Principal Component Analysis with Applications to Image Compression

Math 22A: Vector Calculus and Linear Algebra I

Andrew Sima

1 Cover Letter

Dear Dusty and the Math 22a Community,

This paper explores the mathematics behind principal component analysis, and provides examples of how this statistical tool is used in image compression.

Work for the project was done independently over the last few weeks of the course. The rigorous mathematical treatment for PCA was based on Chapter 7 of Lay and introductory academic papers listed in the works cited section. All research into statistics was done through the STAT110 textbook. The applications to image compression were researched through academic papers and R examples found in these papers were replicated in Python.

I would also like to thank peer reviewers and mention the ways in which I have incorporated their comments. Specifically, I would like to thank Thomas Kaminsky for catching many grammatical errors, and for pointing out a confusion in the way that I was trying to obtain a new basis from the eigenvectors via PCA in Section 4.3.1. This section has since been updated to specify more explicitly what the desired basis is. Thanks to Erin Yuan as well pointing out a typo in the treatment of matrix transposes in Section 4.3.4. This was a mistake in the way that I was notating the matrices originally, and has since been corrected.

I would also like to thank my staff reviewer Mark Kong for his thorough feedback on the mathematical treatment done in the paper. Mark pointed out a critical point that was not originally elaborated upon, specifically that the "dirtiness" in the original dataset is not removed until the projections onto the less important principal components are removed. The big picture has been updated to include the explanation of the projection onto the new basis of principal components, and the Python example explains how the projection is applied in greater depth.

Best,

Andrew Sima

2 Introduction

In any given experiment in which scientists gather a dataset, it is without doubt that in many cases the data is not "clean" (in the sense that there is intrinsic properties of the experimental design that make it difficult to glean any clear results from the data). Principal Component Analysis (PCA) is often chosen as a tool to address this issue, with its primary purpose being to reformat variables (in terms of a different basis) within the dataset in order to draw stronger conclusions about the most relevant factors of an experiment.

Similar to scientific experiments, PCA is becoming increasingly relevant in the world of technology, as computer scientists explore more efficient methods for storing data. One of these areas is that of image compression, in which a PCA algorithm is used to retain the most important pixels for a given image, while reducing the number of variables needed for storing the image. This paper will cover the technique of PCA along with the mathematics underlying it, include examples of how the technique is specifically applied to image compression, and address broader applications of the technique in the growing world of technology.

3 Central Questions

The following will be used as a high level roadmap to the paper:

1. **Scope and Rigor:** What is Principal Component Analysis (PCA)? Why is it useful? How does PCA fit in relation to other similar computational techniques? How does PCA fit into the current scope of research?
2. **Application to Image Compression:** How does PCA play a role in image compression? Are there concrete examples that demonstrate the effectiveness of this technique?
3. **Relevance:** Why use PCA? What are some of the benefits of using this technique? What are some concrete applications of the technique? How does PCA fit into broader use cases in a variety of industries?

4 Scope and Rigor

4.1 What is Principal Component Analysis (PCA)?

Formally, Principal Component Analysis (PCA) is a method used to compute the most important measurement types (i.e. the principal components) of a given dataset, and then selecting a subset of these computed components to change the basis of the data, effectively reducing/ranking the number of variables considered.

PCA is primarily used to abstract the main aspects of a given dataset and generate/model predictions from the data (often visual models); the technique falls into a broader category of techniques known as dimensionality reduction, which is an area of computation that deals with simplifying data that is collected in higher dimensions (imagine a vector in $\mathbb{R}^n$) into lower dimensions (a new

vector in $\mathbb{R}^m$ satisfying $m < n$) while preserving key characteristics of the data. This is often used when data in higher dimensions is sparse (an overwhelming number of elements in the matrix representing the data are 0) because of the shear number of variables measured.

Furthermore, PCA is closely related to areas of machine learning and statistics. In particular, PCA relies heavily on definitions from statistics such as variance (explored later in the paper), and is the basis of many machine learning models and algorithms (PCA appears in K-means clustering and lives in the subset of unsupervised learning). As Principal Component Analysis is much easier to conceptualize in relation to the dataset, we will return to the analogy of variables within an experiment for the rigorous treatment.

4.2 Brief Detour into Statistics:

To understand the following rigorous treatment, we will need to cover some basic concepts in statistics. We will quickly define random variables and expected value as follows:

Definition 1: Random Variables, Probability, and Expected Value

1. **Random Variable and Probability:** Given a set of possible outcomes A and a set B consisting of numerical values $b \in \mathbb{R}$, a random variable X is defined as a function mapping $A \to B$. The probability of a given outcome $a \in A$, denoted by $P(X = a)$ is the corresponding mapped element in B denoted by $X(a)$.

