A Generalized Method to Solve Text-Based CAPTCHAs

Jason Ma, Bilal Badaoui, Emile Chamoun

December 11, 2009

1 Abstract

We present work in progress on the automated solving of text-based CAPTCHAs. Our method consists of two large components: firstly, we segment the text from the background. Afterwards, in order to identify the text, we extract features and use learning algorithms to classify each character. In particular, we developed a weighted k-means algorithm, which is largely able to segment the text portion of the CAPTCHA image. Additionally, we developed an algorithm to separate the individual characters. Finally, we explored classifying the separated characters using several methods, including through support vector machines and optical character recognition algorithms. We also present some current challenges we face in this algorithm, and some ideas we plan on exploring to overcome them.

2 Background information

A completely automated public Turing test for telling computers and humans apart, or CAPTCHA, is used in many web-based applications. The most prevalent CAPTCHAs consist of an image of a sequence of distorted letters, which is purportedly difficult for a computer to solve, but easy for humans. CAPTCHAs are used to prevent the automation of certain tasks; for instance, they are often found on website registration forms, which helps ensure that each registration is from a human. Previous research has been traditionally focused on solving single specific types of CAPTCHAs.

3 Data retreival

We began by obtaining 1000 CAPTCHAs from the ICQ registration site, 400 CAPTCHAs from Gmail, 400 CAPTCHAs from Yahoo!, and 400 CAPTCHAs from Kongregate. In addition, we used a small number of CAPTCHAs from other generators, in order to ensure that our algorithm is general.

4 Segmentation

We explored five main areas, comprising the segmentation portion of our method.

4.1 Weighted k-means

4.1.1 k-means

We began by exploring k-means clustering of the colors (treating each pixel as a vector in $\mathcal{R}^3$). By compressing the colorspace of the image, we hope to segment the text from the background. It is likely that the text has different color values, as otherwise humans will not be able to identify the text easily. Naturally, then, if we were able to cluster the pixels so as to place the text and the background into separate clusters, we would be able to more easily recognize the text.

Unfortunately, simple k-means does not perform well in certain cases, especially when the both the background and text vary in color. Moreover, simple k-means performs poorly when the text consists of progressively differing colors, as the different parts of characters will be placed in different clusters. Other weaknesses include when the text is obscured by lines that extend across the text, as the lines themselves will be clustered with the text.

4.1.2 Weighted k-means

To surmount these challenges in simple k-means, we investigated an enhanced algorithm in which we consider both the color and the location of the pixel. Specifically, for each pixel, the distance to the centroid is defined as a weighted combination of the Euclidean distance between the respective colors, and the Euclidean distance between the geometric locations of the pixels within the image. By adjusting the weight of the two components, we hope to strike a balance and cluster pixels that are of a similar color and location.

Whereas simple k-means performed poorly on certain CAPTCHAs, with the proper weights and the proper k, weighted k-means performs significantly better.

Below, we show three figures comparing the performance of weighted and unweighted k-means. Figure 1 shows the original CAPTCHA, which is difficult to segment due to the background. Figure 2 shows the result when simple k-means is used; Figure 3 shows the corresponding result for weighted k-means. Both algorithms used $k = 15$ clusters; the weighted k-means algorithm weighted the proximity with a factor of $w = 4$. By weighting the k-means algorithm, we are able to cluster the text into a small number of clusters, with each character completely within one cluster.

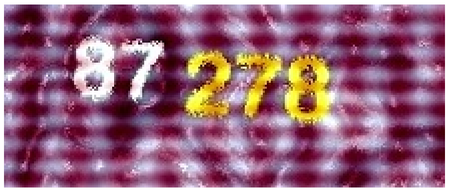

Figure 1: A CAPTCHA from ICQ

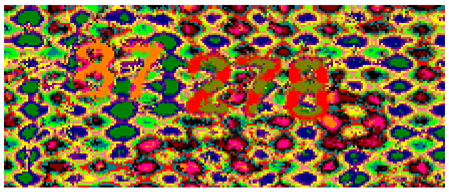

Figure 2: Unweighted k-means

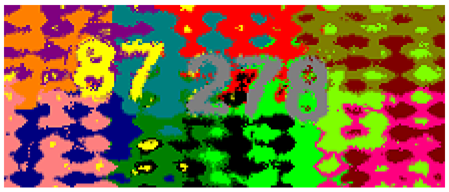

Figure 3: Weighted k-means

4.2 Vertical cuts segmentation

As can be seen in Figure 3, weighted k-means can cluster multiple characters into the same cluster. As classification requires individual letters, separation of clusters is necessary. To accomplish this, we first determine the orientation of the text. Then, we discretize the image space into lines that are perpendicular to the dominant orientation. Characters are separated by lines whose change in pixel density is greatest.

