ANALYSIS OF A CONTENT-BASED IMAGE RETRIEVAL SYSTEM USING A WEIGHTED K-NEAREST NEIGHBOR CLASSIFIER ON MULTIPLE-EXAMPLE IMAGE QUERIES

ANDREW RINGLER

Abstract. I assess the issue of retrieving images of interest from a large unclassified database. I address the topic of retrieval using queries consisting of multiple positive and negatively classified images. A method for assessing the similarity of images provided by the user to those in the database is sought. I attack the problem by assuming that each image can be represented by a low dimensional feature vector that preserves similarity comparisons. I propose we can find images similar to the positive query images using a k-Nearest Neighbor classifier about each positive image. A suitable distance metric is needed for the classifier: I propose to use a weighted Euclidean distance metric. The weights of the metric are to be determined iteratively by minimizing the misclassification error of the aforementioned classifier.

Introduction

The graphic artist, the web designer and the advertiser are constantly looking for fresh ideas, new media. With hundreds of millions of images on the Internet, and more appearing every day, the Internet provides this constant source of inspiration. How does one find visual content in this zoo of information? Commonplace are text search engines for finding documents, some engines even providing image search by text query. Text based search of images is limited to descriptions provided by the website host, which are often not to be found. This motivates search techniques based on the images themselves. One such paradigm is search by pictorial example.

I propose query by example is an excellent retrieval mechanism for finding images on the Internet. Some proposed such systems are based on query by single image. These systems are severely limited however. It is often difficult for the user to find a single image representative of the goal image they desire. More often they can find multiple images, each containing some feature that is present in the goal image. As well, it is often difficult for the user to obtain query images on their own. I will provide all such images in my algorithm. This leads us to query by multiple example [4].

I present the user with images from my database. I then let the user decide which are positive examples of the goal image they desire. But then how do we use these newly classified images to search for other similar images? Ideally we would want to generalize the properties of the query images. We could then find images in the database that fit these properties. We would be introducing a segmentation of the database into two sets. Lets denote the set of images that satisfy the aforementioned

Key words and phrases. CBIR, Content-Based Image Retrieval, QBPE, image databases, multiple example, k-Nearest Neighbor, weighted k-Nearest Neighbor.

properties as positive images and all others as negative images. Any procedure which performs such a segmentation is called a Classifier. We could then return the newly classified positive images to the user.

How can we classify our database using the query images? Theoretically the best way to do this is to use a Bayesian classifier. This is a special ideal classifier which unattainable in practice but is instead used to benchmark practical classifiers. There are, however, classifiers which closely approximate the Bayesian classifier. It is well proven that the error of the k-Nearest-Neighbor classifier p_{NN} is no more than twice the error of the ideal Bayesian classifier p_B [2]:

$$p_B \leq p_{NN} \leq 2p_B$$

The error bound, and the simplicity of implementation, led me to use the k-Nearest Neighbor classifier in my search algorithm. I will now explain the user feedback domain in which we can use the above techniques.

1. Image Query by Positive and Negative Example

The domain of query by image using multiple examples not only allows a more descriptive query than that using a single image but also allows for user feedback. In my system the user is initially presented with N random images from the database. The user chooses those images which are representative of the goal image they seek. The images the user chooses are considered positive examples and those the user does not choose are considered negative examples. Negative and Positive are denoted as the Class Labels of the images. Note we will only be actually working with feature vectors of the images, and not the actual images themselves. For the rest of this paper I will use the terms image and the feature vector of that image interchangeably (unless otherwise noted). Let the set of positive image examples be denoted as C_p, the set of negative image examples be denoted as C_n, and the sum of all images seen and classified by the user as:

$$C_p + C_n = C.$$

I will denote the entire image database as I and the set of all images seen but not yet classified by the user:

$$U = I - C.$$

I now want to use the set of user classified images C to help classify the unclassified images U from the large database. Using a classifier which I will discuss later I can partition U into positive and negative images. I could then return to the user a new set of images U_k which representing the k images most similar to the user specified positive images C_p. The user would then classify the new images U_k as they did before. Lets denote the classification of U_k as U_{kp} for positive and U_{kn} for negative examples. Now we have a new set of classified images C, namely

$$(U_{kp} + C_p) + (U_{kn} + C_n) = C.$$

As well we have a new set of unclassified images

$$U = U - C.$$

We can now refine our estimated classification of I, and return to the user another set of images U_k. The process continues, with the user seeing more and more images, our algorithm continuously refining its search, until the user has found his/her goal

image(s). But how do we classify the unclassified images using a small example set of classified images?

2. k-Nearest Neighbor Classifier

A basic k-Nearest Neighbor classifier is straightforward in concept. Let C be a set of classified feature vectors, and U be a set of unclassified feature vectors. Then the class label of each vector U_i can be said to be equal to the majority class label of the k vectors in C closest to U_i. As stated above, the error in this approximation is no more than twice the error of an ideal classifier on the data given an infinite data set. This assumes we have a way to measure closeness of feature vectors. The most common distance measurement (denoted metric) is the Euclidean distance. This gives a good approximation to the closeness of features, but would not allow us to use multiple image queries.

