Similarity and Invariance; Searching for Similar Images

36-350: Data Mining

September 6, 2006

READING: Section 14.5 in the textbook.

So far, we have seen how to search and categorize texts by representing them as feature vectors — bag-of-word vectors, appropriately scaled. All of the procedures we have used have depended *only* on the feature vectors, not on the underlying text. We can use the same techniques on any other kind of data, if only we define features for them, too, and represent them as feature vectors. (In terms of computer programming, using features rather than raw data is a kind of *abstraction*, and one of the advantages of abstraction is, exactly, facilitating the re-use of methods and procedures.) Whether recycling our techniques from text to some other domain will work well depends largely on how well we chose our features. As an example, let's look at images.

Computers represent images as matrices of pixels[1]; the matrix entries give the color of each pixel.

Colors Colors themselves are typically represented as three-dimensional vectors. Probably the most common color representation is *RGB*, where each color gets a red value, a green value and a blue value, between 0 and 1, indicating how the three primary colors should be blended[2]. Black is $(0, 0, 0)$ and white is $(1, 1, 1)$. The collection of all possible RGB colors is called the *RGB cube*. An alternative representation is the *HSV* scheme, where the *hue* gives the color type, represented as angle around a standard color wheel, *saturation* gives the "vibrancy" or "purity" of the color (low saturation = a lot of gray mixed in) and *value* is simply the brightness. The HSV *color space* is generally graphed as a cylinder or cone, and not a cube. These two schemes are actually equivalent to each other, and generally the most common for screen colors; there are however many others, and printed colors need a different color space, because they have a different physical basis.

[1]Actually, some graphics formats, like `jpeg`, use other representations, which are more compact for typical images. All formats get translated into pixel matrices when they're displayed, however, so we'll ignore this complication.

[2]There are three primary colors because normal human eyes contain three different sorts of color-sensitive cells, each tuned to a different band of light. Color-blind people effectively only have two primary colors; some other species of animals have four.

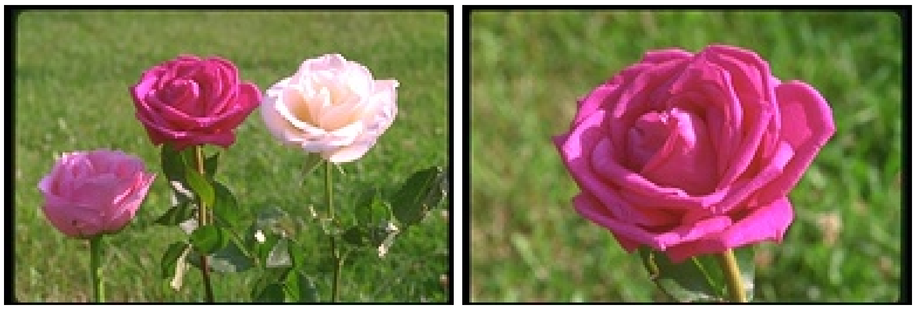


Figure 1: The images `flower1` (left) and `flower2` (right) from our data set. `flower2` is a close-up photo of the middle flower from `flower1`.

Many physically distinct colors, from different points in the RGB cube, are psychologically indistinguishable[3], or differ only trivially. Rather than use the full color space available to the computer, it is often desirable to *quantize* color down to a smaller number of values, effectively merging similar colors (so that we just have one "red", and not millions of shades of red). Geometrically, this means *partitioning* the RGB cube, cutting it into cells, and treating all colors in a cell as equivalent, calling them with a single name. (This corresponds to stemming for words, or, if the cells are very big and coarse, going beyond stemming to lumping together words with similar meanings, like "steel" and "iron".)

Back to Images Just as with documents, we would now like to do similarity searching for images. The user should be able to give the system an initial image and tell it "find me more which look like this".

