International Journal of Computer Applications (0975 - 8887)
Volume 170 - No.2, July 2017

A Simple Approach to Automatic Filling CAPTCHA using Pattern Recognition

Ademir B. Santos Neto
Universidade Federal Rural de Pernambuco
Rua Dom Manoel de Medeiros
Recife, Pernambuco - Brazil

Maria da C. M. Batista
Universidade Federal Rural de Pernambuco
Rua Dom Manoel de Medeiros
Recife, Pernambuco - Brazil

Tiago A. E. Ferreira
Universidade Federal Rural de Pernambuco
Rua Dom Manoel de Medeiros
Recife, Pernambuco - Brazil

ABSTRACT

This article shows an easy and simple approach to recognize characters in CAPTCHA images, where the k-NN (k nearest neighbor) algorithm is employed. This proposal to recognize characters in CAPTCHA images has the objective of autofill these components in order to support automation of access to systems. The main aim of this article is to show the steps involved in the proposed process about automatic filling CAPTCHAs since the image's handling until the classification of the characters through a simple and low-cost (implementation) technique of pattern recognition. Experimental results and an error distribution about the characters' classification are showed, where it is demonstrated the possibility of application in real cases of the proposal presented.

General Terms

Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA)

Keywords

CAPTCHA, pattern recognition, classification, automation, character recognition

1. INTRODUCTION

The growth of the world wide web also raised the web bots, generating the necessity of develop mechanisms of security to websites. Websites security involves many mechanisms to protect them against malicious users. One technique that is widely used to prevent such attacks is the use of CAPTCHA. The proposal of CAPTCHA methodology was presented by von Ahnn et al. [17], where it was employed in the real world application for first time in the Yahoo website. Since its creation, the CAPTCHA methodology has been used for many website and a multitude of Internet services. Nowadays, CAPTCHA is a common strategy for websites to prevent attacks from anyone who try to have an automatic interaction with these websites, for example soft robots for automatic information extraction [16, 10].

There are many situations where the use of CAPTCHA is strongly recommended, as for example: on line pools, free email services, search engine boots, worms and Spam prevent dictionary attacks, among others [17]. The most common CAPTCHA's model used basically consist in a set of ordered characters, distorted by some technique to make unclear the whole message, which the human brain can understand, but be more difficult to a machine decipher it.

Although the CAPTCHA is applied to improve the security of the Internet services, there is an daily expectation about 200 millions of CAPTCHAs written in the cybernetic space [10]. If an Internet user take about 10 seconds to understand and write the CAPTCHA code, in the statistical sense, many year-people of work are thrown away per day only with the CAPTCHA human resolve. In the economic sense, an automatic system to resolve the CAPTCHA will bring a cost reduction, at least in lawful Internet transactions.

In order to try to find out the characters that compose the CAPTCHA is necessary to use some classification or recognize algorithm, generally for images or characters. There are some proposals in the literature about techniques to discover CAPTCHA using some classification algorithm, among them it is possible to mention: Support Vector Machine (SVM) [7], k nearest neighbor (k-NN) [2], deep learning [16] and many others [13].

Strategies to auto-fill CAPTCHAs are not something necessarily new. The need to automatize the access to system have been moving many researches for create strategies to decipher CAPTCHAs. For example, in 2008 was detected a span sent by Windows Live accounts trying to sign up in the Microsoft's email service. However these span was getting a success rate in decipher the CAPTCHA only about 30% [19]. Bursztein et al. [2] used non-parametric technique (SVM) to decipher text based CAPTCHAs of many websites, among then Baidu, eBay, Blizzard, with a success rate of 24%, 10% and 50% respectively, these authors also recommend the use of k-NN for its nice stability properties. Gao et al [6] effectively deciphered many CAPTCHAs from Google, Microsoft, Yahoo and Amazon with a success rate ranging from 5% to 77% using a sophisticated technique to get the components of the CAPTCHA and then classifying them with k-NN algorithm. Hussain et al [9]

