ISSN 1870-4069

Adversarial Attacks in Word Processing: Impact on SPAM Detection Models

Samantha Acosta Ruiz

Benemérita Universidad Autónoma de Puebla,
Facultad de Ciencias de la Computación,
Mexico

224570157@viep.com.mx

Abstract. Word processing in machine learning faces several challenges, such as language variability, semantic ambiguity, and the need to correctly interpret context. These challenges are intensified in spam detection, where attackers use sophisticated techniques to evade detection. This study examines the impact of poisoning and evasion attacks on machine learning models, highlighting that the former have a considerably greater effect. These attacks modify the training data, deteriorate the performance of the model and cause systematic errors in the predictions. The spam data class is especially affected, which compromises the model's ability to detect unwanted emails. For the generation of adversarial examples, a DistilBERT classifier was used. The models evaluated included AdaBoost, Support Vector Machine and RandomForest. To measure the impact of the attacks, metrics such as accuracy, recall, precision and F1-score were analyzed. The results showed that the most vulnerable model to these attacks was AdaBoost, which initially had an accuracy of 97.8% in classifying spam data. However, the poisoning attack turned out to be the most harmful, reducing this metric to 69%. This analysis highlights the importance of implementing robust defenses against poisoning attacks and developing advanced word-processing techniques to maintain the integrity and effectiveness of models in adverse environments.

Keywords: Adversarial machine learning, spam detection, generative adversarial example, distilBERT, text classification.

1 Introduction

Within the field of cybersecurity, it has been consistently recognized that human beings represent the most vulnerable link and the first line of vulnerability in any system. Although conventional threats can be assessed and quantified through penetration testing, social engineering (SE) poses more subtle and complex challenges. Social engineers use various tactics, such as phishing and adware, to manipulate users and obtain information voluntarily. In the context of social networks, social engineering takes on an appearance that resembles regular publications, albeit with a latent

malicious background. Intruders have the ability to impersonate legitimate entities, such as banking institutions in order to gain access to accounts or passwords by sending spam messages, these attacks usually start with meticulous and planned reconnaissance phases. There is little understanding in identifying SE attacks due to the subjectivity of social media posts [3]. In addition, it is challenging for a machine to recognize sentiments and read between the lines of social posts that may be a SE threat. Common network security appliances, such as firewalls, only detect illegitimate traffic at flow levels.

Meanwhile, the Intrusion Detection System (IDS)/Intrusion Prevention System (IPS) must be highly customized with intelligent rules to read application data [10]. In recent years, there has been much work focused on training deep learning models for threat intelligence using network properties, but SE attacks are mostly textual data that require integration with Natural Language Processing (NLP) [12].

In the fields of security and how to protect Artificial Intelligence (AI) models against similar threats, we have to talk about a particular area of Machine Learning (ML), in this case it is the Adversarial Machine Learning. When deploying on Machine Learning (ML)/Deep Learning (DL) model, it should be expected that the models do not present any variability, but this is not true in most cases. This is the central theme of Adversarial Machine Learning, which is a branch of ML that tries to find out what attacks a model can suffer in the presence of a malicious adversary and how to protect against them. There are three types of attacks [4]:

- **White box attacks:** The adversary has access to the architecture used by the model, training data, parameters and hyperparameters.
- **Black box attacks:** The adversary only has access to the inputs and outputs of the model.
- **Gray box attacks:** The attack is located at an intermediate point between the data types of previous attacks.

Although it is possible to carry out numerous attacks on machine learning models.

In Adversarial Machine Learning these attacks fall into four categories: poisoning, evasion, extraction and inversion [7]. These attacks exploit vulnerabilities inherent in ML models, leading to serious consequences in critical applications, such as security, health and finance.

Figure 1 shows a representation of where each attack would influence the lifecycle of the ML model, providing a clear view of the critical points of vulnerability. These attacks constitute significant threats to the integrity and security of Machine Learning Systems. The following are the types of attacks illustrated in Figure 1:

- **Poisoning Attacks:** Poisoning attacks are also known as causative attacks. The adversary tries to corrupt the training set with the aim that the learned model produces a misclassification that benefits him. This type of attack is carried out in the training phase, and it is very difficult to find the underlying vulnerability, since it can be transmitted to all models that use these data. This kind of attack can be carried out both in white box and black box, and compromises the availability of the models.