To find the orientation of the text, we compute the first principal component of all the points in the text. Subsequently, to simplify the discretization step, we rotate the image so that this principal component is horizontal. Next, we project the image on the principal component, and find the points with highest differences in intensity. We calculate the mean and standard deviation of the changes in intensity and, for all points that lie c standard deviations above the mean, we make a vertical cut to divide the cluster into two parts. With a continuous region of high difference, we make the cut at the region's median to minimize information loss.

4.3 Minimum spanning tree

One of the approaches we tried was using the minimum spanning tree to segment the CAPTCHAs. In essence, minimum spanning tree clustering techniques resemble k-means, with the extra advantage of constructively building the clusters.

Each pixel corresponds to a vertex in the graph. Neighboring pixels within a predetermined window size will be connected by edges, the weight of each edge being a weighted average between the color intensity and proximity (similarly to weighted k-means). We can build the minimum spanning tree resulting from this graph.

First, let us lay out the theoretical foundations of our MST clustering algorithm.

Definition 1.1 For a data set D, $C \subseteq D$ forms a cluster in D only if for any arbitrary partition $C = C_1 \cup C_2$, the closest data point d to $C1$, $d \in D - C_1$, is from C_2.

In essence, by this definition, we are trying to capture our intuition about a cluster; that is: distances between neighbors within a cluster should be smaller than any inter-cluster distances.

4.3.1 Fundamental results

Property 1.2 Expansion Property. If c_1 and c_2 are two points of a cluster C, then all data points in the tree path, P, connecting c_1 and c_2 in the minimum spanning tree, must be from C.

The proof of this result can be found in [4].

Property 1.3 Let c_1 and c_2 be such that c_2 is the closest point to c_1. Then either c_1 defines a cluster alone, or c_1 and c_2 are in the same cluster.

The proof of this result can be found easily by applying the definition 1.1.

4.3.2 Corollaries

We can get from the first fundamental result that a cluster is a connected component of the minimum spanning tree. Hence, clustering is only a matter of pruning certain edges from the minimum spanning tree. The resulting graph will have a number of connected components, defining the clusters.

4.3.3 Exploration clustering algorithm

The algorithm consists of two steps.

1. **Mandatory components.** To avoid singleton clusters, the algorithm first uses the second fundamental result to build agglomerative mandatory components. We start first with each vertex considered as a component. For each point A in a component C, we consider the component D of A's closest neighbor. If $C \neq D$, $C := C \cup D$. We apply iteratively this rule to every component as long as merging is occurring. We stop when the process stabilizes.

2. **Exploration.** In the exploration phase, we take every mandatory component MC. For each point B on the border of MC, we get its closest neighbor N in MC. If the distance between B and N is less than $m + s \times d$ (where m is the mean of distances in MC, d the standard deviation and s is a parameter we will call sensitivity) we merge MC and the mandatory component containing B. After doing this for every point that was on the border, we update the mean and the standard deviation and reiterate.

4.3.4 Results

Unfortunately, the results were not satisfactory at all. The Mandatory Component phase in the algorithm produces a huge initial number of mandatory components (typically in a 100 x 240 picture, the number of Mandatory Components is around 4000). This made the exploration phase very inefficient with mandatory components being merged (due to similar small size) and resulting in creating one very big cluster that would contain more than 99% of the image.

4.4 Weighted k-means noise removal

As discussed earlier, the weighted k-means segmentation manages to create successfully clusters containing text. However, one cluster might contain more than one character (possibly if the k used is small or if the characters are close enough and have very close colors), and the clusters will not necessarily consist of characters only (namely because of the nature of a CAPTCHA: characters and noise nearby might be clustered in the same cluster). For example, after running weighted k-means on the CAPTCHA in Figure 1, one of the clusters obtained is shown in Figure 4.