used a simple image processing techniques and recognition of the characters by using neural network. They got a success rate about 50.48%, 49.69% and 97.5% against CAPTCHAs from eBay, JD and news.cn respectively. Hussain et al. [8] used techniques as thresholding, thinning and pixel count to discover the characters in CAPTCHAs from Taobao, MSN and Ebay along with neural networks achieving an overall precision about 51.3%, 27.1% and 53.2% respectively. There are also other technique of CAPTCHAs that are being adopted based in images like the case of Asirra CAPTCHA [5] that ask to the user to chose between images of cats and dogs. Golle's work [7] shows the use of SVM (support vector machines) to classify the images from Asirra CAPTCHA and he got a success rate of 82.7%. Sivakorn [16] used deep learning techniques to decipher images associates to the new model of CAPTCHA called reCAPTCHA getting good success rate 70.78% to the CAPTCHAs in the reCAPTCHA website and 83.5% in the CAPTCHAs from Facebook.
Therefore, with the sophistication of the CAPTCHA methodology, new and more complex algorithms are proposed to automatization the systems access. However, development and implementation cost of these new algorithms also are improved when compared with simple algorithms. In special, the k nearest neighbor algorithm, or simply k-NN, is an algorithm with low complexity and low run time in computational sense. This algorithm has been applied to discover the characters which compound the CAPTCHA if there is an information database of CAPTCHA images [2]. Furthermore Gao et al. [6] says that most of the proposal to decipher CAPTCHAs are limited to some specific schemes and just a few can resolve a security mechanism as a whole. Hence, the general proposal of this article is show that given a website where the CAPTCHA needs be resolved, a simple procedure can be applied to create a database of CAPTCHA images, where a simple k-NN can be applied to classify the new images and resolves the CAPTCHA code. This approach might be helpful to automatized systems that need to fill a CAPTCHA to get a specific information or enable a service in a lawful solicitation.
This paper is structured as follows. Section 2 talks about the theoretical foundation of the article addressing CAPTCHA and the general theory about k-NN. Section 3 shows the details about the methodology used in this article to find and classify the characters in CAPTCHA images. Section 4 shows the results obtained by the proposed approach. Finally, Section 5 has the conclusion of the article.

2. THEORETICAL FOUNDATION

In this Section is described the theoretical foundation for the methodology applied to the proposed approach. It is important to highlight that CAPTCHA is a sequence of objects formed by a finite set of symbols (or alphabet). In general, this sequence must be recognized by a human as a requirement for some process. In this article, will be assumed that the sequence of objects is a sequence of characters, where the k-NN algorithm was used to cluster and recognize those characters in the CAPTCHA.

2.1 CAPTCHA

Perhaps one of the first articles to talk about identification by "Turing test" was the Naor's article [12]. The article propose a test that a human can easily solve, but a program will fail in the most of the attempts. This is the basic idea about CAPTCHA operation. CAPTCHA is an acronym to Completely Automated Public Turing tests to tell Computers and Humans Apart, they are also called "reverse Turing test" because they aim to allow a computer to determinate from a response if it comes from a computer or a human [2].
CAPTCHA is one of the most widely used mechanics of defense used by websites against undesirable or malicious Internet bot programs [19]. Many commercial websites uses CAPTCHA, as for example: Google, Yahoo and Microsoft. Also some governmental agencies as the Brazilian Ministry of Education[1] uses this kind of mechanism to protect the information of the users against hackers' attack. CAPTCHA is almost a standard in information security and many studies propose new methodologies to design CAPTCHAs in order to avoid automatized systems decipher them [2]. In fact, due the developing of the CAPTCHA's design, today there are many kind of CAPTCHAs used by websites. These models are divided between visual and non-visual CAPTCHAs [14]. Furthermore, the types of CAPTCHAs are based mostly on: text, image, audio, video and puzzle [15]. The most common of these CAPTCHAs are based in visual components, usually these components are based in an identification of a sequence of characters.
The text-based CAPTCHA are the most widely used CAPTCHA's scheme. These kind of CAPTCHAs are based on a sophisticated distortion of characters that are set in a grid, so that most of the humans can decipher it intuitively. This scheme has its popularity due to the fact that it is intuitive to think about associate an image to Roman character for a big amount of users world-wide [3]. A good CAPTCHA should be human friendly whereas be robust enough to resist to the most algorithms which try to decipher it [18]. The Figure 1 shows some examples of text-based CAPTCHAs used by websites.