We obtain valuable information by using multiple image queries as compared to single image queries. Namely, we obtain a set of images each of whom has some feature that is present in user's yet to obtained goal image G. Ideally we want a strategy for obtaining G from our set of classified images C. I could propose a simple strategy by returning the k-Nearest Neighbors to each of the classified images C:

$$\min_k(dist(c_{pi}, u_i)), \forall i, j, c_{pi}, u_j \in C_p, U \tag{2.0.1}$$

The equation above returns (for each positive classified image c_p) the k images most similar. There is a fatal flaw with this approach however. For a given image c_{pi} we are finding images that are similar to ALL of its features. The user does not want this but in fact wants images that are similar to the group C_p as a whole.

We could pose the problem as one of minimization as follows:

$$\min(\varepsilon = \sum_{i=1}^{|C_p|} dist(G, c_{pi})) \tag{2.0.2}$$

In fact with an ideal distance metric (dist) all points c_p would converge onto a single point G. We could then find the k-Nearest Neighbors to G. This would represent the k images most similar to ALL of the users query images. We would then be done. But how do we find such a distance metric? I propose to use a Euclidean distance measurement with learned weights.

3. Weighted Euclidean Distance

The Euclidean distance d, between two n-dimensional vectors u and v is defined by:

$$d = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2 + \ldots + (u_n - v_n)^2}$$

and in vector notation as:

$$d = \|u - v\|$$

Now I can denote the weighted Euclidean distance as:

$$d = \|u \cdot W - v \cdot W\| \tag{3.0.3}$$

where W is a diagonal weight matrix. Now I can learn the weights W using a text categorization algorithm WAKNN proposed by Han, Karypis, and Kumar [3].

The (adapted) algorithm is as follows:

(1) Initialize weight matrix to all ones.

$$W =< 1, 1, \ldots, 1 >$$

(2) For each image $c_i \in C$ and denoting the class label (positive or negative) of c_i as $\zeta(c_i)$ calculate using k-Nearest Neighbors whether it is correctly classified. We will say some c_i is correctly classified if the majority c_i's k-Nearest Neighbors have the same class label c_i. I will denote the misclassification error for some c_i as:

$$\epsilon_i = \begin{cases} 0 & \text{if } |\{\alpha | \alpha \in \zeta(c_i) = \zeta(k_j) \forall j \neq i\}| \geq (k/2) \\ 1 & \text{Otherwise} \end{cases} \tag{3.0.4}$$

Where $k_j \in K$ are the k-Nearest Neighbors calculated in equation 2.0.1 and using the weighted Euclidean distance from 3.0.3 with weight matrix W from step 1.

(3) Now we will denote the TOTAL misclassification error ϵ for a given weight matrix W as the sum of all misclassifications of c_i obtained by:

$$\epsilon = \sum_i \epsilon_i \tag{3.0.5}$$

(4) Now we want to minimize the TOTAL number of misclassifications obtained from 3.0.5. We do so by making small changes to W, one dimension at a time. We then choose the update that minimized 3.0.5. The procedure is illustrated in the pseudo code below:

```
for i = 1:|m| {
   for j = 1:|W|  {
       W' = W
       W'_j = W'_j * m_j  Try New Perturbation of W
       Calculate ε_mj using 3.0.5 and W'
    }
}
```

where m is an array of update values namely [.2, .8, 1.5, 2.0, 4.0]. Now W' corresponding to the minimum ϵ_{mj} is our new weight matrix W.

(5) Now we can keep repeating step 4 until we have a small enough ϵ and hence a good weight matrix W

We now have a weight matrix W which minimizes the distance between every positive image c_{pi} and his closest positive neighbors. With any luck we have grouped all the positive images together in the feature space. If they are close enough to each other we can now easily minimize equation 2.0.2. In fact we can use an iterative procedure similar to that above to solve 2.0.2. I will not explain the details of such a procedure here.

We are now very close to a Content-Based Image Retrieval procedure. So far we have a way to let the user specify query data. We have a way to use this data to generalize about the goal image the user is imagining. Lastly we have a way to generate similar images to the abstract image goal of the user. Of course we are missing feature vectors. How do generate feature vectors which are representative of

the image. How do we insure these feature vectors preserve similarity comparisons from the user's perspective?

4. Color Histogram

Color is a large determining factor in the perception of image similarity in humans. For simplicity I have chosen to only implement one feature vector, that of Color. Comparison by color must be made in such a way that the perception of color differences by humans corresponds to that of the computer. Fortunately many computer image formats are designed to closely match the human perception of color. JPEG images are stored using the Red, Green, and Blue components of the image (RGB) which correspond to the three cones in human eyes. Typically we can extract RGB color from an image using the following formula to scale out image intensity [1]:

$$r = R/(R+G+B)$$
$$g = G/(R+G+B)$$
$$b = B/(R+G+B)$$

We can then calculate the average rgb values over regions in the image. Each region average is denoted as one dimension in the feature vector. In my implementation I evenly split the image into 4 rectangular quadrants. I then calculated the red, green, and blue value average for each quadrant. I was then left with a $4 * 3 = 12$ dimensional feature vector for each image.

