**CIELAB Analysis**

**Using**

**Principal Components**

**Analysis**

**Made By:-**

**Ankit Gupta 2K10/IT/016**

**Chirag Gupta 2K10/IT/023**

**Sanchit Gupta 2K10/IT/056**

**Vaseem Ahmed Khan 2K10/IT/068**

**Contents**

* Acknowledgment
* Introduction
  + PCA
  + Face Recognition
  + CIELAB
  + Euclidean Distance
* Working Methodology
  + Algorithm for finding Eigenfaces
  + Example
  + Algorithm for CIELAB analysis
  + Why is the threshold theta important?
* Conclusion
* Scope and Future Prospects
  + Neural Networks
  + Wine Colour Measurement
* References

**Acknowledgment**

**INTRODUCTION**

**PCA**

Principal component analysis (PCA) is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components .The number of principal components is less than or equal to the number of original variables.

This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it be orthogonal to (i.e., uncorrelated with) the preceding components. Principal components are guaranteed to be independent only if the data set is jointly normally distributed. PCA is sensitive to the relative scaling of the original variables. Depending on the field of application, it is also named the discrete Karhunen–Loève transform (KLT), the Hotelling transform or proper orthogonal decomposition (POD).

PCA was invented in 1901 by Karl Pearson. Now it is mostly used as a tool in exploratory data analysis and for making predictive models. PCA can be done by eigenvalue decomposition of a data covariance (or correlation) matrix or singular value decomposition of a data matrix, usually after mean centering (and normalizing or using Z-scores) the data matrix for each attribute. The results of a PCA are usually discussed in terms of component scores, sometimes called factor scores (the transformed variable values corresponding to a particular data point), and loadings (the weight by which each standardized original variable should be multiplied to get the component score).

**FACE RECOGNITION**

Facial recognition was the source of motivation behind the creation of eigenfaces. For this use, eigenfaces have advantages over other techniques available, such as the system's speed and efficiency. Using eigenfaces is very fast, and able to functionally operate on lots of faces in very little time. Unfortunately, this type of facial recognition does have a drawback to consider: trouble recognizing faces when they are viewed with different levels of light or angles. For the system to work well, the faces need to be seen from a frontal view under similar lighting. Face recognition using eigenfaces has been shown to be quite accurate. By experimenting with the system to test it under variations of certain conditions, the following correct recognitions were found: an average of 96% with light variation, 85% with orientation variation, and 64% with size variation.

To complement eigenfaces, another approach has been developed called [eigenfeatures](http://en.wikipedia.org/w/index.php?title=Eigenfeatures&action=edit&redlink=1" \o "Eigenfeatures (page does not exist)). This combines [facial metrics](http://en.wikipedia.org/w/index.php?title=Facial_metrics&action=edit&redlink=1) (measuring distance between facial features) with the eigenface approach. Another method, which is competing with the eigenface technique uses '[fisherfaces](http://en.wikipedia.org/wiki/Fisherface" \o "Fisherface)'. This method for facial recognition is less sensitive to variation in lighting and pose of the face than the method using eigenfaces.

A more modern alternative to eigenfaces and fisherfaces is the [active appearance model](http://en.wikipedia.org/wiki/Active_appearance_model), which decouples the face's shape from its texture: it does an eigenface decomposition of the face after [warping](http://en.wikipedia.org/wiki/Image_warping) it to mean shape. This allows it to perform better on different projections of the face, and when the face is tilted.

**CIELAB**

A ***Lab* color space** is a [color-opponent](http://en.wikipedia.org/wiki/Opponent_process" \o "Opponent process) space with dimension ***L*** for [lightness](http://en.wikipedia.org/wiki/Lightness_(color)) and ***a*** and ***b*** for the color-opponent dimensions, based on nonlinearly compressed [CIE XYZ color space](http://en.wikipedia.org/wiki/CIE_XYZ_color_space) coordinates.