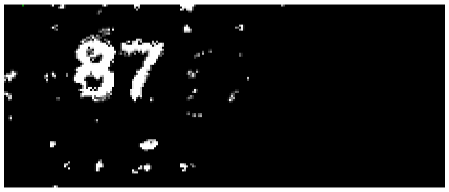

Figure 4: One cluster from weighted k-means

Our noise removal algorithm tackles these challenges. The algorithm is based upon the observation that a cluster found by weighted k-means consists of a set of "continuous" components. The idea is then to break a cluster into these continuous components. Each of these components would be considered a separate cluster provided it is not "too small". To find continuous clusters, we use again a graph structure to connect neighboring cluster pixels. Finding the "continuous" components is equivalent to finding the connected components of the graph. By applying the algorithm on the cluster in Figure 4, we get the following clusters:

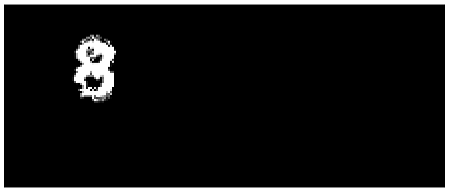

Figure 5: Separated cluster, after noise removal

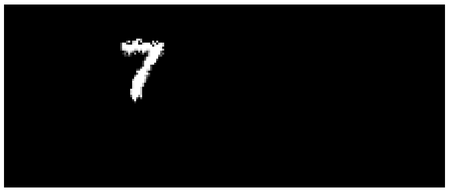

Figure 6: Separated cluster, after noise removal

4.5 Normalized Cuts

The apparent dissimilarities between the text and the background in CAPTCHAS suggested the use of normalized cuts as a potential segmentation method [1].

Effectively, in addition to measuring the total similarity within each cluster, the normalized criterion takes into account the total dissimilarity between different groups.

Essentially, the algorithm models the image as a weighted undirected graph. Each pixel is a node in the graph and is connected by edges to the other pixels in the image. Moreover, a weight indicating the degree of similarity between two pixels is assigned to each edge.

Thus, segments are formed by removing edges from the graph. The normalized cut ratio is defined by the weight of the cut to the weight of the edges in a given segment. The normalized cut segmentation algorithm seeks to minimize the normalized cut ratio, in accordance with the overall goal of minimizing the weight of the edges removed.

Unfortunately, this algorithm did not perform too well on our dataset since the text in the CAPTCHAs often progressively changes intensity which makes it very hard to segment individual letters properly. Effectively, it turns out that this algorithm often splits the same letter into two or more clusters if the different parts of the character share more similarities with the background that with each other.

4.6 Fuzzy c-means Clusterization

We applied fuzzy clustering to the segmented text parts to try to further separate characters in the same cluster. The fuzzy clustering algorithm can be described in the following steps:

```
Chose a number of clusters.

Assign randomly to each point coefficients
that represent their degree of belonging in
a cluster.

Repeat until convergence {
  Compute the centroid of each cluster,
  which is defined as the mean of all
  points in the cluster weighted by their
  degree of belonging to the cluster.

  Update the coefficients to reflect the
  new degree of belonging to the cluster.
}
```

Although this algorithm performed well in separating spaced out characters, it did not succeed in segmenting connected characters as is the case in google CAPTCHAs. In particular, we found the assignment of clusters in this algorithm to heavily depend on the physical location of points in the image. Thus, in the case of connected characters, the algorithm might assign two halves of the same character to the two neighboring clusters if the distance of each half to the neighboring cluster is smaller than the distance that separate them.

5 Classification

We tried a variety of methods to classify the individual segmented characters. This comprises part two of our method. Our training data consisted of the CAPTCHAs collected, as described in the data retrieval section.