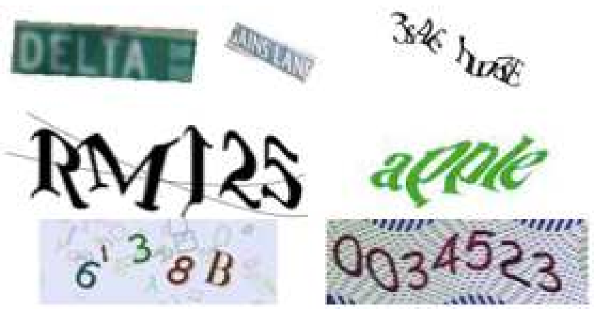


Fig. 1: Example of some text-based CAPTCHAs used by websites

These model of CAPTCHA has many failure models. However, designers learned from previous attacks and developed CAPTCHAs much more sophisticated than the earlier generations [6]. There are many resistance techniques that text-based CAPTCHA uses against attacks. Some of these techniques are: same color letters in the background, background noise, random lines, character collapsing, text distortion, multiples fonts and random strings [9]. Even with the use of these techniques the based text CAPTCHAs present some weak-points points as for example: complex text which the user might can not understand and pattern recognition techniques can recognize the text in the CAPTCHA [1].

[1]This website, at the date of this article, provides information about a governmental program which supply a student credit to high level students in Brazil http://sisfiesaluno.mec.gov.br/seguranca/principal.

2.2 K-NN

Classification algorithms aim to automatically categorize some data in a class by a feature vector [4]. Among those algorithms there is a class denominated nonparametric algorithms, what means that they do not have parameters to make their classification. Some examples of nonparametric algorithm are Density Estimation, Parzen Windows and k-NN [4]. The k-NN (k nearest neighbor) is an algorithm to classify some element in a class depending on the distance of this element to its neighbors. The k-NN is classified as lazy learning algorithm because all the computation is differed until the classification of some element. This algorithm is also one of the simplest machine learn algorithm. The k-Nearest Neighbor algorithm is an widely used technique and many articles propose modification in the algorithm in order to improve its efficient and accuracy to their specific classification, as an example Muralindharan [11] and Zhang [20]. Here, the algorithm used is the standard version of the k-NN.
Let n be an arrangement of all the elements in the universe U where each element belongs to one class C_i ($i = 1, 2, \dots, W$, where there are W classes). The W classes are defined *a priori* and contains elements that have similar traits. Now, suppose that there is an element x and it is inserted in the universe U, the algorithm k-NN try to classify x in one of the class W comparing the traits of x with the other elements present in U (the n elements). For this, the k-NN algorithm measures the distance between x and the k neighbors and check which class these k neighbors belongs to classify x. The Figure 2 shows a distribution of the n elements in the Cartesian axis representing the universe U for three distinct classes and the element x with the k neighbors delimited by the dashed circle.

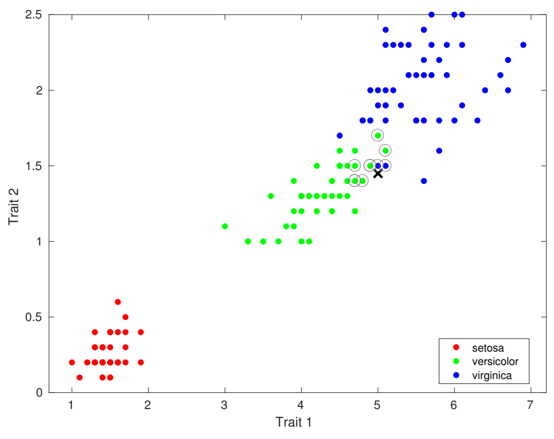


Fig. 2: Example of delimiting the neighborhood in a k-NN classification. The image was generated in the software Matlab version R2015a.

