# Principal Component Analysis

Applying linear algebra to compress images

Krishn Ramesh, 20521942 Anshuman Patnaik, 20514402

SYDE 312 – Prof. Birkett

April 3rd, 2017

**Abstract**

*Principal Component Analysis (PCA) is a powerful statistical technique that applies key concepts from linear algebra including orthogonal transformations, eigendecomposition and projection in order to compress data. This paper details a simple implementation of PCA in Python to compress images.*

## I. Background

PCA is an old but powerful technique employed abundantly in any field that does data analysis, but is especially useful in statistics, data science and machine learning. It is primarily a means of dimensionality reduction for high dimensional datasets, but is also commonly used for data visualization, trend analysis, factor analysis, and noise removal. PCA is very popular because with minimal knowledge of statistics and linear algebra, it provides a way to reduce a complex dataset into a simpler one where it reveals hidden patterns and relationships. It is closely related to other linear algebra methods such as Singular Value Decomposition (SVD) and Linear Discriminant Analysis (LDA).

## II. Application of Linear Algebra

PCA draws upon several linear algebra concepts, primarily eigenvalue decomposition and change of basis. A primary goal of PCA is to first determine the most meaningful basis for a given set of data, and then project the data onto this basis to filter out the noise in a dataset. Mathematically, the goal of PCA is to determine a matrix **P** to transform the original dataset **X** into a more meaningful representation **Y**. The transformation matrix **P** is assumed to be composed of orthonormal basis vectors which are linear combinations of the original basis vectors.

$$PX = Y \tag{1}$$

To determine the most meaningful basis and the associated transformation matrix, the covariance of the dataset must be analyzed. A square covariance matrix $S_x$ is constructed that describes the relationships between all pairs of features in the dataset, as shown below.

$$S_x = \frac{1}{n-1} XX^T \tag{2}$$

Larger covariances imply that the pair of measurements share a high degree of redundancy, while smaller covariances imply the points have lower redundancy. PCA looks to choose the set of basis that best minimizes the total covariance between points by favouring lower co-varying points and disfavouring high-redundancy points. Hence, the optimal covariance matrix is a diagonal matrix which implies that no feature in the given dataset is redundant. The required diagonal matrix solution in combination with the orthonormality assumption lends the problem to an eigenvalue decomposition solution method. Following a similar procedure as SVD (a generalization of PCA), it can be concluded that the rows of **P** are the eigenvectors of $XX^T$. Shlens [1] shows a thorough derivation of the solution, while this paper focuses on implementation.

## III. Image Compression

To apply PCA to a real-world problem, image compression was considered. Images are widely used in machine learning to build recognition systems, but they are often unwieldy to work with because of how large image data can get. Even a small 100 x 100 pixel image contains 10,000 data points. PCA is a good algorithm that captures the most important image data while getting rid of the redundant data, essentially performing lossy compression.

The image compression implemented for this paper was inspired by PCA tutorials [2]. The full source code can be found on Github [3], but the most important function (which performs PCA) is displayed below.

```
def principal_components(A, n = 50):
 # A is a 2D matrix representing
 # grayscale pixel values of an image

 # normalizing the matrix by subtracting
 # the mean (along columns)
 A_norm = (A-np.mean(A.T,axis=1)).T

 # computing eigenvalues and
 # eigenvectors of covariance matrix
 cov = np.cov(A_norm)
 e_vals,e_vecs = np.linalg.eig(cov)

 # sorting the eigenvalues and
 # eigenvectors in descending order
 idx = np.argsort(-e_vals)
 e_vals = e_vals[idx]
 e_vecs = e_vecs[:,idx]

 # taking only the top n components
 e_vals = e_vals[:n]
 e_vecs = e_vecs[:,range(n)].real

 # projection of the data in the
 # new space
 proj = np.dot(e_vecs.T,A_norm)

 return e_vecs,proj,e_vals
```

