Deciphering CAPTCHAs Using Machine Learning

Sagun Bajracharya
Department of Computer Science
University of Toronto
10 King's College Road, Room 3302
sagun@cs.toronto.edu

Abstract

A Completely Automated Public Turing test to Tell Computers and Humans Apart (CAPTCHA) is the most prominent form of Human Interaction Proof (HIP) on the internet. They usually consist of a combination of letters and numbers that have been distorted to make it difficult for computers to predict. In this paper we evaluate the effectiveness of 3 different machine learning algorithms in breaking gimpy CAPTCHAs from Captcha.net. We begin with a deterministic segmentation algorithm to separate each CAPTCHA into a set of letters. We then apply Naive Bayes, Probabilistic Principal Component Analysis (PPCA) and a Feed Forward Neural Network (FFNN) to classify each of the letters. If any of the above mentioned classifiers are capable of predicting the CAPTCHAs with a success rate significantly greater than 0.01% then the CAPTCHA can be considered broken and cannot be used as a HIP [1].

1 Introduction

The task of distinguishing humans and computers is a fairly old concept that dates back to the original Turing Test [2]. This task has become increasingly more popular as the number of internet services have grown (such as e-mail, social media, online banking etc..). Without an automatic filter to differentiate between machines and humans, malicious programs can consume resources that would otherwise have been spent providing services to humans. Systems that are capable of effectively distinguishing between other machines and people are called HIPs [3].

An effective HIP should be a challenge that humans are capable of solving more than 80% of the time, while machines should succeed less than 1 in 10,000 tries [1]. This constraint has led to the development of various types of CAPTCHAs. An overview of CAPTCHAs can be found on www.captcha.net.

Most CAPTCHAs on the internet are in the form of an Optical Character Recognition (OCR) challenge, where users are required to decipher warped letters and numbers on various backgrounds to successfully overcome the HIP. While humans are relatively good at predicting characters under various transformations, machines have significant difficulty. Thus, by studying CAPTCHA breaking as a subset of the OCR problem, this project aims to shed light into various methods of overcoming limitations that deterministic algorithms have. In particular, 3 algorithms will be considered and tested on the CAPTCHAs. If any of the algorithms are capable of successfully predicting the CAPTCHAs significantly more than 0.01% of the time then the CAPTCHA can be considered broken.

2 Approach

CAPTCHAs can either be classified as a whole or letter by letter. When CAPTCHAs consist of complete words, language models can be exploited to improve classification accuracy. Mori and Malik [4] demonstrated the use of bigrams in breaking the EZ-Gimpy CAPTCHA set with 92% accuracy.

However, the CAPTCHAs used in this project consist of four random letters that have been warped on various backgrounds. Hence a language model cannot be used. For the purpose of this project we will aim to break the CAPTCHAs by analyzing them letter by letter similar to the approach of Chellapila and Patrice [1]. Note that Chellapila and Patrice only used a convolution neural network to classify each of the letters. Here we will use Naive Bayes, PPCA, and a FFNN to classify each of the letters. Chellapila and Patrice were capable of achieving a 90% letter recognition accuracy on the EZ-Gimpy CAPTCHAs, which translated to only 34.4% accuracy in breaking the entire CAPTCHA [1]. Since the CAPTCHAs we will be analyzing use similar warping as the EZ-Gimpy data set, we will compare our performance to Chellapila and Patrice by comparing our letter recognition accuracy with theirs. We will not compare our total accuracy of breaking the CAPTCHA with them since the length of our CAPTCHAs is different than theirs.

2.1 Segmentation

Figure 1 shows some examples of the types of CAPTCHAs that are considered in this paper.

Figure 1: Example of CAPTCHAs

In order to segment the letters, a simple deterministic algorithm was used. The algorithm presented here is similar to the one presented by Chellapila and Patrice [1]. The main difference is that they chose to use the red channel while we converted the images to grayscale. We chose grayscale instead of the red channel because in our set of CAPTCHAs the text was always black.

In our algorithm the images were converted to grayscale, thresholded to retain only pixels with intensity lower than 0.3 (where 0 is black and 1 is white), binarized (all pixels that did not have intensity 1 were set to 0), inverted, and lastly Connected Component (CC) analysis was done to isolate individual letters. Since the letters come in different sizes and shapes, all the individual letters were resized to 15 x 15 pixels (this was the smallest observed dimension among all segmented images). Roughly 70% of the CAPTCHAs had exactly 4 connected components and were classified as properly segmented. The remaining CAPTCHAs had 3 or less CCs and were broken by dividing the largest CC into 2 or more smaller components. This division did not always yield correct segmentation. Figures 2 and 3 show some examples of segmented CAPTCHAs.