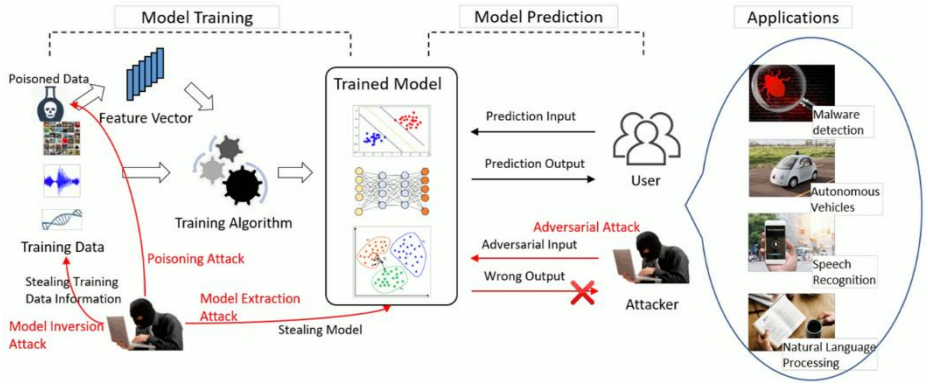

Fig. 1. Classification of adversarial machine attacks.

- **Evasion attacks:** Evasion attacks are known as exploratory attacks. The goal of the adversary is to inject a small amount of noise into the input so that a classifier predicts the output (or label) incorrectly. These noisy inputs, called adversary examples, can be created on different types of data, with the most widespread and well-known examples being images. In fact, adversarial examples of images are created so that they are imperceptible to the human eye.
- **Extraction attacks:** Model extraction attacks consist of an adversary trying to steal the parameters of a machine learning model. This type of attack allows to compromise the intellectual and confidential property of a model, and allows to carry out evasion and/or inversion attacks. This attack can be performed in both white box and black box.
- **Inversion attacks:** The model inversion attacks, consist in that an adversary trying to take advantage of the predictions of the model to compromise the user's privacy or infer whether or not certain data was used in the training set. It can be used in both white box and black box. This type of attack is especially relevant among models that have been trained with sensitive data such as clinical data, which require special protection.

1.1 Related Works

In text data, the discrete nature of the inputs makes the gradient-based attack on images no longer applicable and forces people to craft discrete perturbations on different granularities of text (character-level, word-level, sentence-level, etc.). In this section, we introduce the related work in attacking NLP architectures for different tasks [15].

The work [1] it is an analysis of the generation of abstracts of scientific articles exploring different types of architectures. For this work, 227 NLP articles applied to the tourism sector were collected and 3 types of initial representations were tested to generate the summaries. One based on the title of the article, another adding keywords

and finally another adding important research data. The results indicate that using a method with pre-training, such as GPT-3, can obtain good performance in the task.

The work [2] focuses on the analysis and classification of a set of texts labeled disaster and non-disaster, where those labeled as non-disaster include metaphorical context. The classification is focused on classical models, such as RandomForest, Support Vector Machine, Decision Tree and XGBOOST. This is achieved using SentenceBERT and n-grams as feature extractors, aiming to assess the significance of feature selection in identifying relationships between texts and the importance of specific words.

The work Hot-Flip [5] considers replacing a letter in a sentence in order to mislead a character-level text classifier (each letter is encoded in a vector). The attack algorithm manages to achieve this by finding the most influential letter replacement via gradient information. These adversarial perturbations can be noticed by human readers, but they do not change the content of the text as a whole, nor do they affect human judgments.