2. **Expected Value:** The expected value can be thought of as a weighted average in some sense. Given a random variable X, the expected value denoted as $E[X]$ is formally defined as follows:
$$E[X] = \sum_a a \cdot P(X = a).$$

Building on these two definitions, one of the most fundamental concepts used will be variance. As a formal statement, given a random variable X, and the mean of this random variable denoted by $\mu = E[X]$, the variance is the expected value of the square of the distance from mean. In a sense, the variance measures the spacing between elements of a given dataset. We can further define the covariance between two random variables as a measure of variability between the two.

Definition 2: Variance and Covariance

1. **Variance:** Given a random variable X and its mean defined as $\mu = E[X]$, we define the variance denoted by $\text{Var}(X)$ to be
$$\text{Var}(X) = E[(X - \mu)^2].$$
The variance is usually denoted by σ_X^2, and for the rest of the paper we will use this notation.

2. **Covariance:** Given two random variables A and B, along with means of the two variables denoted by $\mu_A = E[A]$ and $\mu_B = E[B]$, the covariance $\text{Cov}(A, B)$ is defined to be
$$\text{Cov}(A, B) = E[(A - \mu_A) \cdot (B - \mu_B)].$$
Likewise, the notation we will be using is σ_{AB}^2.

With these definitions in mind, we return to a rigorous mathematical treatment of PCA.

4.3 Rigorous Treatment:

4.3.1 Big Picture:

Suppose we have an experiment in which there are m measurement types or experimental variables that we are interested in. Here, m is known as the dimension of the initial dataset. The goal of PCA is to choose a new orthogonal basis (in a sense a new set of coordinates instead of the given measurement types) that will reduce the extraneous data and allow for cleaner analysis.

Consider a given data point consisting of m measurement entries. We can write this data point as a vector $\mathbf{x} \in \mathbb{R}^m$ as follows:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}.$$

So given n data points, we can express the dataset as an m-by-n matrix $\mathbf{X}$, where each of the columns $\boldsymbol{x_i} \in \mathbb{R}^m$ represents the m measurements of a given data point i. We can then express the entire dataset as:

$$\boldsymbol{X} = [\ \boldsymbol{x_1}\ \boldsymbol{x_2}\ \cdots\ \boldsymbol{x_n}\ ].$$

Now our goal is to transform the dataset $\boldsymbol{X}$ into a dataset $\boldsymbol{Y}$ through the selection of a more meaningful basis from the principal components. We will choose this new basis to arbitrarily be, say $\mathcal{B} = \{\boldsymbol{b_1}, \boldsymbol{b_2}, \ldots, \boldsymbol{b_n}\}$. Post transformation, we would like each column of $\boldsymbol{Y}$ to correspond to the original column of $\boldsymbol{X}$.

Now to construct the transformation matrix (we will denote this as $\boldsymbol{P}$), we instead take the columns of the matrix to be the basis vectors of $\boldsymbol{Y}$ as follows (this is essentially a change of basis matrix):

$$\boldsymbol{P} = [\ \boldsymbol{b_1}\ \boldsymbol{b_2}\ \cdots\ \boldsymbol{b_n}\ ].$$

If we specifically restrict the transformation matrix $\boldsymbol{P}$ to be an orthonormal matrix (i.e. that the basis $\mathcal{B}$ is orthonormal), then we know that $\boldsymbol{P}^{-1} = \boldsymbol{P}^T$ by the properties of an orthonormal matrix. Then to obtain our new dataset, we take a transformation matrix from $\boldsymbol{X}$ to $\boldsymbol{Y}$, given by $\boldsymbol{Y} = \boldsymbol{P}^T\boldsymbol{X}$ (we take the inverse of the transformation $\boldsymbol{P}$ from $\boldsymbol{Y}$ to $\boldsymbol{X}$), which when expanded with matrix multiplication looks like the following:

$$\begin{aligned} \boldsymbol{Y} &= \boldsymbol{P}^T\boldsymbol{X} \\ &= \begin{bmatrix} \boldsymbol{b_1}^T \\ \boldsymbol{b_2}^T \\ \vdots \\ \boldsymbol{b_n}^T \end{bmatrix} [\ \boldsymbol{x_1}\ \boldsymbol{x_2}\ \cdots\ \boldsymbol{x_n}\ ] \\ &= \begin{bmatrix} \boldsymbol{b_1} \cdot \boldsymbol{x_1} & \cdots & \boldsymbol{b_1} \cdot \boldsymbol{x_n} \\ \vdots & \ddots & \vdots \\ \boldsymbol{b_n} \cdot \boldsymbol{x_1} & \cdots & \boldsymbol{b_n} \cdot \boldsymbol{x_n} \end{bmatrix} \end{aligned}$$