Each element x has its own characteristics (or traits) and to measure the distance D between this element and other element incident in the universe U is possible use a large set of different metrics. In particular, here will be used the Minkowski Metric [21]:

$$D(x, C_i) = \left(\sum_{v=1}^{n_i} |x - x_v|^p \right)^{1/p} \qquad (1)$$

where n_i is the number of elements in the class C_i. Therefore, the Equation (1) is a measure of distance between the element x and all elements of the class C_i. The most common distances used in k-NN classification are the Euclidean Distance (when $p = 2$) and the Manhattan or city block when ($p = 1$) [21]. Finally, the class which the element x belongs will be considered the class that the most of elements among the k neighbors belong
After the classification, when it is known *a priori* which class the element x belongs, we can make a measure of error about the algorithm, this measure can be a confusion table for example. The idea is to show the two types of error that the classification algorithm have presented. The first type of error (Error Type I – false positive, incorrect rejection of a true null hypothesis) is when the algorithm classified an element of a given class as not belonging to this class, and the second type (Error Type II – false negative, incorrect not rejection of a false null hypothesis) is when the algorithm classify an element as belonging to a given class, but in reality this element does not pertain to this class. For example, suppose you have an arrangement of 100 elements to classify and 50 elements of this arrange belongs to the class W_x. The other half belong to other classes in the universe U. Suppose that the algorithm classify 40 elements (of the 50 elements of the class W_x) that belong to the class W_x and 10 other elements to other class. The other elements that do not belong to the class W_x the algorithm classifieds 45 to others class in the universe U and 5 to the class W_x. A confusion table for this classification is presented in the Table 1.

Table 1. : Confusion Table for the showing the error about a classification using k-NN algorithm.

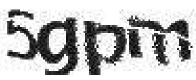
		Classified Data	
		$\boldsymbol{W_x}$	$\overline{\boldsymbol{W_x}}$
Real Data	$\boldsymbol{W_x}$	40	10
	$\overline{\boldsymbol{W_x}}$	5	45

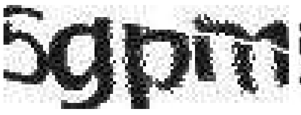
According to the Table 1 the error type II (false negative) was $10/100$ or 10%, what means the classifications that the algorithm did for the other class $\overline{W_x}$, but the elements in fact belonged to the class W_x . The error type I (false positive) for this example was $5/100$ or 5%, what means the number of elements classified as W_x that does not belongs to this class divided by the amount of experiments. The total error of this example is the sum of the error type I and the error type II, what is $15/100$ or 15% of error in the classification.

3. METHODOLOGY

This article shows an approach to autofill CAPTCHA from some website following a set of steps. These steps are: collect CAPTCHA images from the target website, processing the image, make a clusterization dividing similar characters into clusters, give a label to each cluster and then correctly classify the characters in the CAPTCHA in order to autofill this component for automate some process.
To demonstrate how the methodology of this proposal works it will be shown an example of CAPTCHA auto-fill through the steps involved in this approach. The website used in this example has lawful transactions where anyone can have access only providing a correct key. Initially the algorithm download many CAPTCHA images to define the CAPTCHA alphabet, this images are in the way

(a) fig I


(b) fig II


(c) fig III

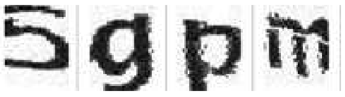

(d) fig IV

Fig. 3: Steps of image's pre-processing.