The work [11] considers manipulating the victim sentence at the word and phrase level. They try adding, removing or modifying the words and phrases in the sentences. In their approach, the first step is similar to Hot-Flip [5]. For each training sample, they find the most influential letters, called “hot characters”. Then, they label words that have more than 3 “hot characters” as “hot words”. “Hot words” compose “hot phrases”, which are the most influential phrases in sentences. Manipulating these phrases is likely to influence the model prediction, so these phrases compose a “vocabulary” to guide the attack. When an adversary is given a sentence, he can use this vocabulary to find the weakness of the sentence, add one hot phrase, remove a hot phrase in the given sentence, or insert a meaningful fact that is composed of hot phrases.

Deep Word Bug [6] and Text-Bugger [9] are black box attack methods for text classification. The basic idea of the former is to define a scoring strategy to identify the key tokens that will lead to a wrong prediction of the classifier if modified. Then they try four types of “imperceivable” modifications on such tokens: swap, substitution, deletion and insertion, to mislead the classifier. The latter follows the same idea, and improves it by introducing new scoring functions.

The works of Samanta and Mehta [13], Iyyer et al. [8] start by crafting adversarial sentences that grammarly correct and maintain the syntax structure of the original sentence. Samanta and Mehta [13] achieve this using synonyms to replace original words, or by adding some words that have different meanings in different contexts. On the other hand, Iyyer et al. [8] manage to fool the text classifier by paraphrasing the structure of sentences.

Therefore, this paper analyzes how word processing influences adversary attacks, using a SPAM database to create examples of adversaries. Subsequently, these examples will be tested with classifiers to perform a poisoning attack and determine if they detect them. The objective is to analyze word processing with a focus on creating adversarial examples for spam detection models, aiming to cause the most damage with the least amount of disturbance.

Table 1. Representation of a sample of the dataset.

Category	Message
Ham	So how many days since then?
Ham	Even u dont get in trouble while convincing..j…
Ham	So anyways, you can just go to your gym or wha…
Spam	You have been specially selected to receive a …
Spam	You will be receiving this week's Triple Echo…

Table 2. Representation of the processed text.

Category	Message
Ham	mani day sinc
Ham	even u dont get troubl convinc tel twice tel n…
Ham	anyway go gym whatev love smile hope ok good d…
Spam	special select receiv 3000 award call 08712402…
Spam	receiv week triple echo rington shortli enjoy

2 Methodology

In the development of this work, a Kaggle text database was downloaded for further analysis and testing [14]. The environment used was Google Colaboratory. To start the process, several libraries are used, including NLTK for handling and processing text files in Python, as well as Scipy, Scikit-Learn, Numpy and Matplotlib. Once these libraries are imported, the file with the text messages is loaded from the local environment. The file is analyzed to verify that the database contains texts in English and has no empty or null data; in this case, the data set was complete. The database consists of two columns: one for messages and one for tags. This particular database contains SPAM messages, which are divided into normal messages (Ham) and SPAM messages, with 4825 and 747 messages respectively, as shown in Table 1.

Then the following text preprocessing is performed: a function is applied to convert the text to lowercase. The text is divided into a word list. Special characters and empty words are removed. Subsequently, the words are reduced to their basic form (for example, "running" becomes "run") and punctuation marks are removed. In Table 2 you can see how the processed text from the dataset turned out.

Then, we define a function called make-adversarial that takes a sample of text and generates an adversarial version of it. An adversarial sample is a modified version of the original text that can change the classification of the machine learning model (for example, from "spam" to "ham").

The make-adversary function modifies the original text by replacing or deleting words to change their classification, thus generating an adversarial version of the text.

Based on the code suggested by [13], where three different types of modifications are proposed to alter a regular entry in an adversarial sample: (i) replacement, (ii) insertion and (iii) deletion of words in the text. Its objective is to change the class label of the sample by means of a minimum number of alterations. The pseudocode of the proposed method is given in Algorithm 1.