**Listing 1:** *PCA in Python*

Python is widely used in data science, as it has a variety of useful libraries such as `scikit-learn`, `numpy`, `pandas` and `matplotlib` which make working with large datasets easier. The function in Listing 1 is well-commented and goes through the process outlined in section 2. It takes a matrix `A` as input, which is a 2D matrix representing an image with grayscale values for all its pixels. Next, it normalizes the input image and finds its covariance i.e. the relationship between every pixel. The covariance matrix contains all the information about the image; it is broken down with an eigendecomposition. The eigenvalues are then sorted in descending order and limited to the top `n`, as inputted by the user (or defaults to 50). The largest eigenvalues capture the largest amount of variance or information. The original data is finally projected onto the basis eigenvectors (a.k.a. principal components) that correspond to the `n` largest eigenvalues to produce a compressed representation of the image.

It is important to note that the result of PCA does not represent anything visually, and is merely a compressed and lossy representation of the original data. To reconstruct the original image, the following projection is used:

```
# Reconstructing original image
mean = np.mean(img,axis=0)
img_r = np.dot(e_vecs,proj).T + mean
```

**Listing 2:** *Reconstructing the original image*

Listing 2 projects the PCA representation of the image back onto the principal components and adds the mean to reconstruct the original image. To test out this code, this PCA implementation was run on a sample image.

## IV. Results

The sample image was 427 x 640 pixels, meaning the maximum number of components was 640. To test PCA, the image was inputted into the PCA function while varying the number of components from 0 to 350 in step sizes of 50.


**Figure 1:** *Reconstructions of varying number of components.*

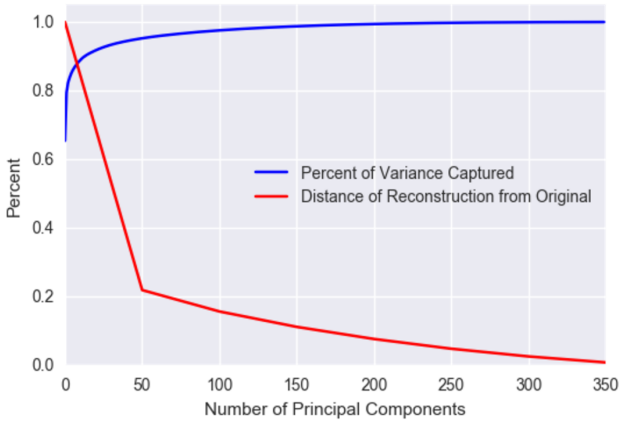


**Figure 2:** *Distance error and variance captured while varying the number of components.*

The resulting reconstructions are seen in Figure 1, compared to the original at the bottom right.

After about 150 components, it becomes very difficult to visually distinguish between the reconstruction and the original. To analyse the error, a simple Euclidean distance metric was computed. The percentage of variance of the original image captured by the number of components used was also calculated. Both of these results are summarized in Figure 2. If all of the original 640 components were used, the reconstructed image would be exactly the same as the original. For practical purposes, however, only about 150 components are required to store the image almost indistinguishably, leading to a 76.5% reduction in size!

## V. Conclusion

PCA is a powerful and widely-used dimensionality reduction algorithm. This paper used PCA to express an image using only 150 of its original 640 components while still maintaining most of its informational integrity. While useful in most contexts, PCA does have some limitations. It is non-parametric and so cannot take advantage of prior probabilistic information; it cannot deal with non-linear relationships between features; it fails when largest variance does not lead to a meaningful set of axes e.g. in non-Gaussian distributed data [1]. Nevertheless, if none of the above limitations apply, PCA harnesses the power of linear algebra to effectively reduce the computational strain of big data analysis.

## References

[1] Jon Shlens. *A Tutorial on Principal Component Analysis*. `https://www.cs.princeton.edu/picasso/mats/PCA-Tutorial-Intuition_jp.pdf`

[2] *PCA and image compression with numpy* `http://glowingpython.blogspot.ca/2011/07/pca-and-image-compression-with-numpy.html`

[3] *Source Code for image compression application* `https://github.com/krishnr/PCA/blob/master/PCA.ipynb`