that the user visualize when the website is charged. The Figure 3.(I) is an example of image that was download from this website. In order to discover how many charterers this CAPTCHA present was download around 2000 CAPTCHAs. After processing these images the algorithm find the number of elements in the CAPTCHA and the alphabet. For the analyzed studied case, the algorithm find out that the CAPTCHA always had four elements and the elements belongs to an alphabet of 19 characters, they are: *2*, *3*, *4*, *5*, *6*, *7*, *8*, *b*, *c*, *d*, *e*, *f*, *g*, *m*, *n*, *p*, *w*, *x* and *y*. When the algorithm establishes the alphabet it make a cluster to each element and start to increase the database processing the images and adding elements to the clusters of the characters. As much character the cluster have, better is the algorithm classification. However, if there is too many characters in the clusters it is possible that the algorithm run slowly. For these experiment each cluster had around 100 examples of the characters.
The process for auto-fill CAPTCHA has two main phase: the processing and the classification. The processing phase has four steps: take the image, convert in gray scale, remove blank spaces and slice the image (here, the number os CAPTCHA elements is discovered). The Figure 3 shows the results of the steps to this phase.
The Figure 3.(I) shows how the target CAPTCHA is when it is download from the website. The first effort in order to process the image is put it in gray scale. Initially, the algorithm converts the image to *jpg* format where each pixel of the image has one value between 0 and 255 for one shade of red, green and blue (RGB). After convert this image, the algorithm change the value of the RGB for each pixels from the image following the rule: for any value of green, red or blue which is greater than 60 this pixel is converted in white pixels, in other words its RGB is converted to (255, 255, 255); otherwise, this pixel is converted in a black pixel and its RGB will be (0, 0, 0). This first transformation can be noticed in the Figure 3.(II), where in a short comparative the gray background and the lines that difficult the visualization of the image were removed.
After the image be converted in gray scale, the algorithm aim to remove, the blank spaces of the gray background. It is important remove these spaces because the objective is to check the characters that are present in the image and these blank spaces does not provide any relevant information. In order to do that it is made a horizontal search in the image to find the first horizontal pixel that is not white, it means the first pixel with RGB distinct to (255, 255, 255), so it is established the first pixel of the new image. This process is repeated backwards and is obtained the last horizontal picture of the image. A similar process is used in the vertical way of the image, first vertical top to down and is set the new start point of the image and after backwards to get the last vertical point of the image. With these new points is necessary just an execution of a

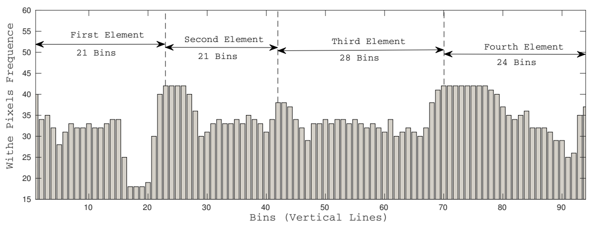


Fig. 4: Histogram of the withe pixels in the vertical lines of the typical treated CAPTCHA image.

cut function passing the new start points as parameters to have the image with no blank spaces as showed in Figure 3.(III).
The last step is slice the image. This step is the most complex in the process. Usually, in text-based CAPTCHAs, the characters are not spaced in the same regions, and sometimes they are overwriting other characters. The basic idea is adopt an heuristic that look for the concentration of white points in each vertical line of the image. Therefore, the concentration of white points in each vertical line of the image is stored in a vector. With this vector, it is possible to make a histogram of the withe pixels where each vertical line is a bin. Therefore, analyzing the maximums of this histogram and the distance between two adjacent maximums is possible infer the number of CAPTCHA elements and where is the localization of each line division between two consecutive elements. In the example of the Figure 3, the target CAPTCHA has four elements and tree division lines. The three bigger points in this vector histogram indicate a huge concentration of white pixels which leads to a great probability that this line is a space between two characters (or elements). With the three larger points in this vector is possible to split the Figure 3.(III) in four characters as showed in the Figure 3.(IV). The distribution of the white pixels of the Figure 3.(III) is:

vector = [[40, 34, 35, 32, 28, 31, 33, 32, 32, 33, 32, 32, 33, 34, 34, 25, 18, 18, 18, 19, 30, 40, ***42****, 42, 42, 42, 40, 36, 30, 31, 33, 34, 33, 33, 34, 33, 35, 34, 33, 31, 34,* ***38****, 38, 37, 34, 32, 29, 33, 33, 34, 33, 34, 34, 33, 34, 33, 32, 33, 32, 31, 34, 30, 31, 32, 31, 30, 32, 38, 41,* ***42****, 42, 42, 42, 42, 42, 42, 42, 41, 40, 37, 35, 34, 35, 36, 32, 32, 32, 31, 29, 29, 25, 26, 35, 37]*