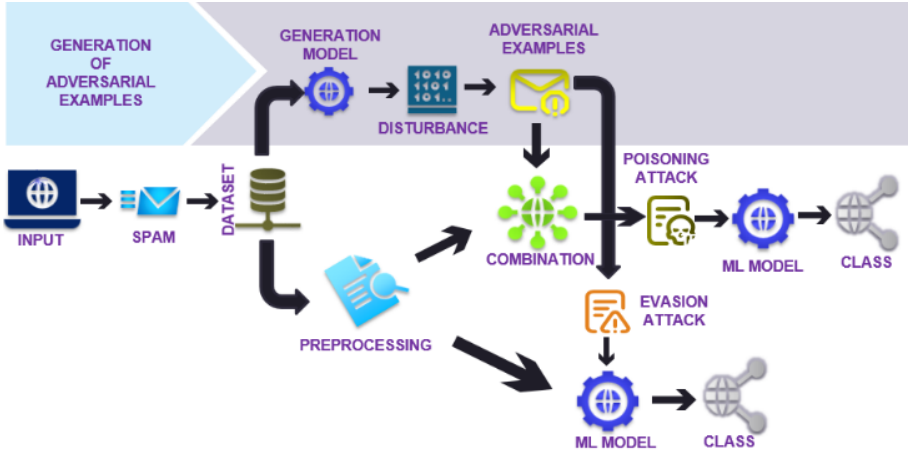

Fig. 2. Outline of the general process of the stages of elaboration of the project.

Algorithm 1. Convert the input text *s* into an adversarial sample

Require: Sample text –*s*, Classifier trained for sentiment analysis *F*.
1: Find class-label *y* of the sample *s: y* ← *F(s)*
2: Find contribution $C_F\ (w_i, y)$ of each word w_i towards determining the class-label of the sample *s* with the respect to the classifier *F*.
3: Rank words according to $C_F\ (\mathbf{w}, y) : \mathbf{w} \rightarrow \{w_1, w_2, \cdots, w_n\}$ where, $C_F\ (w_1, y) > C_F\ (w_2, y) > \cdots > C_F\ (w_n, y)$
4: **while** *y* does not change **do**
5: **if** w_i is an Adverb and $C_F\ (w_i, y)$ is considerably high **then**
6: Remove w_i from *s*
7: **else**
8: Consider a candidate pool $P = \{pk\} \forall k$ for w_i
9: $j \leftarrow$ argmin $C_F\ (p_k, y) \forall\ p_k \in P$
10: k
11: **if** w_i is Adjective and p_j is Adverb **then**
12: Add p_j before w_i in *s*
13: **else**
14: Replace w_i with p_j in *s*
15: **end if**
16: **end if**
17: $y \leftarrow F(s)$
18: $i \leftarrow i + 1$
19: **end while**

Following the process described in the previous pseudocode, a DistilBERT classifier was used to generate adversaries. This model had previously been refined with different datasets, different from those of the current project. To configure it, the HuggingFace Transformers library was used. DistiBERT has its own tokenizer, which preprocesses the texts to adapt them to the model. Since DistilBERT does not support texts larger than 512 tokens, a dictionary was established that instructs the tokenizer to trim texts

to 512 tokens and add padding if necessary, ensuring that all texts are of the same size. Subsequently, the SPAM dataset was uploaded.

The main purpose of the make-adversary function is to generate an adversary text from an original text string. To do this, before applying the algorithm that introduces the modifications to the text, Spacy is used to preprocess and tokenize the text, obtaining a list of Python objects with linguistic information for each token. The spacy-load command loads a language model included in the library, and the model returns a document object. The main idea for creating an adversary is to introduce changes that might confuse a classifier trained in sentiment analysis. DistilBERT is trained on an SPAM dataset. This classifier makes an initial prediction about the text and the result is saved in the sample-class variable.

Then, for each word, its contribution is calculated, which is obtained by deleting the word and using the score returned by the classifier as a contribution measure. The word and its contribution are stored in a dictionary in an orderly way. A high contribution value is obtained using percentiles of the calculated contributions, in this case equivalent to the 90th percentile. Each of the tokens of the text is traversed according to its contribution. It is checked if the token is an adverb with a high contribution and if so, the token is deleted. Otherwise, a second modification is carried out. As a second modification, the token is replaced by some synonym using NLTK.