Figure 2: Properly segmented CAPTCHA

Figure 3: Improperly segmented CAPTCHA

Figures 2 and 3 show the segmented versions of the CAPTCHAs shown in figure 1. The segmentation algorithm works sufficiently well when the letters are not connected in the original image. Since our algorithms are applied on a letter basis, even though the entire CAPTCHA may not be segmented properly, at least half of the segmented letters yield good training/testing data. In order to apply the various classifiers described below, each letter was reshaped from a 15x15 matrix to a 225x1 vector.

2.2 Naive Bayes (NB) Classifier

A Naive Bayes classifier works on the assumption that features are independent given the class.

$$p(\mathbf{x}|C_k) = \prod_i p(x_i|C_k) \tag{1}$$

Using eq.1 we then classify each letter based on the decision rule shown in eq. 2.

$$y = \arg\max_k p(C_k) \prod_i p(x_i|C_k) \tag{2}$$

Here, the features are the binary values of each pixel in the segmented image and the classes range from 0 to 25 representing each of the possible alphabets.

2.3 Probabilistic Principal Component Analysis (PPCA)

PPCA uses a generative model to classify each letter. In order to apply PPCA, the training set must first be categorized by letters. For each of the letters, we then compute a subspace by picking the M largest eigenvectors of the covariance matrix C_l described by eq. 3.

$$C_l = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_{il} - \bar{\mathbf{x}}_l)(\mathbf{x}_{il} - \bar{\mathbf{x}}_l)^T \tag{3}$$

In eq. 3 C_l is the data covariance matrix for letter l, $\mathbf{x}_{il}$ is the i^{th} training image for letter l where $l = 0 \ldots 25$ and $\bar{\mathbf{x}}_l$ is the mean of all the training images for letter l. Once the subspace is computed, each test image $\vec{\mathbf{I}}$ can be approximated by eq. 4:

$$\vec{\mathbf{I}} = \bar{\mathbf{x}}_l + \left(\sum_{k=1}^{M} a_{kl}\vec{\mathbf{b}}_{kl}\right) + \vec{\mathbf{e}}_l \tag{4}$$

where $\vec{\mathbf{b}}_{kl}$ is the k^{th} largest eigenvector of C_l, $a_{kl} \sim \mathcal{N}(0, \sigma_{kl}^2), \sigma_{kl}^2$ is the sample variance associated with the k^{th} principal direction in the training data for letter l and $\vec{e}_l \sim \mathcal{N}(0, \sigma_{el}^2 \mathrm{I}_D)$ where $\sigma_{el}^2 = \frac{1}{D}\sum_{k=M+1}^{D}\sigma_{kl}^2$ is the per pixel out-of-subspace variance and D is the dimension of our data [5].

The likelihood of each image is then computed using eq. 5:

$$p(\vec{\mathbf{I}}|\mathcal{M}_l) = \left(\prod_{k=1}^{M} p(a_{kl}|\mathcal{M}_l)\right) p(\vec{\mathbf{e}}_l|\mathcal{M}_l) \tag{5}$$

$$p(a_{kl}|\mathcal{M}_l) = \frac{1}{\sqrt{2\pi}\sigma_{kl}} e^{\frac{-a_{kl}^2}{2\sigma_{kl}^2}}$$

$$p(\vec{\mathbf{e}}_l|\mathcal{M}_l) = \prod_{i=1}^{D} \frac{1}{\sqrt{2\pi}\sigma_{el}} e^{\frac{-e_{jl}^2}{2\sigma_{el}^2}}$$

where a_{kl} can be computed as $\vec{\mathbf{b}}_{kl}^T\vec{\mathbf{I}}$ and σ_{kl}^2 is the k^{th} largest eigenvalue of C_l.

Once the likelihood of the image under each subspace model $\mathcal{M}_l$ is computed, the final

decision rule is given by eq. 6.

$$y = \arg\max_{l} p(\vec{\mathbf{I}}|\mathcal{M}_l) \cdot p(\mathcal{M}_l) \tag{6}$$

2.4 Feed Forward Neural Network (FFNN)

The FFNN considered in this paper consists of 1, 2 and 3 hidden sigmoid layers and a final softmax layer to classify the segmented letters. Training is done via back propagation and the test set is classified through a single forward pass on the FFNN using the trained weights. For a detailed description on the implementation of a FFNN see [6].

The effects of standard PCA on dimensionality reduction prior to training and classification using the FFNN is also considered.

3 Experiments and Results

In this section we describe the experiments conducted to tune each of the classifiers and the final classification accuracies achieved as a result of the training. The data set consisted of 560 CAPTCHAs. All training and validation is limited to the first 75% of the data (i.e. the first 420 CAPTCHAs). Testing is only conducted once the model has been generated and fixed. It is conducted on the remaining 140 CAPTCHAs.