where the three underlined numbers are used to cut the image.The histogram of the $vector$ can be viewed in the Figure 4, where the dashed lines indicate where the image was cut, defining each one of the CAPTCHA elements.
However, due the variant number of blank lines separating two characters, this heuristic sometimes does not works. The key point in this step is after get the first cut point, the next one should be at least 10% of the picture's size more forward than the previous cut point. This additional rule avoid take two cut points close in the vector. The selected cut points for the Figure 3.(III) are the positions 23, 42 and 70 of vector. Adopting this strategy over 80% of the target CAPTCHAs are sliced in the right space, generating to this specific CAPTCHA four images each one with only one character. Lastly, each generated image is processed into a resize algorithm for all of them have the same dimension.
The other phase to autofill the CAPTCHA is classify the character using the k-NN algorithm. The k-NN algorithm was chosen among the classification algorithms because its simple implementation associated to its fast run time. In the algorithm's execution, first the k-Means algorithm [4] was applied to create the clusters of the processed CAPTCHA elements images. For each one of these

Table 2. : Error rate for different values of k

K value	Total Hits for each interaction - Test set										Mean Classification
	1	2	3	4	5	6	7	8	9	10	Error Rate
88	121	119	126	137	123	125	135	118	126	125	0.3394
44	133	134	126	146	131	139	144	137	142	134	0.2810
22	145	149	146	157	146	146	154	145	154	146	0.2168
10	160	157	154	164	154	158	163	152	152	166	0.1684
8	163	161	156	171	159	159	166	158	157	165	0.1500
7	159	157	157	173	161	159	165	160	155	164	0.1526
5	161	156	161	169	164	164	160	156	152	164	0.1542
2	173	163	167	177	168	173	169	156	166	167	0.1163
1	180	173	174	183	175	184	178	171	173	179	0.0684

clusters is associated a representing character. Each cluster has around 100 examples of characters image. All these clusters will be the universe of the k-NN. Once the universe was established, the number k of neighbors is defined. The number k was set initially as $\sqrt{(100 * 19)} \simeq 44$, according some authors recommendations [4]. Subsequently the character that is being classified is measured with all the other elements in the universe by the Euclidean distance ($p = 2$ in the Equation (1)). The 44 elements that have the smallest distance are going to be the neighborhood of the character which is being classified. Finally, the element is classified to the class that appear more in the neighborhood. Taking an example to try classified the character g in the Figure 3.(IV) using the same parameters and the Euclidean distance, the neighborhood of the g is: *neighborhood*(g) = [*4*, *4*, *4*, *6*, *6*, *6*, *6*, *c*, *c*, *d*, *d*, *d*, *d*, *d*, *d*, *d*, *d*, *d*, *e*, *e*, *e*, *f*, *g*, *g*, *g*, *g*, *g*, *g*, *g*, *g*, *g*, *g*, *g*, *g*, *g*, *g*, *g*, *g*, *n*, *n*, *n*, *n*, *n*, *p*]

For this vector is possible to notice that the dominant class is *g* with around 36% of occurrence in all the vector. Therefore for this character and under these conditions the algorithm correctly classified the character.

4. EXPERIMENT

In order to evaluate the algorithm's efficiency was performed an experiment for classify some characters collected in the same website analysed in the Section 3. The universe of the experiment contain 19 classes, each one representing one character in the alphabet of the target CAPTCHA. For each class were collected 100 examples of characters. The value of the k was chosen by an experiment where was verified which value minimize the error rate of the algorithm. In this experiment were tried 9 different values of k for 10 different experiments, where the method of resampling called bagging was applied. The bagging [4], whose name is derived from bootstrap aggregation, uses multiple versions of a training set. Here, the characters images set has 1900 members (100 images for each character). It was created randomly 10 training subsets from this original set, where each training subset was composed by 90% of the images for each character, totalizing 1710 images (90 images for each character). For each one of theses 10 subset, the respective 10% of data not used were employed to test the classification performance of the classifier. The idea about take the average of the error rate comes from bagging experiments, where it is expected that more than one sample will conduct to a result more accurate once. The Table 2 shows the total hits associate to different values for k in the 10 experiments and the mean classification error associate to each value of k for the test set (190 images, 10 images for each one character).