The WordNet thesaurus is used to find synonyms for that word. If no synonyms are found, the algorithm proceeds with the next word. The best synonym will be stored in the variable P. If there is more than one possible synonym, they are all evaluated by replacing the word with each synonym and checking the classifier score. The synonym that causes the worst score is selected. Then, it is checked whether the original word is an adjective and the synonym is an adverb; if so, the synonym is inserted before the original word. Otherwise, the synonym replaces the original word.

Finally, after any modification, it is evaluated whether it causes a change in the classifier prediction. If so, the class and the adversary text are returned. If, after making all the modifications, the classifier still does not change the class of the original text, None is returned, indicating that the text cannot be corrupted. It is important to note that the function that creates the adversary dataset uses parallelism. The Python multiprocessing library is used, and with the reserved words ”with... as...” a managed context is set where the memory used by the pool variable is freed.

A number of 9 cores and a chunk size of 32 were used. The function ”Pool.imap-unordered” obtains the results in parallel within an iterator. This function automatically distributes the list of texts between the different threads for parallel processing. It is observed in Table 3, examples of the adversary dataset that was generated.

Table 3. Examples of Dataset with the adversary text.

Category 1	Message-G
Spam	urgent! call-option 09066350750 from your land…
Ham	live you uncommitted for soirée along June third?
Ham	Or just brawl that 6times
Ham	That make-up random watch my honest-to-god roo…
Ham	mail ME ir emasinglel singlecalciferol soon

Table 4. Results with the original data with the test set.

Model	Accuracy	Precision	Recall	F1-Score	Class
SVM	0.9865470	0.99	1.00	0.99	0
SVM	0.9865470	0.99	0.92	0.95	1
RandomForest	0.9793721	0.98	1.00	0.99	0
RandomForest	0.9793721	1.00	0.86	0.93	1
AdaBoost	0.9784753	0.98	0.99	0.99	0
AdaBoost	0.9784753	0.96	0.90	0.93	1

Figure 2 shows the general process followed in the methodology, as well as the procedure used to carry out the poisoning and evasion attacks. The data generated by the adversarial example generator was used to execute these attacks. To evaluate the behavior of the data, the SVM, Random Forest and Adaboost classifiers were used.

3 Results

In this section, it will be explained how the poisoning and evasion attacks were carried out with the aim of producing an incorrect classification (see Figure 2). To do this, the SVM, Random Forest and AdaBoost models were used, initially, to analyze the behavior of these models in the face of this type of disturbances and observe their performance. The SVM, Random Forest and Adaboost classifiers were chosen for their efficiency and particular characteristics: SVM excels at classifying high-dimensional data, Random Forest is robust against disturbances and handles complex data well, and Adaboost improves performance by combining several weak models. These models make it possible to compare how different classification approaches respond to adversary attacks.

The hyperparameters used were as follows: For SVM, a linear core with a value of C=1.0 was used. A number of estimators of 100 and a random state of 42 were used for Random Forest. For AdaBoost, it was configured with a number of estimators of 50 and a random state of 42. First of all, we will describe how the final dataset was formed, which contains five columns: the original message, the original tag, the preprocessing of the original message, the adversary message and its adversary tag.

It should be noted that no pre-processing was performed on the adversary message, as this could eliminate the noise that had been included in said message, therefore, this option was discarded. Classes are represented, 0 is not spam and 1 is spam. The data set was divided into 80% for training and 20% for testing, and a 42 seed was established

Table 5. Results with pre-processed data with the test set.

Model	Accuracy	Precision	Recall	F1-Score	Class
SVM	0.9874439	0.99	1.00	0.99	0
SVM	0.9874439	1.00	0.92	0.96	1
RandomForest	0.9802690	0.98	1.00	0.99	0
RandomForest	0.9802690	1.00	0.87	0.93	1
AdaBoost	0.9704035	0.97	0.99	0.98	0
AdaBoost	0.9704035	0.95	0.85	0.90	1

for random status. Tables 4 and 5 show the results of the training of the non-attack models.