The most straightforward approach would use the computer's representation of the image directly. If it's a color image of $M \times N$ pixels, we have a list of $3MN$ numbers; call this our "feature" vector, and calculate Euclidean distances as before. This can actually work OK for some purposes, like undoing the effects of some kinds of camera distortions, but it's not very good at finding meaningfully similar images. For instance, `flower2` is a close-up of the middle flower from `flower1`, but their Euclidean distance will be very large. (Why?)

Since the bag-of-words representation served us well with documents, we might try the same thing here; it's called the bag of colors. We first chose a quantization of the color space, and then, for each quantized color, count the number of pixels of that color in the image (possibly zero). That is, our procedure is as follows:

1. The user gives us an image Q. Convert it into a vector of color counts.

[3]On average, women tend to be better at color discrimination than men, and there is some evidence that this is due to hormonal differences, but the overlap is large, and most people can get better with practice.

2. For each image in the collection, measure the distance to Q.
3. Return the k images closest to Q.

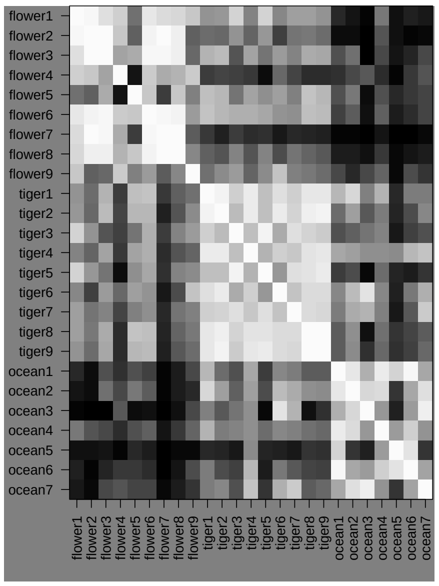
Today's data are photographs of flowers, photographs of tigers, and photographs of the ocean. Here is part of the color-count representation of the data.

	violetred4	goldenrod	lightskyblue4	gray58.2
flower1	59	0	0	0
flower2	128	0	0	0
flower3	166	75	0	0
tiger1	0	4	457	85
tiger2	0	0	0	2
tiger3	0	0	0	115
ocean1	0	0	4326	433
ocean2	0	0	2761	142
ocean3	0	0	1179	8596

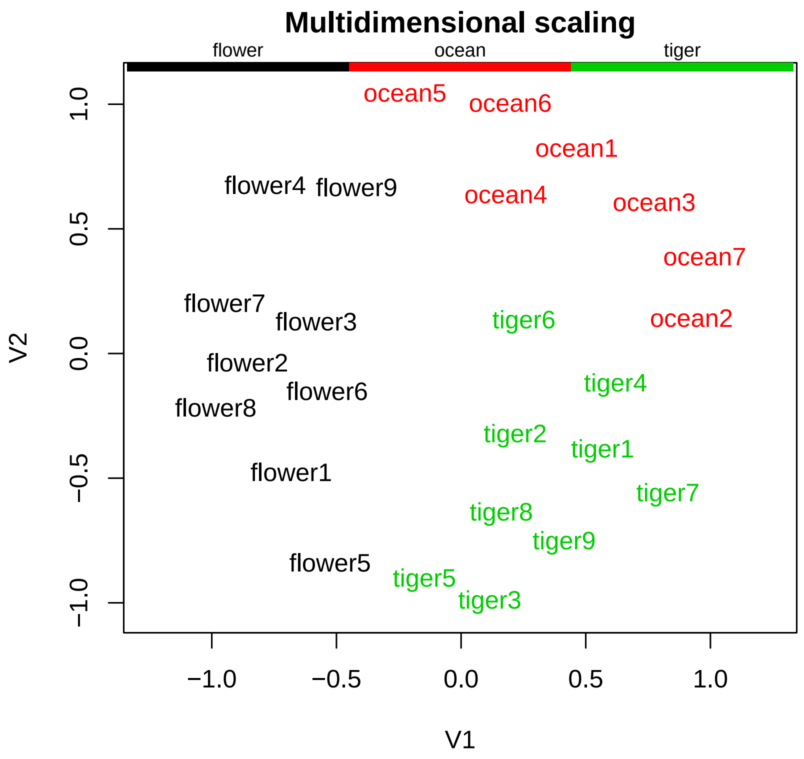
Just as with documents, we found it useful to emphasize rare words, it can be helpful to emphasize rare colors, by using the *inverse picture frequency*, IPF:

$$IPF(c) = \log\left(\frac{1}{PF(c)}\right)$$

Colors which occur frequently, like a black border, will have a small $IPF(c)$. (Notice that both `flower1` and `flower2` have such a border.)