The L\*a\*b\* color space is used e.g. in Adobe Photoshop when graphics for print have to be converted from RGB to CMYK, as the L\*a\*b\* gamut includes both the RGB and CMYK gamut. Also it is used as an interchange format between different devices as for its device independency.

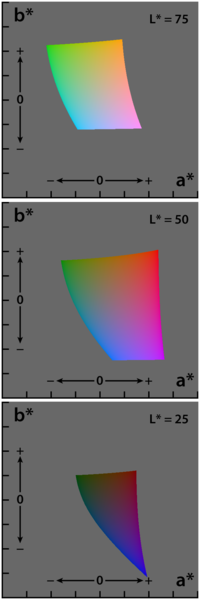
Unlike the [RGB](http://en.wikipedia.org/wiki/RGB_color_model) and [CMYK](http://en.wikipedia.org/wiki/CMYK_color_model) color models, *Lab* color is designed to approximate human vision. It aspires to perceptual uniformity, and its *L*component closely matches human perception of lightness, although it doesn't take the [Helmholtz–Kohlrausch effect](http://en.wikipedia.org/wiki/Helmholtz%E2%80%93Kohlrausch_effect) into account. It can thus be used to make accurate color balance corrections by modifying output [curves](http://en.wikipedia.org/wiki/Curve_(tonality)) in the *a* and *b* components, or to adjust the lightness contrast using the *L* component.

Because *Lab* space is much larger than the [gamut](http://en.wikipedia.org/wiki/Gamut) of computer displays, printers, or even human vision, a bitmap image represented as Lab requires more data per pixel to obtain the same precision as an RGB or CMYK bitmap.

Additionally, many of the "colors" within Lab space fall outside the gamut of human vision, and are therefore purely imaginary; these "colors" cannot be reproduced in the physical world.

The three coordinates of CIELAB represent the lightness of the color (***L\**** = 0 yields black and ***L\**** = 100 indicates diffuse white; specular white may be higher), its position between red/magenta and green (***a\****, negative values indicate green while positive values indicate magenta) and its position between yellow and blue (***b\****, negative values indicate blue and positive values indicate yellow).

As mentioned previously, the *L*\* coordinate ranges from 0 to 100. The possible range of *a*\* and *b*\* coordinates is independent of the color space that one is converting from, since the conversion below uses X and Y which come from RGB.



**EUCLIDEAN DISTANCE**

In mathematics, the Euclidean distance or Euclidean metric is the "ordinary" distance between two points that one would measure with a ruler, and is given by the Pythagorean formula.

The **Euclidean distance** between points **p** and **q** is the length of the [line segment](http://en.wikipedia.org/wiki/Line_segment) connecting them (\overline{\mathbf{p}\mathbf{q}}).

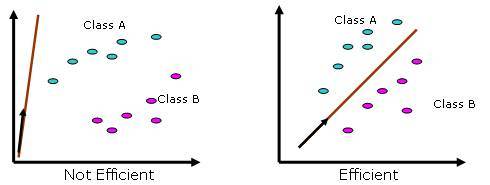
In [Cartesian coordinates](http://en.wikipedia.org/wiki/Cartesian_coordinates), if **p** = (*p*1, *p*2,..., *pn*) and **q** = (*q*1, *q*2,..., *qn*) are two points in [Euclidean *n*-space](http://en.wikipedia.org/wiki/Euclidean_space), then the distance from **p** to **q**, or from **q** to **p** is given by:

|  |
| --- |
| \mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2} = \sqrt{\sum_{i=1}^n (q_i-p_i)^2}. |

**WORKING METHODOLOGY**

The [Principal Components](http://en.wikipedia.org/wiki/Principal_components_analysis) (or Eigenvectors) basically seek directions in which it is more efficient to represent the data. This is particularly useful for reducing the computational effort. To understand this,  suppose we get 60 such directions, out of these about 40 might be insignificant and only 20 might represent the variation in data significantly, so for calculations it would work quite well to only use the 20 and leave out the rest.

