserved even on independent data, but it should be noted that the performance of all models drops slightly. It was also determined that the model detects phishing sites the best, and spam sites - the worst.

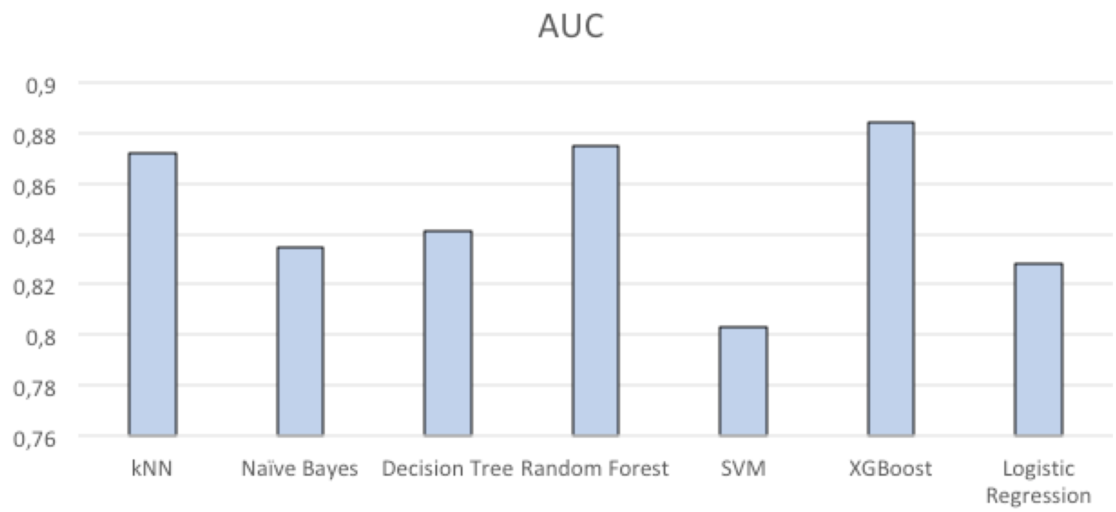

Figure 2: AUC values for models trained on all features

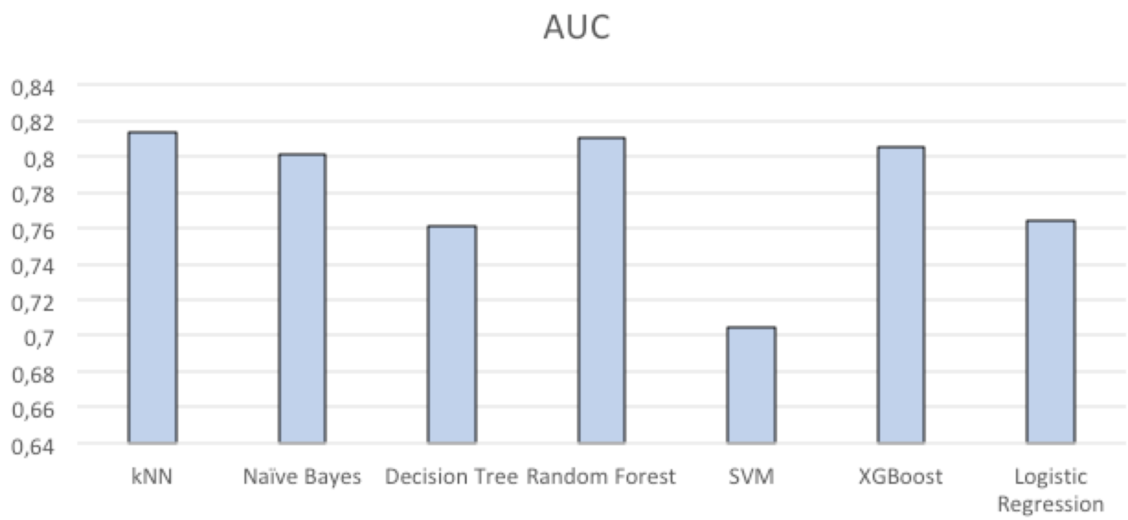

Figure 3: AUC values for address string lexical features

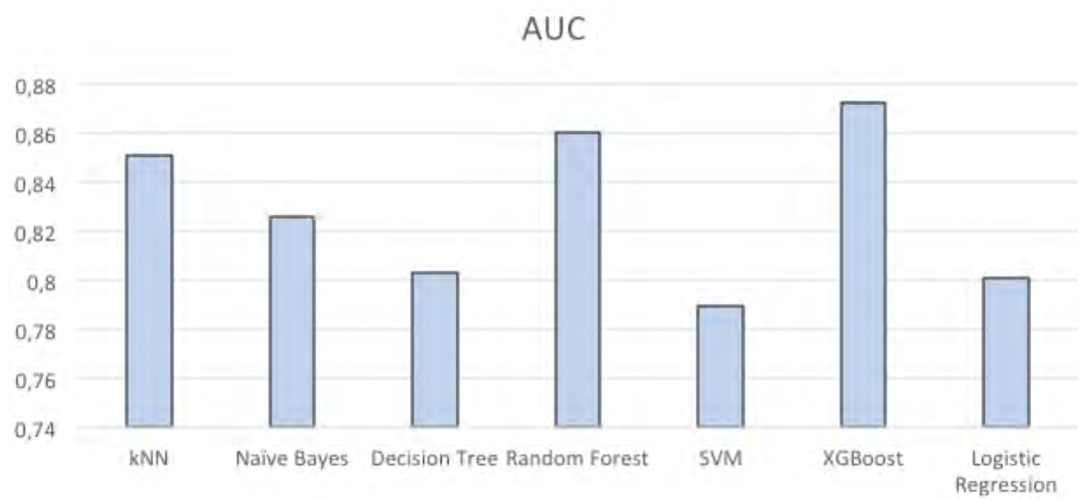

Figure 4: AUC values for address string network features

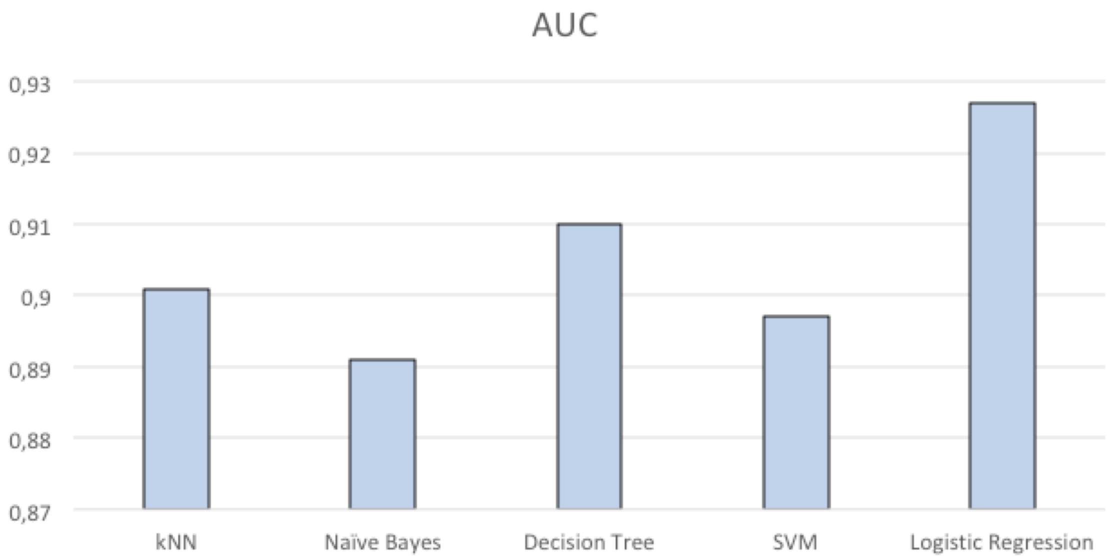