5.1 Support vector machines

Our first attempt was to try to classify characters using SVM. We started first by generating a training set consisting of all alphanumeric characters in different shapes, fonts and sizes. Using SURF, we extracted the features from each element in the training set. For each training element, we ran k-means over the features (k was heuristically fixed at 15). We then trained a one-versus-one multi-classifier SVM using the SURF features of the centroids as inputs. The aim was to use the trained SVM on the segmented clusters from the previous section. However, the accuracy of the trained SVM was very low: the SVM had 10% accuracy. Conceptually, this was not very surprising. Consider for examples the characters **E** and **F**. Any interest point in **F** will be in **E**. However, E should contain at least one more interest point (the bottom right edge). Running k-means could only be harmful to the classification of either **E** or **F**. If k is big enough for **E**, the classification of **F** will be harmed by added irrelevant features. The converse would happen if k were small.

5.2 Optical character recognition

Another classification method we considered was optical character recognition. Optical character recognition (OCR) is the electronic translation of text-based images into a computer editable form. Given the success of OCR technology in advanced identify the segmented characters.

Tesseract, which is widely considered as one of most accurate open source OCR engines available, was run on the segmented characters in an attempt to recognize them.

Although Tesseract succeeded in identifying some characters, it turned out to not be suitable for our classification problem for the two following reasons:

1. Tesseract learns the font of the characters in its training set, which consists of scanned electronic documents. Therefore, Tesseract will not succeed in identifying characters whose font differs significantly from that training set. Moreover, training Tesseract on a customized dataset is not feasible since the engine will not be able to learn the font given that different types of CAPTCHAs do not have the same font.

2. Tesseract uses a dictionary to identify whole set of words, as opposed to individual characters. In that sense, Tesseract does poorly when it comes to identifying individual characters. In some cases, it even tries to match a character to a full word in its dictionary if such match is possible.

6 Work in progress

Given the low success rate achieved by out-of-the-box character recognition packages and our intent to continue work on the project, we are currently considering other classification methods outlined in the next sections.

6.1 Back propagation

One idea worth exploring is back propagation. Briefly, similar to the way humans reason about CAPTCHAs, a high confidence classification of a certain segment in a CAPTCHA provides us with valuable information about the "structure" of the text, such as orientation. This information could be propagated back to the segmentation in order to obtain better quality clusters, which will provide better results in general.

6.2 LeNet

Indentifying individual characters from segmented text-based CAPTCHAs requires robustness to font selection and image transformations. In that context, we are currently exploring using convolutional neural networks to try to recognize images with minimal preprocessing. Convolution Neural networks are a form of multi-layer neural networks designed to recognize visual patterns from images. They are robust to distortions and geometric transformations and can recognize patterns with a high degree of variability such as CAPTCHAs.

One such convolutional network is LeNet-5 [3] which is originally designed to identify handwritten and machine-printed character recognition. The fact that LeNet-5 is successfully able to recognize handwritten characters leads us to believe that it might perform very well on CAPTCHAs if trained on a good dataset. Effectively, handwritten characters present features that are very similar to CAPTCHAs such as a high degree of variability in font, distortion as well as the spacing between individual characters.

6.3 Shape context features

Shape context features reflects the distribution of a pixel's neighboring points relative to it[2]. Thus, corresponding points on two related shapes will have similar shape context features. In particular, letters are defined by the relative locations of each pixel's neighbors, and thus shape context features are highly robust to distortions in text, as capturing the dependencies between neighboring pixels is key to successful identification. For this reason, shape context features will likely perform significantly better than SURF in a support vector machine.

7 Conclusion

The key step in solving CAPTCHAs, segmentation to extract individual letters, has been reasonably successful. Through using a variety of algorithms, we reduce color variations in the image, separate clusters, and reduce the noise in each cluster. From our algorithms, we can retrieve with reasonable success a set of binary images that represents each character in the CAPTCHA. Additionally, more work remains to be done in classification. Several ideas have significant potential in this area. These results show promise for a generalized CAPTCHA solver, which would render them ineffective and promote the need of a improved system.

References

[1] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[2] Jitendra Malik Serge Belongie and Jan Puzicha. Shape Matching and Object Recognition Using Shape Contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[3] Y. Bengio Y. LeCun, L. Bottou and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[4] Victor Olman Ying Xu and Dong Xu. Minimum Spanning Trees for Gene Expression Data Clustering. *Genome Informatics*, 2001.