3.1 NB Classifier

Since the NB classifier does not have any hyper parameters that need to be tuned, the weights $p(\mathbf{x}|C_k)$ and $p(C_k)$ were trained on all of the 420 CAPTCHAs set aside for training and validation. The letter training accuracy was 67%. This translates to a CAPTCHA training accuracy of $0.67^4 \times 100\% = 20\%$.

The decision rule explained in eq. 2 was then applied to the test data to yield a letter classification accuracy of 56% which translates to a CAPTCHA testing accuracy of $0.56^4 \times 100\% = 9.8\%$.

3.2 PPCA Classifier

For the PPCA classifier, the main parameter to be tuned is the subspace dimension (that is the number of eigenvectors used to approximate each letter).

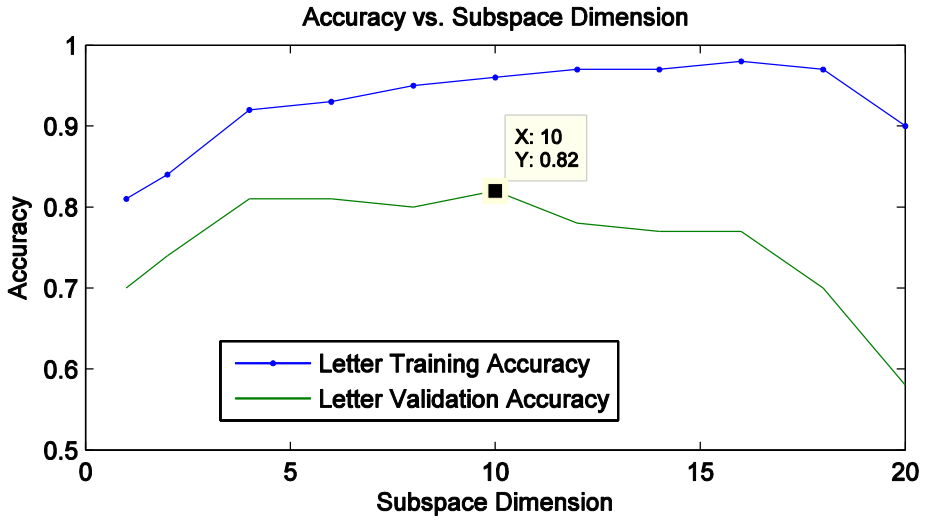

Figure 4: PPCA accuracy as a function of the number of basis vectors used

Figure 4 shows the letter training and validation accuracies achieved for the various subspace dimensions. Training was conducted on the first 75% of the 420 CAPTCHAs set aside for training and validation. The remaining 25% was used as the validation set. Since the validation accuracy is highest when the subspace dimension is 10, the final PPCA model was generated using only 10 basis vectors. Figure 5 shows the 10 basis vectors for the letter 'a'.

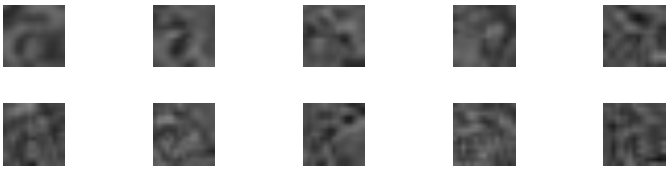

Figure 5: The 10 largest eigenimages for the letter 'a'

Using PPCA with only 10 basis vectors, a letter testing accuracy of 73.75% was achieved. This is a CAPTCHA testing accuracy of $0.7375^4 \times 100\% = 29.58\%$ which is significantly better than the testing accuracy of the NB classifier.

3.3 FFNN Classifier

The FFNN has multiple parameters to be tuned. In order to identify the best parameters, a one layer FFNN was trained for 200 epochs for varying values of batch size, eps, momentum, L2 regularization, and the number of hidden units. Since all the parameters had to be tuned simultaneously there are too many values to present and are hence omitted for brevity. The final set of parameters chosen for training are shown in table 1.

Table 1: Optimum Parameters for FFNN

	Batch Size	**EPS**	**Momentum**	**L2**	**Hidden Units**
Value	150	0.01	0.9	0.0001	100

The training was conducted on 315 of the 420 CAPTCHAs set aside for training. The model was validated on the remaining 105 CAPTCHAs to test the generalization ability of the model. Early stopping was used to terminate the training if the validation accuracy dropped for more than 2 successive turns or if the training accuracy reached 100%.