From Tables 4 and 5, it is concluded that SVM maintains a very high performance in both data sets, with an overall accuracy close to 99%. Random Forest also shows a high performance, which improves slightly with the pre-processed data, suggesting that it is sensitive to pre-processing and improves its generalizability. On the other hand, AdaBoost shows a slight decrease in accuracy with pre-processed data, indicating that, although it is robust, it is more sensitive to variations in the data.

Text preprocessing seems to have a positive or neutral effect on the performance of SVM and Random Forest, while for AdaBoost, the performance decreases slightly with preprocessed data. This suggests that SVM and Random Forest are more resistant to changes in input data, while AdaBoost may be more sensitive to text preprocessing. The impact of text preprocessing varies between models: SVM shows high and consistent performance on both data sets, suggesting that preprocessing can be beneficial by maintaining accuracy and consistency.

Random Forest experiences a slight improvement with pre-processing, indicating better management of the text features after the initial processing. In contrast, AdaBoost exhibits a decrease in performance with pre-processed data, suggesting a possible loss of crucial information during the pre-processing process.

To analyze the poisoning attack, the variables of the original message with its preprocessing and the adversary message were concatenated into a single column. The same procedure was also performed for the labels. The results obtained are shown in Table 6. The results obtained by applying the poisoning attack to the dataset show a significant decrease in the performance of all models compared to the original or pre-processed data. The poisoning attack has significantly degraded the performance of all models, especially affecting the ability to correctly identify class 0 examples.

SVM and Random Forest show greater comparative resilience, maintaining a better performance in class 1, while AdaBoost is the most vulnerable to attack, with a more pronounced drop in its overall performance. These results underscore the importance of considering robustness against poisoning attacks when selecting or designing models for critical tasks.

Table 6. Results by applying the poisoning attack to the dataset.

Model	Accuracy	Precision	Recall	F1-Score	Class
SVM	0.7354260	0.76	0.41	0.54	0
SVM	0.7354260	0.73	0.92	0.82	1
RandomForest	0.7479820	0.81	0.41	0.55	0
RandomForest	0.7479820	0.73	0.94	0.83	1
AdaBoost	0.6905829	0.66	0.32	0.44	0
AdaBoost	0.6905829	0.70	0.90	0.79	1

Table 7. Results by applying the evasion attack to the dataset.

Model	Accuracy	Precision	Recall	F1-Score	Class
SVM	0.9883408	0.99	1.00	0.99	0
SVM	0.9883408	0.99	0.93	0.96	1
RandomForest	0.9748878	0.97	1.00	0.99	0
RandomForest	0.9748878	1.00	0.83	0.91	1
AdaBoost	0.9757847	0.98	0.99	0.99	0
AdaBoost	0.9757847	0.94	0.90	0.92	1

The analysis of the evasion attack consisted of introducing a small amount of noise at the entrance. In this case, the original labels were used to observe if there was any disturbance in the results, achieving an incorrect prediction as can be seen in Table 7.

The results obtained after applying the evasion attack to the dataset indicate that, despite the disturbances designed to evade detection, the models maintain a relatively high performance. All models retain high accuracy, which suggests remarkable robustness against this type of attack. However, a slight decrease in performance is observed for Class 1 in all models, particularly in Random Forest and AdaBoost. This suggests that the evasion attack has a more significant impact on the classification of certain examples, although not critically. In summary, the models are still effective against evasion attacks, but they present a slight vulnerability that could be exploited under certain conditions.

4 Conclusion

In this paper, the study has shown that, under normal conditions, the SVM, Random Forest and AdaBoost models exhibit outstanding performance in terms of accuracy, Recall and F1-Score. These models show high effectiveness in class detection without the influence of adversary attacks. SVM stands out for its consistency and accuracy close to 99%, while Random Forest and AdaBoost also present solid performance, although with some variations depending on data preprocessing. However, SVM and Random Forest show greater robustness and consistency against adversary attacks compared to AdaBoost.