According with the Table 2 is possible to verify that the value of k achieves the minimum error rate when it is equal to 1. When k is equal to 1, the k-NN algorithm is a special case called Parzen Window, according to Duda [4]. This error rate is obtained by the average of errors of all the experiments. The average error in the 10 experiments was approximately 6.8% when $k = 1$. This classification error is for a single character. Therefore, since the target CAPTCHA has 4 characters, the probability to classify correctly a sequence of 4 characters is $(1 - (6.8/100))^4 * 100 = 75,31 \approx 75\%$, *i.e.* on average, for each 4 attempts to resolve the CAPTCHA, 3 attempts obtain success. With this classification error rate, the proposed methodology has a classification performance similar to many others classification methods applied to CAPTCHA recognize. However, the proposed methodology is much simpler when compared with those papers (Section 1), presenting a computational cost in time very low and a quick computational implementation.

As informed in Section 3, the metric used to check the distance between the points were the Euclidean distance ($p = 2$ in the Equation (1)). The experimental configuration, like the division of 90% – 10% of the data to training and testing, the experiments numbers (10 experiments), were chosen after some investigation of articles in the literature about decipher CAPTCHAs. Having this prior knowledge the experiment was planned to try classify 10 characters of each class obtained from the target CAPTCHA (the test set). For this experiment, all the 10 characters were randomly chosen as the target website provide it. These steps basically consist in use the k-NN algorithm to classify each character and classify to which class this character belongs. The Table 3 shows the confusion table of the generated classifier based on k-NN algorithm, where the real data are in the lines and the classified data are in the columns, in same sense of the Table 1.

Table 3. : Confusion table about the error of the algorithm

	2	3	4	5	6	7	8	b	c	d	e	f	g	m	n	p	w	x	y
2	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	9	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	8	1	0	0	0	0	1	0	0	0	0	0
c	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0	2	0	8	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
g	0	0	0	0	0	2	0	0	0	1	0	0	7	0	0	0	0	0	0
m	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0
n	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0
p	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0
y	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	8

From the Table 3 is possible to check the error of the classification by grouping the misclassification that appears in the line of the characters that does not make part of the same class. For example taking the character e, in this simulation, the algorithm classified 8 for the actual class that the elements belong and classified others 2 to other class. These information leads us to know that the error associated with the algorithm to the character *e* is the misclassification divided by the total number of classifications what for this case is $2/10 = 20\%$. Furthermore, it is observed that 13 of the 19 characters were optimally classified. Thus, there are misclassification only in 6 characters, or 31,6% of the CAPTCHA generator alphabet. The average error associate to the classification algorithm to all the characters is 6.8%, as reported previously, implying that the algorithm proposed hits, approximately, about three CAPTCHAs in each four attempts. The experiment was conducted in a standard computer (with a 2.2GHz Intel Core i5 CPU and 4 GB RAM). The total time to run the processing and the classification was in average 16 seconds for each attempt to decipher the target CAPTCHA.

5. CONCLUSION

This article showed an automatized process to fill a CAPTCHA of any website used often by someone, which presents: low computing cost, fast implementation and an equivalent classification error when compared with many others articles.
Based on the results of the experiments, was exposed that it is possible to automatize a system that can use a simple machine learning algorithm and classify with a good margin of error the character. The algorithm correctly decipher about 75% of the CAPTCHAs, what is a good hit rate when compared with results of previews works, as for example Gao et al. [6] and Hussain et al. [8].
As observed in the Table 3, the confusion table of the proposed classifier, some characters have an individual classification error and other characters have not misclassification. However, during the execution of the experiments was also observed that this configuration of which character has or has not misclassification is dependent of the k value. In this way, as further work, an investigation is necessary, in order to decrease the classification error, to propose a methodology to combine distinct classification models based on k-NN algorithm.
Due the simplicity and the fast run time of the algorithm associated with the use in an automatized system, this approach can be well applied in a context of trial and error to get information. Many applications need information, sometimes in real time, from websites and some of these web services uses some security mechanisms as CAPTCHAs. Therefore to all these applications which try to have an automatic interaction with websites the presented approach is well recommended.
Finally, as an ethic consensus, the proposed approach is dedicated to system that needs to automate a lawful operation. The target website, used in the example allows the access of the content. The information in this website is of public domain. Therefore, these ethics conditions were employed for the work proposed here, where there was no illegal action during the article's production. Finally is expected that the information in this article can support anyone who need a simple and efficient way to auto-fill CAPTCHAs.