Figure 5: AUC values for different 3-model stacking algorithms

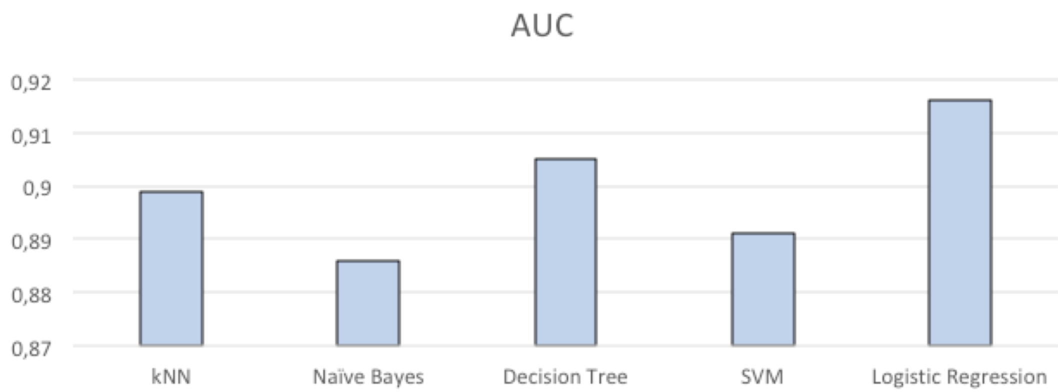

Figure 6: AUC of the two models

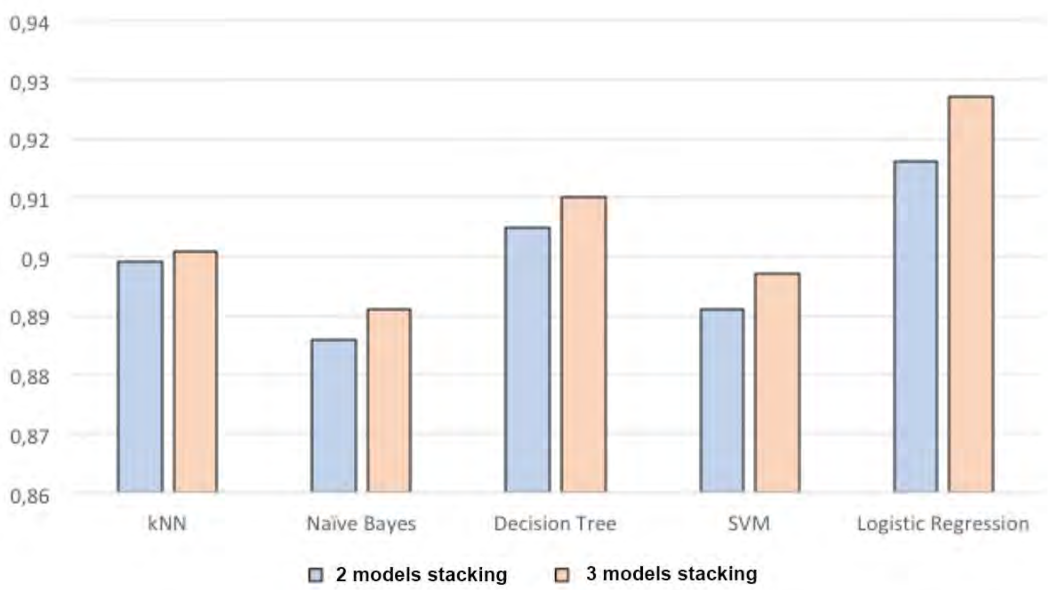

Figure 7: Comparison of stacking performance of 2nd and 3rd models

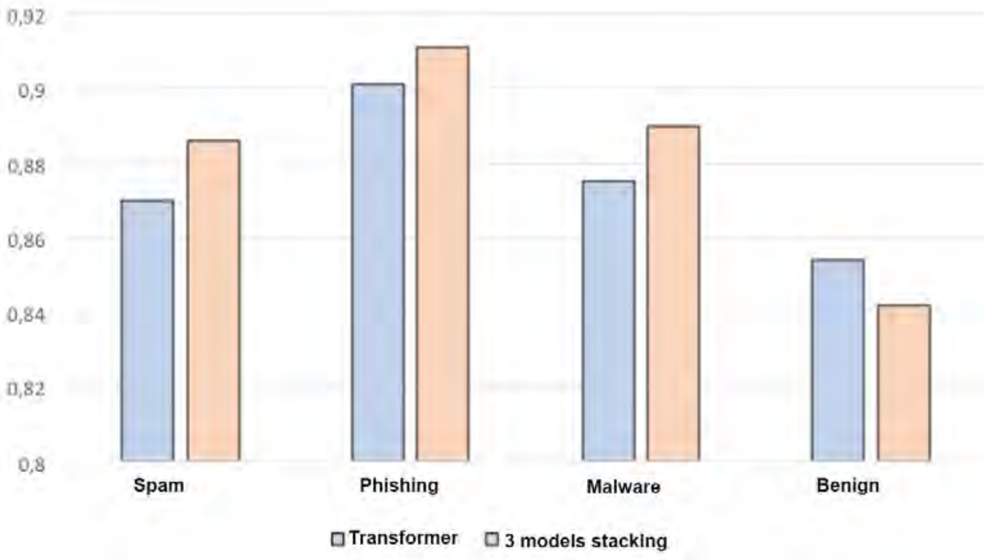

Figure 8: Comparison of models on different link types

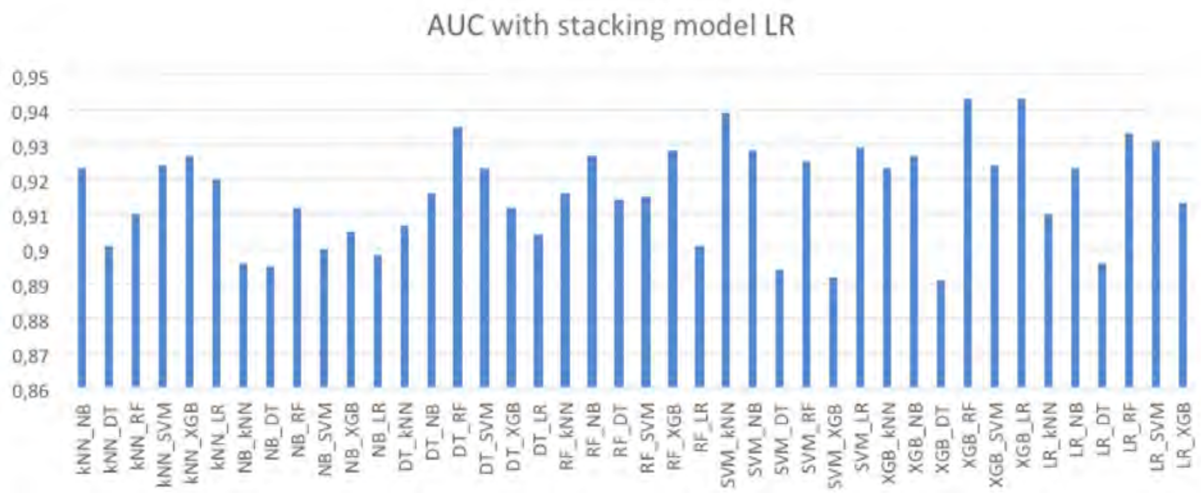

Figure 9: Comparison of AUC for all possible combinations of two first-level models

References