As can be seen, each entry is a dot product of an original data point $\boldsymbol{x_i}$ and a chosen basis vector $\boldsymbol{b_j}$. Note further that if $|\boldsymbol{b_j}| = 1$, then $\boldsymbol{b_j} \cdot \boldsymbol{b_j} = 1$ by definition. This yields that

$$|\text{proj}_{\boldsymbol{b_j}} \boldsymbol{x_i}| = \left| \frac{\boldsymbol{x_i} \cdot \boldsymbol{b_j}}{\boldsymbol{b_j} \cdot \boldsymbol{b_j}} \boldsymbol{b_j} \right| = |(\boldsymbol{x_i} \cdot \boldsymbol{b_j})\boldsymbol{b_j}| = |\boldsymbol{x_i} \cdot \boldsymbol{b_j}| \,.$$

So therefore, if each of the basis vectors have magnitude 1, then the entries of the matrix above represent the magnitude of the projections of each respective data point onto the respective chosen basis vector.

This procedure described above can be varied by choosing the basis $\mathcal{B}$ to have smaller dimension (this corresponds to having a smaller number of principal components). Then we can project the original dataset $\boldsymbol{X}$ onto $\mathcal{B}$ to obtain a different dataset. Note that the projection

$$\text{proj}_W \boldsymbol{y} = UU^T \boldsymbol{y},$$

where $y \in \mathbb{R}^n$, W is a subspace of $\mathbb{R}^n$, and U is an orthonormal matrix obtained from the basis of W. Therefore, projecting $\boldsymbol{X}$ onto the basis $\mathcal{B}$ amounts to multiplying the transformation matrix $\boldsymbol{P}$ by the transformed matrix $\boldsymbol{Y}$ defined above. When executed with the correct basis, this yields a lense into the data (based on the chosen principal components and discarding the rest) with the original "dirtiness" removed.

4.3.2 Noise and Redundancy:

Now given a big picture framework, we would like to specify exactly what we would like to optimize for (i.e. exactly what we would like to project onto). To do this, we first identify two sources of "dirtiness" of the experimental data that we would like to account for, namely noise and redundancy.

Noise is often times a result from imprecisions in the measurement process. More simply, noise can be thought of as the quality of the data. Regardless of the basis chosen for PCA, or more in general the method of data analysis, if the quality of the data is not above a certain threshold, then the method will not reveal meaningful results. Thus, we must assume that our measurements are sufficiently clean (i.e. low noise) in order to use PCA.

In contrast to this is redundancy, which is when our method of data collection records various occurences multiple times. This causes data to be recorded (possibly) in multiple different measurement types. In this case, PCA is very effective, because we can choose a new coordinate system conveniently using a single basis vector for representing these various measurement types. With this established, we now turn to methods for finding the best choice for basis $\mathcal{B} = \{\boldsymbol{b_1}, \boldsymbol{b_2}, \ldots, \boldsymbol{b_n}\}$.

4.3.3 The Covariance Matrix:

To find the ideal basis, we intuitively need to find a way to compare the different measurement types. This can be done by analyzing the covariance of data points given.

Suppose we have two specific measurement types, namely A and B, with row vectors $\boldsymbol{x_a}$ and $\boldsymbol{x_b}$ representing the values of these measurements at each of the n data points as follows ($x_{c,i}$ corresponds to the measurement of type c at the i-th data point):

$$\boldsymbol{x_a} = [\ x_{a,1}\ x_{a,2}\ \cdots\ x_{a,n}\ ] \text{ and } \boldsymbol{x_b} = [\ x_{b,1}\ x_{b,2}\ \cdots\ x_{b,n}\ ].$$

Now we convert normalize each of the measurements above to make it easier to calculate the covariance. In particular, we can adjust each of the entries by subtracting off the mean. Given that each $a_i = x_{a,i} - E[\boldsymbol{x_a}]$ and $b_i = x_{b,i} - E[\boldsymbol{x_b}]$, we write the following:

$$\boldsymbol{a} = [\ a_1\ a_2\ \cdots\ a_n\ ],$$
$$\boldsymbol{b} = [\ b_1\ b_2\ \cdots\ b_n\ ].$$

With each of these, we formalize an expression for computing the covariance between measurement type A and B.

