Lab 5

Beeldverwerken

Patrick Spaans (11268476) & Bram Otten (10992456)

May 18, 2018

Principal Component Analysis

Image Correlation

Image correlation is a similarity measure. It denotes how similar two or more images are to each other. One reason for doing this is to match images in a database to input images. Image correlation is fairly naive in that the rotation and brightness and such of the image in the database must match that of the input image for the correlation (and thus, similarity,) to be high. But here's the formula (leaving out normalization) for images I_1 and I_2, both of size $M \times N$:

$$c(I_1, I_2) = \sum_{i=1}^{N} \sum_{j=1}^{M} I_1(i,j)\ I_2(i,j)$$
$$\text{(and } \forall_x \sum_{i=1}^{N} \sum_{j=1}^{M} (I_x(i,j))^2 = 1\text{)}.$$

This formula is related to the one for the sum of squared differences between pixels:

$$\begin{aligned} S(I_1, I_2) &= \sum_{i=1}^{N} \sum_{j=1}^{M} (I_1(i,j) - I_2(i,j))^2 \\ &= \sum_{i=1}^{N} \sum_{j=1}^{M} \Big[(I_1(i,j))^2 + (I_2(i,j))^2 - 2\ I_1(i,j)\ I_2(i,j) \Big] \\ &= \sum_{i=1}^{N} \sum_{j=1}^{M} \Big[(I_1(i,j))^2 + (I_2(i,j))^2 \Big] - 2\ c(I_1, I_2) \\ &= 2 - 2\ c(I_1, I_2) \text{ (so maximizing } c(...) \text{ minimizes } S(...)) \end{aligned}$$

in which that last weird looking step is okay because $\forall_x \sum_{i=1}^{N} \sum_{j=1}^{M} (I_x(i,j))^2 = 1$ is an assumption of the correlation measure above. So images have to be normalized such that their `sum(sum(...)) == 1`, which means dividing by the original amount that is equal to.

The double $\sum$ can be gotten rid of by concatenating all pixel values of I_i, row by row, into a (column) vector $\mathbf{x}_i$. This vector will have a length of $M \times N$, the amount of pixels.

$$\begin{aligned} c(I_1, I_2) = I_1(1,1)\ I_2(1,1) + I_1(1,2)\ I_2(1,2) + \cdots + I_1(1,N)\ I_2(1,N) + \cdots + \\ I_1(M,1)\ I_2(M,1) + \ldots I_1(M,N)\ I_2(M,N) \end{aligned}$$

and the new dot product will be:

$$\begin{aligned} \mathbf{x}_1(1)\mathbf{x}_2(1) + \mathbf{x}_1(2)\mathbf{x}_2(2) + \cdots + \mathbf{x}_1(N)\mathbf{x}_2(N) + \mathbf{x}_1(N+1)\mathbf{x}_2(N+1) + \cdots + \\ \mathbf{x}_1(MN-1)\mathbf{x}_2(MN-1) + \mathbf{x}_1(MN)\mathbf{x}_2(MN) \end{aligned}$$

which is the same because $\mathbf{x}_1(N) = I_1(1,N)$ et cetera, it's $\mathbf{x}_1^T\ \mathbf{x}_2$. This dot product is faster to calculate in MATLAB, but because of the size of images and the amount of different conditions there are under which the same object can occur in an image, this is still not fast enough.

Data Representation as Vectors and Matrices

Representing variations of objects (in images) in their eigenspace reduces the database size significantly. $\forall_{i \in n}\, \boldsymbol{x}_i$, there is an entry in $(NM) \times n$ sized database X with the mean of the object (the $\boldsymbol{x}$'s) subtracted. So every sample gets a column, every row denotes a pixel value (feature). (Shlens does the same as Trucco and us in his relevant paragraph IVC.)

$$X = \left[(\boldsymbol{x}_1 - \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{x}_j) \ \ldots \ (\boldsymbol{x}_n - \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{x}_j \right] = [(\boldsymbol{x}_1 - \bar{\boldsymbol{x}}) \ \ldots \ (\boldsymbol{x}_n - \bar{\boldsymbol{x}})]$$

The square and symmetric covariance matrix $Q = XX^T$ of this database can be 'eigendecomposed' to figure out basisvectors in which the variance can be expressed: the eigenvectors $\boldsymbol{e}_1 \ldots \boldsymbol{e}_n$. We can rank these eigenvectors on how much variance could be expressed using just those by comparing the corresponding eigenvalues (λ's). That best expression for an eigenvector i will be $\forall_{x \in n} g_{(x,i)} \boldsymbol{e}_i$. So g is something like the coordinate values for the eigenbasis axes.

Using just the k largest eigenvalues and -vectors, the database entries can then be approximated using much less computation for comparison with an input $\boldsymbol{x}$. The input will now be compared to $\forall_{i \in n}\, \boldsymbol{x}_i \approx \sum_{j=1}^{k} g_{(i,j)} \boldsymbol{e}_j$, and $n >> k$. The input itself can also be expressed in terms of those k vectors, creating an input $\boldsymbol{g}_i$. Euclidean/squared distance is the opposite of correlation here, and comparing with the whole database would be something like $\forall_{j \in n} \|\boldsymbol{g}_i - \boldsymbol{g}_j\|^2$. (Difference between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is effectively just the difference between $\boldsymbol{g}_i$ and $\boldsymbol{g}_j$, they're multiplied by the same stuff.)

So, eigenspace representation of images can be more compact, and computing their difference is cheap (also because that expression in the eigenspace has to be computed only once).

But that $Q = XX^T$ is pretty big ($(NM) \times (NM)$). $P = X^TX$ would only be $n \times n$ sized, and $n << (NM)$ (number of images in database << size of those images). For an eigenvector / -value of P:

$$\begin{aligned} P\boldsymbol{e} &= \lambda \boldsymbol{e} \\ X^TX\boldsymbol{e} &= \lambda \boldsymbol{e} \\ XX^TX\boldsymbol{e} &= X\lambda \boldsymbol{e} \\ Q(X\boldsymbol{e}) &= \lambda(X\boldsymbol{e}) \end{aligned}$$

meaning that $X\boldsymbol{e}$ is an eigenvector of Q with -value λ. In SVD $X = U\Sigma V^T$, columns of U are the eigenvectors of $XX^T = Q$, V's columns those of $X^TX = P$. Squaring Σ gives the corresponding eigenvalues. These are apparently properties of the SVD, though not immediately as obvious as Leo et al. suppose. So here's why this is the case:

$$\begin{aligned} X &= U\Sigma V^T \\ P = X^TX &= (U\Sigma V^T)^T U\Sigma V^T \\ &= (V\Sigma^T U^T) U\Sigma V^T \\ &= V\Sigma^T \Sigma V^T \end{aligned}$$

where $\Sigma^T\Sigma = \Lambda$. One mystery remains, whether $V^T = V^{-1}$. Because eigendecompositions are usually gives as $P = V\Lambda V^{-1}$, and while Shlens defines U and V to be orthogonal (appending the orthonormal vectors for the range between k and n), fixing the problem, eigenvectors don't *have* to be orthogonal. Okay, they actually do have to be because this is an SVD with a symmetric covariance matrix in the middle. And that's also why the eigenbases are orthogonal to each other.

Implementation and Comparison

See MATLAB files.

Positioning with Nearest Neighbour

Again, see MATLAB files. (Also for the few sort of theoretical questions.)