[1] Hong J. The State of Phishing Attacks [Electronic resource] / Jason Hong. – 2012. – URL: https://www.researchgate.net/publication/220424515_The_State_of_Phishing_Attacks.

[2] Malicious Web Pages Detection Based on Abnormal Visibility Recognition [Electronic resource] / [B. Liang, J. Huang, F. Liu та ін.]. – 2009. – URL: https://www.researchgate.net/publication/224545611_Malicious_Web_Pages_Detection_Based_on_Abnormal_Visibility_Recognition.

[3] Patil D. Survey on Malicious Web Pages Detection Techniques [Electronic resource] / D. Patil, J. Patil. – 2015. – URL: https://www.researchgate.net/publication/287359077_Survey_on_Malicious_Web_Pages_Detection_Techniques.

[4] Malicious URL Categories [Electronic resource] – URL: https://docs.paloaltonetworks.com/pan-os/9-1/pan-os-admin/url-filtering/url-categories/url-category-best-practices.

[5] PhishTank [Electronic resource] – URL: https://phishtank.org/.

[6] Sinha S. Shades of grey: On the effectiveness of reputation-based “blacklists” [Electronic resource] / S. Sinha, M. Bailey, F. Jahanian. – 2008. – URL: https://www.researchgate.net/publication/22

4354096_Shades_of_grey_On_the_effectiveness_of_reputation-based_blacklists.