Definition 3 (Covariance of Measurement Types): The covariance of two measurement types A and B (where each element of $\boldsymbol{a}$ and $\boldsymbol{b}$ is normalized relative to its mean) is given by

$$\sigma_{AB}^2 = \frac{1}{n-1}\boldsymbol{a}\boldsymbol{b}^T.$$

Note from the definition that

$$\sigma_{AB}^2 = E[(A - \mu_A) \cdot (B - \mu_B)].$$

Taking here that

$$\ell_i = a_i \cdot b_i = (x_{a,i} - E[\boldsymbol{x_a}]) \cdot (x_{b,i} - E[\boldsymbol{x_b}]),$$

we can write that

$$\sigma_{AB}^2 = \sum_{\ell_i} \ell_i \cdot P\left((A - \mu_A) \cdot (B - \mu_B) = \ell_i\right).$$

From the dataset at hand being n data points (each occuring with equal probability), we expect the factor inside the summation to be roughly $\frac{1}{n}$. In fact, the standard probability for an unbiased estimator is $\frac{1}{n-1}$, according to the literature (will not prove here, see proper normalization for an unbiased estimator). We can factor out this probability to get

$$\sigma_{AB}^2 = \frac{1}{n-1}\sum_{\ell_i} \ell_i = \frac{1}{n-1}\sum_i a_i \cdot b_i = \frac{1}{n-1}\boldsymbol{a}\boldsymbol{b}^T.$$

With this result, we can further extend this to define a covariance matrix that will measure the covariance with respect to all possible pairs of variables. To address the issue of redundancy, we will ideally want entries other than the diagonals of this matrix to be as small as possible.

Definition 4 (Covariance Matrix): Given measurement types $\boldsymbol{x_1}, \ldots \boldsymbol{x_n}$, we perform a normalization relative to the mean of each measurement type to create a matrix $\boldsymbol{X}$ defined as follows,

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x_1'} \\ \boldsymbol{x_2'} \\ \vdots \\ \boldsymbol{x_n'} \end{bmatrix}$$

where each entry x_i' of $\boldsymbol{x_i'}$ and corresponding entry x_i of $\boldsymbol{x_i}$ satisfy

$$x_i' = x_i - E[\boldsymbol{x_i}].$$

Then we define the covariance matrix $\boldsymbol{S_X}$ as follows:

$$\begin{aligned}
\boldsymbol{S_X} &= \frac{1}{n-1}\boldsymbol{X}\boldsymbol{X}^T \\
&= \frac{1}{n-1}\begin{bmatrix} \boldsymbol{x_1'} \\ \boldsymbol{x_2'} \\ \vdots \\ \boldsymbol{x_n'} \end{bmatrix}\begin{bmatrix} \boldsymbol{x_1'}^T & \boldsymbol{x_2'}^T & \cdots & \boldsymbol{x_n'}^T \end{bmatrix} \\
&= \frac{1}{n-1}\begin{bmatrix} \boldsymbol{x_1'}\cdot\boldsymbol{x_1'}^T & \cdots & \boldsymbol{x_1'}\cdot\boldsymbol{x_n'}^T \\ \vdots & \ddots & \vdots \\ \boldsymbol{x_n'}\cdot\boldsymbol{x_1'}^T & \cdots & \boldsymbol{x_n'}\cdot\boldsymbol{x_n'}^T \end{bmatrix} \\
&= \begin{bmatrix} \sigma^2_{\boldsymbol{x_1}\boldsymbol{x_1}} & \cdots & \sigma^2_{\boldsymbol{x_1}\boldsymbol{x_n}} \\ \vdots & \ddots & \vdots \\ \sigma^2_{\boldsymbol{x_n}\boldsymbol{x_1}} & \cdots & \sigma^2_{\boldsymbol{x_n}\boldsymbol{x_n}} \end{bmatrix}.
\end{aligned}$$

Note that the entries on the diagonals of $\boldsymbol{S_X}$ are simply the variances for each of the measurement types (covariance with itself is variance). In order to actually use this to simplying the relations between measurement types, we would like our final dataset after the transformation, namely $\boldsymbol{Y}$, to have each of the non-diagonal entries be 0. This way, we only need to analyze the diagonals themselves (and in effect the variances of the new coordinates).