5. User Interface

My ultimate goal was to devise a CBIR algorithm for use on images obtained over the internet. Such as there are search engines for text and documents there should be for media. Consequently I chose to implement my algorithm entirely in the most internet friendly language: Java [7], with a web interface using JavaServlets. The implementation is available live at http://www.cs.mcgill.ca/~aringl/cbir.html

The user interface is quite straightforward and modelled closely after other Query by Multiple Image search engines [5]. The user is presented with N random images from the database. The user then places a check mark next to those images which are similar to the their goal image. These are denoted as the positive images. The images they don't choose are denoted as the negative images. The user then clicks continue and the algorithms above are applied. The user is then presented with N new images which are similar to the positive images they chose in the previous iteration. The user can now select additional images which are similar to their goal image. The process continues until the user has found the image they seek. At any time the user can deselect positive images which they feel are no longer representative of their goal image.

The user interface is written entirely in a JavaServlet [8] [6]. This is a server side technology similar to CGI that allows presentation of dynamic content on the web. For each user a session is maintained (usually in a cookie) that remembers where the user is in a search. For modularity the Servlet is kept completely self contained from the algorithm implementation. The Servlet communicates with the search algorithm and image database through java's RMI Registry.

6. Implementation

The algorithm implementation provided search service to the Servlet frontend by means of the Java RMI Registry. The Registry allows separate java processes to communicate with each other. Most importantly the Registry allows a java program to easily provide a service to other java programs.

Notice in my algorithm that in equation 2.0.1 (k-Nearest Neighbors) we have to iterate through every image feature vector in the database. If this database were on disk this would take a very very long time. Instead we want this database to stay memory resident. To insure this we load the database into main memory, and then register the program with the java RMI. The program will then wait forever continuously providing services for whenever the Servlet requests such. We will un-creatively call this service the back-end.

Upon the user's first visit the Servlet request N random images from the back-end. The Servlet then gives the user the images and waits. The user picks positive images and the Servlet asks the back-end to generate the weight matrix for these classified images. The back-end then exercises the algorithm from step (4). This is done without any special data structures such as SS-Trees and so takes fairly long. The Servlet then requests the k-Nearest Neighbors using the weight matrix obtained. The back-end complies and returns the results. The user is presented with the new images, and the process continues.

7. Results

The results were as expected for basic searches. In one such search I chose all of the presented images that had a large amount of blue in them. I was returned photos that had a large amount of blue. The same test worked for images with a lot of green. It seems the algorithm performs well and as expected for the given feature vector of Color Histogram.

The size of each region is entirely too large I found. Most of the images don't have large regions of the same color, and often the user wants to match some smaller sections of color. I believe that lots of smaller regions would be more appropriate for color matching. I store the entire spectrum of colors for each channel (RGB). In the future this could be discretized to save space.

I believe it is too difficult for humans to estimate the average color over a region. I believe some other color similarity measurement would provide better results. My algorithm would also perform better using larger dimension feature vectors. The larger the dimension of the feature vector, the more tunable the weight matrix W is. Hence the more accurate the resultant vector G.

8. Conclusion

My main goal was to test the usefulness of a weighted k-Nearest Neighbor classifier in Image Search and Retrieval. I believe the experiment was a success. Using a trivial feature vector I was able to quickly and accurately partition the image database using my algorithm.

I am excited about this algorithm and would like to further test its usefulness in Content-Based Image Retrieval. I believe a fast and simple algorithm such as this could be implemented on a large scale for internet search of images. I am

particularly interested in testing this algorithm on different image database and with different feature vectors.

References

1. Alberto del Bimbo, *Visual information retrieval*, Morkan Kauffman, San Francisco, California, 1999.
2. Joachim Hornesger Dietrich W.R. Paulus, *Applied pattern recogntion - a practical introduction to image and speech processing in c++*.
3. Vipin Kumar Eui-Hong (Sam) Han, George Karypis, *Text categorization using weight adjusted k-nearest neighbor classification*.
4. A. del Bimbo J. Assfalg and P. Pala, *Using multiple examples for content based retrieval*, Proc. Int'l Conf. Multimedia and Expo.
5. Markus Koskela Jorma Laaksonen and Erkki Oja, *Application of tree structered self-organizing maps in content-based image retrieval*, Artificial Neural Networks (1999).
6. Chuck Musciano and Bill Kennedy, *Html - the definitive guide*, O'Reilly, Sebastopol, California, 1997.
7. Herbert Schildt Patrick Naughton, *Java2 - the complete reference*, Tata McGraww-Hill, New Dhelhi, 1999.
8. Jason Hunter with William Crawford, *Java servlet programming*, O'Reilly, Sebastopol, California, 2001.

E-mail address: andrew.ringler@mail.mcgill.ca