[7] A framework for detection and measurement of phishing attacks [Electronic resource] / [S. Garera, N. Provos, M. Chew та ін.]. – 2007. – URL: https://www.researchgate.net/publication/228619297_A_framework_for_detection_and_measurement_of_phishing_attacks.

[8] Domain Generation Algorithms detection through deep neural network and ensemble [Electronic resource] / [S. Li, T. Huang, Z. Qin та ін.]. – 2019. – URL: https://www.researchgate.net/publication/333060399_Domain_Generation_Algorithms_detection_through_deep_neural_network_and_ensemble.

[9] Chao C. Detecting Spam URLs in Social Media via Behavioral Analysis [Electronic resource] / C. Chao, J. Caverlee. – 2015. – URL: https://www.researchgate.net/publication/302021309_Detecting_Spam_URLs_in_Social_Media_via_Behavioral_Analysis.

[10] Gowtham R. An efficacious method for detecting phishing webpage through Target Domain Identification [Electronic resource] / R. Gowtham, I. Krishnamurthi, K. Kumar. – 2014. – URL: https://www.researchgate.net/publication/260006990_An_efficacious_method_for_detecting_phishing_webpage_through_Target_Domain_Identification.

[11] Pingfan X. A Transformer-based Model to Detect Phishing URLs [Electronic resource] / X. Pingfan. – 2021. – URL: https://arxiv.org/abs/2109.02138.

[12] URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection [Electronic resource] / [H. Le, Q. Pham, D. Sahoo та ін.]. – 2018. – URL: https://www.researchgate.net/publication/323118482_URLNet_Learning_a_URL_Representation_with_Deep_Learning_for_Malicious_URL_Detection.

[13] Chatterjee M. Deep Reinforcement Learning for Detecting Malicious Websites [Electronic resource] / M. Chatterjee, A. Siami Namin. – 2019. – URL: https://www.researchgate.net/publication/333309352_Deep_Reinforcement_Learning_for_Detecting_Malicious_Websites.

[14] Sabir B. An Evasion Attack against ML-based Phishing URL Detectors [Electronic resource] / B. Sabir, M. Ali Babar, R. Gaire. – 2020. – URL: https://www.researchgate.net/publication/341477958_An_Evasion_Attack_against_ML-based_Phishing_URL_Detectors.

[15] Kolari P. SVMs for the Blogosphere: Blog Identification and Splog Detection [Electronic resource] / P. Kolari, T. Finin, A. Joshi. – 2006. – URL: https://www.researchgate.net/publication/221250881_SVMs_for_the_Blogosphere_Blog_Identification_and_Splog_Detection.

[16] Identifying suspicious URLs: An application of large-scale online learning [Electronic resource] / [J. Ma, L. Saul, S. Savage та ін.]. – 2009. – URL: https://www.researchgate.net/publication/221345258_Identifying_suspicious_URLs_An_application_of_large-scale_online_learning.

[17] Lexical feature based phishing URL detection using online learning [Electronic resource] / [A. Blum, B. Wardman, T. Solorio та ін.] // Proceedings of the ACM Conference on Computer and Communications Security. – 2010. – URL: https://www.researchgate.net/publication/221609867_Lexical_feature_based_phishing_URL_detection_using_online_learning.

[18] Malicious web page detection based on on-line learning algorithm [Electronic resource] / [W. Zhang, Y. Ding, Y. Tang та ін.] // Proceedings - International Conference on Machine Learning and Cybernetics. – 2011. – URL: https://www.researchgate.net/publication/221544821_Malicious_web_page_detection_based_on_on-line_learning_algorithm

[19] Detecting Algorithmically Generated Malicious Domain Names [Electronic resource] / [S. Yadav, A. Reddy, S. Ranjan та ін.] // Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC. – 2010. – URL: https://www.researchgate.net/publication/220269670_Detecting_Algorithmically_Generated_Malicious_Domain_Names.

[20] Anh L. Phishdef: URL names say it all [Electronic resource] / L. Anh, M. Athina, F. Michalis // IEEE INFOCOM. – 2010. – URL: https://www.researchgate.net/publication/46584584_Phishdef_URL_names_say_it_all.