This is illustrated by this figure:



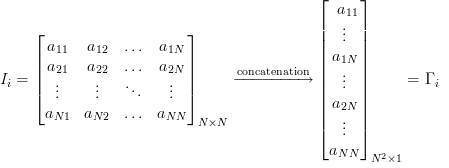
When we find the principal components or the Eigenvectors of the image set, each Eigenvector has some contribution from each face used in the training set. So the Eigenvectors also have a face like appearance. These look ghost like and are ghost images or Eigenfaces. Every image in the training set can be represented as a weighted linear combination of these basis faces.

**Algorithm for Finding Eigenfaces:**

**1.** Obtain M training images I_1, I_2 … I_M, it is very important that the images are centered.

[](http://onionesquereality.files.wordpress.com/2009/02/training-images.jpg)

**2.** Represent each image I_i as a vector \Gamma_i  as discussed above.



**3.** Find the average face vector \Psi.

\Psi = \displaystyle\frac{1}{M}\sum_{i=1}^M\Gamma_i

**4.** Subtract the mean face from each face vector \Gamma_i to get a set of vectors \Phi_i. The purpose of subtracting the mean image from each image vector is to be left with only the distinguishing features from each face and “removing” in a way information that is common.

\Phi_i = \Gamma_i - \Psi

**5.** Find the Covariance matrix C:

C = AA^T, where A=[\Phi_1, \Phi_2 \ldots \Phi_M]

Note that the Covariance matrix has simply been made by putting one modified image vector obtained in one column each.

Also note that C is a N^2 \times N^2 matrix and A is a N^2\times M matrix.

**6.** We now need to calculate the Eigenvectors u_i of C, However note that C is a N^2 \times N^2 matrix and it would return N^2 Eigenvectors each being N^2dimensional. For an image this number is HUGE.  The computations required would easily make your system run out of memory. How do we get around this problem?

**7.**Instead of the Matrix AA^T consider the matrix A^TA. Remember A is a N^2\times M matrix, thus A^TA is a M\times M matrix. If we find the Eigenvectors of this matrix, it would return M Eigenvectors, each of Dimension M \times 1, let’s call these Eigenvectors v_i.

Now from some properties of matrices, it follows that: u_i = Av_i. We have found out v_i earlier. This implies that using v_i we can calculate the M largest Eigenvectors of AA^T. Remember that M\ll N^2 as M is simply the number of training images.

**8.** Find the best M Eigenvectors of C=AA^T by using the relation discussed above. That is: u_i = Av_i. Also keep in mind that \begin{Vmatrix}u_i\end{Vmatrix}=1.

EXAMPLE

**Step 1: Get some data**

The data we have used is listed below or we can use matlab to transform an image into data sets containing different features.

**Step 2: Subtract the mean**

For PCA to work properly, you have to subtract the mean from each of the data dimensions.

The mean subtracted is the average across each dimension. So, all the values have been subtracted from them. This produces a data set whose mean is zero.



**Step 3: Calculate the covariance matrix**

Since the data is 2 dimensional, the covariance matrix will be:-



Since the non-diagonal elements in this covariance matrix are positive, we should expect that both the < and = variable increase together.

**Step 4: Calculate the eigenvectors and eigenvalues of the covariance matrix**

Here are the eigenvectors and eigenvalues:





It is important to notice that these eigenvectors are both *unit* eigenvectors i.e., their lengths are both 1. As expected from the covariance matrix, the two variables do indeed increase together.

**Step 5: Choosing components and forming a feature vector**

Here is where the notion of data compression and reduced dimensionality comes into it. It turns out that the eigenvector with the *highest* eigenvalue is the *principle component* of the data set.