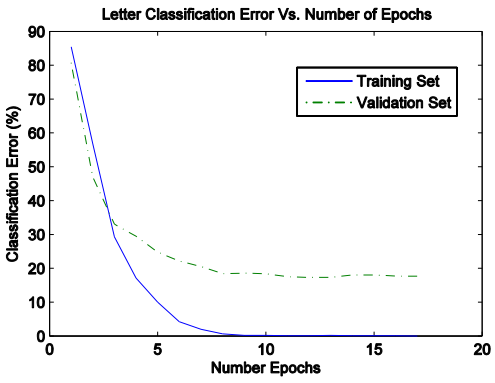

Figure 6: Letter Classification Error for 1 layer network

Figure 6 shows the letter training and validation error curves for a 1-layer FFNN. Using the parameters mentioned in table 1 and the model trained on the first 315 CAPTCHAs a letter testing accuracy of 76.79% was achieved. A similar model was generated using a 2-layer and 3-layer FFNN. However, the 2-layer and 3-layer FFNNs achieved letter testing accuracies of only 76.25% and 75.68%. The training time also went up significantly to 100+ epochs for the 2-layer network and 400+ epochs for the 3-layer network. The slight deterioration in the testing accuracies for the deeper FFNNs is most likely due to the existence of poorly segmented letters in the training data. The fact that the training data is fairly small also increases the chances of the 2-layer and 3-layer FFNNs learning erroneous features. Hence, the effects of standard PCA to reduce the data dimension is not studied for the higher order FFNNs.

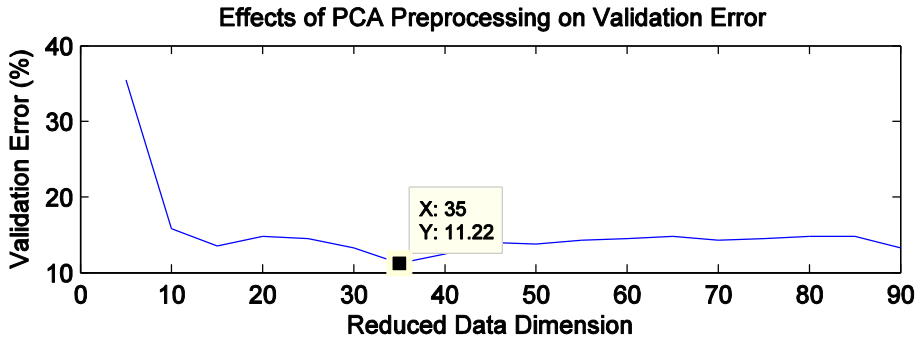

Figure 7: Variation in letter validation error as the data dimension is reduced via PCA

Figure 7 shows the effects of PCA on the letter validation accuracy. It tells us that the letter validation error is lowest when the data is projected down to 35 dimensions. Using this projection, the 1-layer FFNN was capable of achieving an optimum letter testing accuracy of 84.11%. This results in the highest CAPTCHA testing accuracy of $0.8411^4 \times 100\% = 50.05\%$ in comparison to the NB and PPCA classifiers.

4 Conclusion

Looking at the results presented in section 3 it is apparent that the FFNN used in conjunction with standard PCA is the most powerful classifier of the three. It achieved a 50.05% CAPTCHA recognition rate as opposed to the 29.58% and 9.8% classification rates of the NB and PPCA classifiers. The letter recognition accuracy of the FFNN used in conjunction with standard PCA was 84.11%. This is only about 6% less than the letter recognition accuracy presented by Chellapila and Simard. Hence, from our results we can conclude that the effectiveness of a simple FFNN can be boosted using PCA and it can achieve results close to that of a convolution neural network for OCR. We can also conclude that this set of CAPTCHAs cannot be used as a HIP since the CAPTCHA classification rate of the FFNN was 50% which is significantly greater than 0.01%.

References

[1] Chellapila K. & Simard P.Y. (2004) Using Machine Learning to Break Visual Human Interaction Proofs (HIPS). *Advances in Neural Information Processing Systems 17*, pp. 265-272. Cambridge, MA: MIT Press.

[2] Turing A.M. (1950) Computing Machinery and Intelligence. *Mind*, 59:236, pp. 433-460.

[3] *First Workshop on Human Interactive Proofs*, Palo Alto, CA, January 2002.

[4] Mori G. & Malik J. (2003) Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA. *Proc. of the Computer Vision and Pattern Recognition (CVPR) Conference*, IEEE Computer Society, vol.1, pp. 134-141.

[5] Jepson A.D. & Fleet D.J. (2011) *CSC2503: Linear Subspace Models*. pp. 15.

[6] Bishop C.M. (2006) *Pattern Recognition and Machine Learning*. New York: Springer.