[21] McGrath D. Behind Phishing: An Examination of Phisher Modi Operandi [Electronic resource] / D. McGrath, M. Gupta. – 2008. – URL: https://www.researchgate.net/publication/220831975_Behind_Phishing_An_Examination_of_Phisher_Modi_Operandi.

[22] Ma J. Beyond blacklists: learning to detect malicious Web sites from suspicious URLs [Electronic resource] / J. Ma, L. Saul, S. Savage. – 2009. – URL: https://www.researchgate.net/publication/221653642_Beyond_blacklists_learning_to_detect_malicious_Web_sites_from_suspicious_URLs.

[23] Protect Sensitive Sites from Phishing Attacks Using Features Extractable from Inaccessible Phishing URLs [Electronic resource] / [W. Chu, B. Zhu, F. Xue та ін.] // IEEE International Conference on Communications. – 2013. – URL: https://www.researchgate.net/publication/264293197_Protect_Sensitive_Sites_from_Phishing_Attacks_Using_Features_Extractable_from_Inaccessible_Phishing_URLs.

[24] Prophiler: A fast filter for the large-scale detection of malicious web pages [Electronic resource] / [D. Canali, M. Cova, G. Vigna та ін.]. – 2011. – URL: https://www.researchgate.net/publication/221023059_Prophiler_A_fast_filter_for_the_large-scale_detection_of_malicious_web_pages.

[25] CANTINA: A content-based approach to detecting phishing web sites [Electronic resource] / [Y. Zhang, J. Hong, L. Cranor та ін.]. – 2007. – URL: https://www.researchgate.net/publication/221023659_CANTINA_A_content-based_approach_to_detecting_phishing_web_sites.

[26] Malicious web content detection by machine learning [Electronic resource] / [H. Yung-Tsung, C. Yimeng, C. Tsuhan та ін.] // Expert Syst. Appl.. – 2010. – URL: https://www.researchgate.net/publication/220219014_Malicious_web_content_detection_by_machine_learning.

[27] Detecting Malicious Web Links and Identifying Their Attack Types [Electronic resource] / [C. Hyunsang, Z. Bin, L. Heejo та ін.] // Proceedings of the 2nd USENIX Conference on Web Application Development. – 2011. – URL: https://dl.acm.org/doi/10.5555/2002168.2002179.

[28] Automating URL Blacklist Generation with Similarity Search Approach [Electronic resource] / [S. Bo, A. Mitsuaki, Y. Takeshi та ін.] // IEICE Transactions on Information and Systems. – 2016. – URL: https://www.researchgate.net/publication/299542431_Automating_URL_Blacklist_Generation_with_Similarity_Search_Approach.

[29] Choi Y. Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis [Electronic resource] / Y. Choi, T. Kim, S. Choi // International Journal of Security and its Applications. – 2010. – URL: https://www.researchgate.net/publication/289641814_Automatic_detection_for_javascript_obfuscation_attacks_in_web_pages_through_string_pattern_analysis.

[30] Yao W. A deep learning approach for detecting malicious JavaScript code [Electronic resource] / W. Yao, C. Wan-dong, W. Peng-cheng // Security and Communication Networks. – 2016. – URL: https://www.researchgate.net/publication/294283262_A_deep_learning_approach_for_detecting_malicious_JavaScript_code.

[31] Lee S. Warningbird: Detecting Suspicious URLs in Twitter Stream [Electronic resource] / S. Lee, J. Kim. – 2012. – URL: https://www.researchgate.net/publication/228517184_WARNINGBIRD_Detecting_Suspicious_URLs_in_Twitter_Stream.

[32] Click Traffic Analysis of Short URL Spam on Twitter [Electronic resource] / [D. Wang, S. Navathe, L. Liu та ін.]. – 2013. – URL: https://www.researchgate.net/publication/261201832_Click_Traffic_Analysis_of_Short_URL_Spam_on_Twitter.

[33] Wolpert D. Stacked generalization [Electronic resource] / David Wolpert // Neural Networks. – 1992. – URL: https://www.sciencedirect.com/science/article/abs/pii/S0893608005800231.