Furthermore, we would ideally like to choose an orthonormal basis to construct the transformation matrix. This will satisfy the projection condition and ensure that the variances are in a sense "perpendicular" to one another in the new coordinates. It further allows us to discern in which direction the variances are the most important. To do this computationally, we use diagonalization to solve for the desired matrix.

4.3.4 PCA via Diagonalization:

Ideally, after applying PCA, we will be able to identify the dataset in a new basis of principal components, or the directions in the m-dimensional space of our measurement types with the largest variance. Intuitively, these are the most "important" because they produce the most variability (which is why we want to focus in on the particular variable). We will apply the following algorithm:

Theorem (PCA): Given some dataset $\boldsymbol{X}$ as described previously, we can find a matrix $\boldsymbol{P}$ with rows forming an orthonormal basis that transforms the given dataset to $\boldsymbol{Y} = \boldsymbol{P}\boldsymbol{X}$ such that the covariance matrix defined as

$$\boldsymbol{S_Y} = \frac{1}{n-1}\boldsymbol{Y}\boldsymbol{Y}^T$$

is diagonal.

To prove this, we would need to prove that the covariance matrix as defined in the statement is diagonal. We can start by plugging in the definition of $\boldsymbol{Y}$ as follows (we use $(AB)^T = B^T A^T$ to expand the transpose):

$$\begin{aligned}\boldsymbol{S_Y} &= \frac{1}{n-1}\boldsymbol{Y}\boldsymbol{Y}^T \\ &= \frac{1}{n-1}\left(\boldsymbol{P}\boldsymbol{X}\right)\left(\boldsymbol{P}\boldsymbol{X}\right)^T \\ &= \frac{1}{n-1}\boldsymbol{P}(\boldsymbol{X}\boldsymbol{X}^T)\boldsymbol{P}^T\end{aligned}$$

Notice that the format $\boldsymbol{P}$ and $\boldsymbol{P}^T$ already has a very similar format to the diagonalization of a matrix with n distinct eigenvalues. Furthermore, note that the middle term given by $\boldsymbol{X}\boldsymbol{X}^T$ is a symmetric square matrix (note that it is m-by-m) by properties of the transpose and dot product (any entry satisfies $a_{i,j} = \boldsymbol{x}_i \cdot \boldsymbol{x}_j^T = \boldsymbol{x}_j^T \cdot \boldsymbol{x}_i = \boldsymbol{x}_i^T \cdot \boldsymbol{x}_j = a_{j,i}$). Therefore, if we can show that any symmetric square matrix is diagonalizable, then we move closer to proving the desired.

Conveniently, this is exactly the result given by the Spectral Theorem for Symmetric Matrices (Lay, Section 7.1). In fact, the Spectral Theorem further states that this symmetric matrix $\boldsymbol{A}$ is orthogonally diagonalizable. In other words, there exists a decomposition such that

$$\boldsymbol{A} = \boldsymbol{B}\boldsymbol{D}\boldsymbol{B}^{-1} = \boldsymbol{B}\boldsymbol{D}\boldsymbol{B}^T,$$

with the matrix $\boldsymbol{B}$ being an orthonormal matrix, and the matrix $\boldsymbol{D}$ the diagonal matrix with its eigenvalues as the diagonal entries.

Using this, we can choose the matrix $\boldsymbol{P}$ to be exactly the transpose of the matrix from the spectral decomposition $\boldsymbol{B}^T$ (note that since the columns of $\boldsymbol{B}$ form an orthonormal basis, then the rows of $\boldsymbol{P}$ also form the same orthonormal basis). Then we can rewrite the matrix $\boldsymbol{X}\boldsymbol{X}^T = \boldsymbol{B}\boldsymbol{D}\boldsymbol{B}^T$ and plug this back into the equation.

$$\begin{aligned}\boldsymbol{S_Y} &= \frac{1}{n-1}\boldsymbol{B}^T(\boldsymbol{B}\boldsymbol{D}\boldsymbol{B}^T)(\boldsymbol{B}^T)^T \\ &= \frac{1}{n-1}\boldsymbol{B}^{-1}(\boldsymbol{B}\boldsymbol{D}\boldsymbol{B}^{-1})\boldsymbol{B} \\ &= \frac{1}{n-1}\boldsymbol{D}.\end{aligned}$$

This successfully shows that given a matrix with rows forming an orthonormal set, the covariance matrix is diagonal. This gives us the a newly constructed basis as well as the isolated variances with respect to that basis as desired.