In general, once eigenvectors are found from the covariance matrix, the next step is to order them by eigenvalue, highest to lowest. This gives you the components in order of significance. Now, if you like, you can decide to *ignore* the components of lesser significance. You do lose some information, but if the eigenvalues are small, you don’t lose much. If you leave out some components, the final data set will have less dimensions than the original. To be precise, if you originally have \_ dimensions in your data, and so you calculate \_ eigenvectors and eigenvalues, and then you choose only the first { eigenvectors, then the final data set has only { dimensions.

What needs to be done now is you need to form a *feature vector*, which is just a fancy name for a matrix of vectors. This is constructed by taking the eigenvectors that you want to keep from the list of eigenvectors, and forming a matrix with these eigenvectors in the columns.





**Step 5: Deriving the new data set**

Once we have chosen the components (eigenvectors) that we wish to keep in our data and formed a feature vector, we simply take the transpose of the vector and multiply it on the left of the original data set,

transposed.



where RowFeatureVector is the matrix with the eigenvectors in the columns *transposed* so that the eigenvectors are now in the rows, with the most significant eigenvector at the top, and is the mean-adjusted data *transposed*, ie. the data items are in each column, with each row holding a separate dimension. Final Data is the final data set, with data items in columns, and dimensions along rows.

What will this give us? It will give us the original data *solely in terms of the vectors we chose*. Our original data set had two axes, < and = , so our data was in terms ofthem. It is possible to express data in terms of any two axes that you like.

So what have we done here? Basically we have transformed our data so that it is expressed in terms of the patterns between them, where the patterns are the lines that most closely describe the relationships between the data. This is helpful because we have now classified our data point as a combination of the contributions from each of those lines

**Getting the old data back**

Remember that only if we took *all* the eigenvectors in our transformation will we get *exactly* the original data back. If we have reduced the number of eigenvectors in the final transformation, then the retrieved data has lost some information.

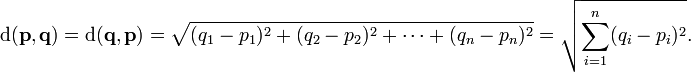






**Step 6: Calculate Euclidean Distance**

Now we calculate euclidean distance between two data sets using the formula:-



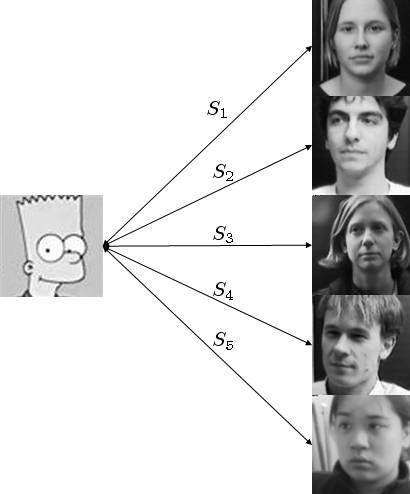
Lesser the value of Euclidean distance between two data sets, closer the data sets are to each other.

**Algorithm for CIELAB Analysis**

* Obtain several data sets Xi’s,Yi’s of several images and test image, containing any two features of CIELAB, say A,B respectively, using matlab’s inbuilt functions.
* Calculate eigenvalues and subsequently eigenvectors for all these data sets using PCA method.
* Using the eigenvector of the test image and those of the other images, taken one at a time, calculate the euclidean distance between them. The minimum Euclidean distance gives the closest features matching images.
* Repeat the above steps for other features of CIELAB, LA and LB.
* Decide the best feature among these by analysing the obtained minimum Euclidean distance and the actual similarity of images.

Why is the threshold, \Theta important?

Consider for simplicity we have ONLY 5 images in the training set. And a probe that is not in the training set comes up for the recognition task. The score for each of the 5 images will be found out with the incoming probe. And even if an image of the probe is not in the database, it will still say the probe is recognized as the training image with which its score is the lowest. Clearly this is an anomaly that we need to look at. It is for this purpose that we decide the threshold. The threshold \Theta is decided heuristically.