Acknowledgements

The authors would like to thank UFRPE (Universidade Federal Rural de Pernambuco) for provide the necessary physical support to this research's developing. Also we would like to thank FACEPE (Fundação de Amparo a Ciência no Estado de Pernambuco) for the financial support to produce this research.

6. REFERENCES

[1] Karmand Abdalla and Mehmet Kaya. An evaluation of different types of captcha: Effectiveness, user-friendliness and limitations. *International Journal of Scientific Research in Information Systems and Engineering (IJSRISE)*, 2(3), 2017.

[2] Elie Bursztein, Matthieu Martin, and John Mitchell. Text-based captcha strengths and weaknesses. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 125–138. ACM, 2011.

[3] Kumar Chellapilla, Kevin Larson, Patrice Y Simard, and Mary Czerwinski. Building segmentation based human-friendly human interaction proofs (hips). In *2nd International Workshop on Human Interactive Proofs*, pages 1–26. Springer, 2005.

[4] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[5] Jeremy Elson, John R Douceur, Jon Howell, and Jared Saul. Asirra: a captcha that exploits interest-aligned manual image categorization. In *ACM Conference on Computer and Communications Security*, volume 7, pages 366–374, 2007.

[6] Haichang Gao, Jeff Yan, Fang Cao, Zhengya Zhang, Lei Lei, Mengyun Tang, Ping Zhang, Xin Zhou, Xuqin Wang, and Jiawei Li. A simple generic attack on text captchas. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, USA, 2016.

[7] Philippe Golle. Machine learning attacks against the asirra captcha. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 535–542. ACM, 2008.

[8] Rafaqat Hussain, Hui Gao, and Riaz Ahmed Shaikh. Segmentation of connected characters in text-based captchas for intelligent character recognition. *Multimedia Tools and Applications*, pages 1–15, 2016.

[9] Rafaqat Hussain, Kamlesh Kumar, Hui Gao, and Imran Khan. Recognition of merged characters in text based captchas. In *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on*, pages 3917–3921. IEEE, 2016.

[10] H. KardanMoghaddam. Proposing an algorithm for converting published and handwritten texts to captcha by using image processing. In *2016 Eighth International Conference on Information and Knowledge Technology (IKT)*, pages 170–176, Sept 2016.

[11] R Muralidharan and C Chandrasekar. Object recognition using svm-knn based on geometric moment invariant. *International Journal of Computer Trends and Technology-July to Aug*, (2011):215–220, 2011.

[12] Moni Naor. Verification of a human in the loop or identification via the turing test. *Unpublished draft from http://www. wisdom. weizmann. ac. il/˜ naor/PAPERS/human abs. html*, 1996.

[13] Sonal Paliwal, Rajesh Shyam Singh, and Mandoria H. L. A survey on various text detection and extraction techniques from videos and images. *International Journal of Computer Science Engineering and Information Technology Research (IJCSEITR)*, 6:1–10, 2016.

[14] V Premanand, A Meiappane, and V Arulalan V Arulalan. Survey on captcha and its techniques for bot protection. *International Journal of Computer Applications*, 109(5):1–4, 2015.

[15] Ved Prakash Singh and Preet Pal. Survey of different types of captcha. *International Journal of Computer Science and Information Technologies*, 5(2):2242–2245, 2014.

[16] S. Sivakorn, I. Polakis, and A. D. Keromytis. I am robot: (deep) learning to break semantic image captchas. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 388–403, March 2016.

[17] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. Captcha: Using hard ai problems for security. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–311. Springer, 2003.

[18] Jeff Yan and Ahmad Salah El Ahmad. Breaking visual captchas with naive pattern recognition algorithms. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 279–291. IEEE, 2007.

[19] Jeff Yan and Ahmad Salah El Ahmad. A low-cost attack on a microsoft captcha. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 543–554. ACM, 2008.

[20] Hao Zhang, Alexander C Berg, Michael Maire, and Jitendra Malik. Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2126–2136. IEEE, 2006.

[21] Wolfhart Zimmermann. The power counting theorem for minkowski metric. *Communications in Mathematical Physics*, 11(1):1–8, 1968.