Distance matrix
Normalized by Euclidean length
Lighter = Closer
No retrieval errors



Retrieval errors:

Normalization	Equal weight	IPF weight
None	8	4
Picture size	8	0
Euclidean length	7	0

As a sheer classification task, this example is a bit trivial. Our ancestors spent the last sixty-odd million years getting very good at sorting flowers from big cats at a glance. On the other hand, we do not have the same evolutionary priming for other kinds of image classification, like looking at a microscope slide and telling whether the cells on it are normal or cancerous...

Invariance

When trying to decide whether or not a certain representation is going to be useful, it's often helpful to think about the *invariances* of the representation. That is, what sorts of changes could we make to the object, without changing its representation? (The representation is *invariant under* those changes.) Invariances tell us what the representation ignores, what concrete aspects of the object it abstracts away from. Abstraction as such is neither good nor bad; particular abstractions are more or less useful for particular tasks.

The bag-of-words representation is invariant to changes in punctuation, word order, and general grammar. If it is normalized, it is also invariant to changes in document length. If inverse-document-frequency weighting is used, it is invariant under changes in the exact counts of the most common words ("the", "of", "and", etc.) which appear in any text written in English. Stemming, and grouping together synonyms, would add extra invariances. These invariances make the representations good at finding documents with similar topics, even though they might be written by different people. (By the same token, these invariances make it hard to, say, find all the stories by a given author.)

What invariances do we get for images in the bag-of-colors representation?

- Position of objects, pose, small changes in camera angle
- Small amounts of zooming
- Small amounts of unusual colors
- Differences in texture: we can scramble the pixels in any region, even the whole image, without effect. (Plaids and stripes seem the same.)

Some other invariances we might like, but don't get with color counts:

- Lighting and shadows; time of day
- Occlusion (one object covers another), placement in 3D
- Blurring, changes in focus

If there is an invariance we don't want, we can "break" that invariance by adding additional features. There are various schemes for representing textures, so that plaids, stripes, spots, "snow", solid colors, etc. can be distinguished, and many systems for content-based image searching combine color-counting with textures.

It's harder to *add* an invariance than to break one; generally you need to go back to the beginning and re-design your representation. The sorts of invariances we I just said we'd like to have, but don't, with this representation are all very hard to achieve; roughly 40% of the human cortex is devoted to visual processing for a very good reason.

You should think about what invariances the following representations have, and how they compare to the invariances of the bag-of-colors representation. The first one is much smaller than the bag of colors, and the second one is larger.

1. Represent an image by its average pixel color.
2. Represent an image by dividing it into four quarters, and doing color-counts in each quarter. (Is there any reason why four equal parts should be special?)

Practice

Content-based image retrieval systems are used commercially. All the ones I know of include the bag-of-colors representation, but generally also a number of other features. The one from Convera has a nice online demo, `http://vrw.convera.com:8015/cst`. You should play around with it and try to think about what representation it is using of the images.

Google's image search is very popular, but it's a total cheat, and doesn't analyze the images at all. Instead, it searches on the phrases you give it, takes the pages which get a high rank, and assumes that images which appear on those pages close you your query phrase are probably pictures of what you are looking for. If you ask it for images of "stellar collapse", it gives you lots of astrophysical diagrams and illustrations, pictures taken at astrophysics conferences, and the cover of an album titled *Stellar Collapse* by a band called Birchville Cat Motel.