[](http://onionesquereality.files.wordpress.com/2009/02/non-face_score.jpg)

Consider a Simpson image as a non-face image, this image will be scored with each of the training images. Let’s say S_4 is the lowest score out of all. But the probe image is clearly not belonging to the database. To choose the threshold we choose a large set of random images (both face and non-face), we then calculate the scores for images of people in the database and also for this random set and set the threshold \Theta accordingly.

PCA is a common statistical technique for finding the patterns in high dimensional data’s.

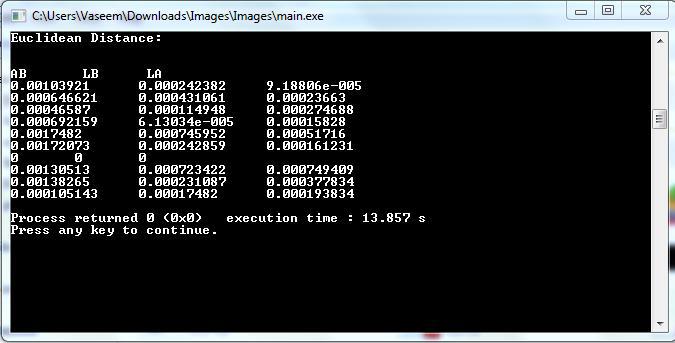
Feature extraction, also called Dimensionality Reduction, is done by PCA for a three main purposes like:-

i) To reduce dimension of the data to more tractable limits

ii) To capture salient class-specific features of the data,

iii) To eliminate redundancy.

**CONCLUSION**



* Ten facial images were taken and converted to data sets containing two features out of CIELAB, AB,LA & LB respectively.
* Then, PCA analysis was conducted on all data sets to generate their respective eigenvectors, which were used to calculate euclidean distance between test image and other images.
* Closest match was obtained in case of A,B features, indicated by the minimum euclid distance, thereby proving that A,B is the best 2-D feature combination present in CIELAB.

**SCOPE & FUTURE PROSPECTS**

**WINE COLOUR MEASUREMENT**

Colour is one of the principal parameters of the quality of wine and it is the first sensory property to be perceived in the glass by a consumer, winemaker or wineshow judge. In wineries, routine analysis of wine colour is generally performed using absorbance measurements by Somers or Glories methods, or by transmittance values (OIV method) at two to four wavelengths. However, the information provided is limited as the colour the eye perceives covers the full visible spectrum (380 nm ñ 770 nm). Using the whole visible spectrum, the Commission Internationale de líEclairage (CIE) has established a three dimensional colour space defined by L\*, a\* and b\* values, which represent lightness, red-green and yellow-blue characteristics, respectively.

With the CIELab parameters, the colour of wine can be measured precisely determined unambiguously, and differences to a colour standard can be quantified. Thus, CIELab colour measurements are used extensively by many industries and are gaining greater use as descriptors for colour by the wine industry.

**REFERENCES**

* [Face Recognition Using Eigenfaces](http://www.cs.ucsb.edu/~mturk/Papers/mturk-CVPR91.pdf), Matthew A. Turk and Alex P. Pentland, MIT Vision and Modeling Lab, CVPR ’91.
* [*Eigenfaces Versus Fischerfaces : Recognition using Class Specific Linear Projection*](http://cs.gmu.edu/~kosecka/cs803/pami97.pdf), Belhumeur, Hespanha, Kreigman, PAMI ’97.
* [*Eigenfaces for Recognition*](http://www.face-rec.org/algorithms/PCA/jcn.pdf), Matthew A. Turk and Alex P. Pentland, Journal of Cognitive Neuroscience ’91.
* B.K.Gunturk, A.U.Batur and Y.Altunbasak,(2003) “Eigenface-domain super resolution for face recognition,” IEEE Transactions of Image Processing