All models suffer a decrease in performance when faced with poisoning and evasion attacks. The results suggest that SVM and Random Forest have a greater ability to maintain their accuracy and generalization in adverse conditions, while AdaBoost, although effective in normal scenarios, needs improvements to more effectively handle

adversary attacks. These findings highlight the importance of developing additional mitigation and robustness methods to face adversary attacks in classification systems, with the aim of improving the security and stability of the models in real environments.

In addition, poisoning attacks have a significantly greater impact on machine learning models compared to evasion attacks. Poisoning attacks directly alter the training data, degrading the overall performance of the model and leading to systematic errors in its predictions. These attacks manipulate training data, making the model less effective at identifying unwanted emails and negatively affecting its accuracy and detection ability.

In future work, the creation of a generative adversary text model of its own will be explored and more classification models will be analyzed to assess its vulnerability to these attacks.

Acknowledgments. The first author thanks the support provided by the CONAHCYT scholarship number 1106756.

References

1. Alcantara, T., Garcia, O., Calvo, H.: Analysis and Classification of Contextual Disaster Tweets. Research in Computing Science, 152(7), pp. 163–175 (2023)
2. Alvarez, M., Aranda, R., Diaz, A.: Automatic Generator of Scientific Summaries in Tourism Research. Research in Computing Science, 151(5), pp. 5–14 (2022)
3. Aun, Y., Gan, M.L., Wahab, N.: Social Engineering Attack Classifications on Social Media Using Deep Learning. Computers Materials Continua, 74(3), pp. 4917–4931 (2023) doi: 10.32604/cmc.2023.032373.
4. Calvo, J.: Can Deep Learning Be Fooled? (2022) https://www.europeanvalley.es/noticias/se-puede-enganar-al-deep-learning/.
5. Ebrahimi, J., Rao, A., Lowd, D.: Hotflip: White-Box Adversarial Examples for Text Classification. arXiv:1712.06751 (2017) doi: 10.48550/arXiv.1712.06751.
6. Gao, J., Lanchantin, J., Soffa, M.L.: Black-box Generation of Adversarial Text Sequences to Evade Deep Learning Classifiers. In: IEEE Security and Privacy Workshops (SPW), pp. 50–56 (2018) doi: 10.1109/SPW.2018.00016.
7. Gonzalez, S.: Adversarial Machine Learning: Introduction to Attacks on ML models (2022) https://www.welivesecurity.com/laes/2022/05/30/adversarial-machine-learning-introduccion -ataques-modelos-ml/.
8. Iyyer, M., Wieting, J., Gimpel, K.: Adversarial Example Generation with Syntactically Controlled Paraphrase Networks. arXiv:1804.06059 (2018) doi: 10.48550/arXiv.B1804.06059.
9. Li, J., Ji, S., Du, T.: Textbugger: Generating Adversarial Text Against Real-World Applications. arXiv:1812.05271 (2018) doi: 10.48550/arXiv.1812.05271.
10. Li, S., Yun, X., Hao, Z.: A Propagation Model for Social Engineering Botnets in Social Networks. In: 12th International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 423–426 (2011)
11. Liang, B., Li, H., Su, M.: Deep Text Classification Can Be Fooled. arXiv:1704.08006 (2017) doi: 10.48550/arXiv.1704.08006.

12. Lorenzen, C., Agrawal, R., King, J.: Determining Viability of Deep Learning on Cybersecurity Log Analytics. In: IEEE International Conference on Big Data (Big Data). pp. 4806–4811 (2018) doi: 10.1109/BigData.2018.8622165.
13. Samanta, S., Mehta, S.: Towards Crafting Text Adversarial Samples. arXiv:1707.02812 (2017) doi: 10.48550/arXiv.1707.02812.
14. KAAGLE: Spam Text Message Classification (2017) https://www.kaggle.com/datasets/team-ai/spam-text-message-classification.
15. Xu, H., Ma, Y., Liu, H.C.: Adversarial Attacks and Defenses in Images, Graphs and Text: A Review. International Journal of Automation and Computing 17, pp. 151–178 (2020) doi: 10.1007/s11633-019-1211-x.