Furthermore, by definition if we have the diagonal covariance matrix $\boldsymbol{S_Y}$, then the variances in the i-th diagonal entries correspond to the variance for the dataset $\boldsymbol{Y}$ with respect to the i-th vector in the orthonomal basis described by the rows of $\boldsymbol{P}$. This leads us to the final definition.

Definition 5 (Principal Components): From the orthogonally diagonalizable matrix $\boldsymbol{X}\boldsymbol{X}^T = \boldsymbol{B}\boldsymbol{D}\boldsymbol{B}^T$ in the above, we obtain a matrix $\boldsymbol{D}$ of eigenvalues along the diagonal, along with a matrix of eigenvectors $\boldsymbol{B}$. These eigenvectors forming the rows of $\boldsymbol{P} = \boldsymbol{B}^T$ are known as the principal components of the dataset.

4.4 Limitations and Further Study:

Note that because we constructed the matrix $\boldsymbol{P}$ based solely on linear combinations of the basis that we chose, we essentially assume that our data must fit a linear model (i.e. no variables carry powers greater than 1).

There are various PCA variants such as non-linear PCA (having to do with Neural Networks) that consider the algorithm outside of these constraints. There are also linear variants such as Independent Component Analysis that deals with non-Gaussian distributions (also not defined here) to ensure statistical independence of two measurement types, as well as Sparse Principal Component Analysis in which the model purposely ignores certain measurement types, while building the model on the rest.

Although the mathematical background shown above thoroughly covers PCA, it doesn't cover the more general version known as Singular Value Decomposition (SVD). Essentially, SVD is a theorem that says that any matrix can be rewritten as the product of an orthogonal matrix, a diagonal matrix, and another orthogonal matrix (similar format to the diagonalization theorem). SVD is even more rigorously tied to linear algebra, and produces many nice implications for the row and column space of the given dataset $\boldsymbol{X}$.

5 Application to Image Compression

Given how increasingly valuable computer storage is in today's modern data boom, we often desire to compress certain file formats to occupy less space, yet preserve the most identifiable features of the file. This is particularly the case with image files, which are essentially represented by a matrix of pixels, each of which is essentially three bytes (RGB), with each byte (integer from 0 to 255) specifying the "intensity" of Red, Green, and Blue.

There are a variety of methods used to compress images. In some cases, we would like to identify and compress the image based on color. In this case, we typically take each of the three RGB bytes from the file and form three vectors (essentially three data points), one for each respective color. Then each pixel of the file will be a measurement type, which we will reduce to a smaller number while preserving the integrity of the photo.

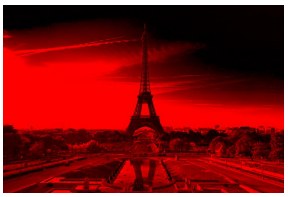
In other cases, we have many images that share particular similarities (for example, pictures of the same person) and we would like to compress these images into a fewer number of images that can be linearly combined to produce the original images. In these cases, the entire image is usually treated as a 1D array with every byte being a "measurement type", and using PCA to reduce this massive number of inputs to a smaller more manageable basis.

To emphasize again, the representations of number of input are only typical. As the way that PCA is applied varies heavily on use case, a particular image compression algorithm may use PCA differently, or altogether a different technique. In essence, the general idea is to identify a bottleneck in the way that the image is currently using storage, then apply PCA to reduce the quantity required of this bottleneck, effectively reducing the amount of storage required.

5.1 Example Compression in Python

For your reference, the python code for the following example is found **here**. Given the following image, we would like to reduce the size of the image (currently 71KB) while retaining its main properties. We will first extract the RGB colors for each pixel and separate them into 3 color matrices (one for each color). Below are these respective matrices:

Figure 1: Initial Image and Color Matrices



(a) initial red matrix

(b) initial green matrix

(c) initial blue matrix

On each of the specific matrices, we will perform PCA to extract the desired number of components, then rebuild the matrices using these components. We then overlay the three transformed matrices to build the colored image (this is done using the `np.dstack` function). In this specific example, we will treat the rows of the image as each of the data points and the columns of the image as the each of the measurement types.

For each color matrix listed above, we first compute the covariance matrix as defined in Section 4 above [1] using the `covariance_matrix` function. Then using the `np.linalg.eigh` package, we can compute the corresponding eigenvalues and eigenvectors for this symmetric covariance matrix we have defined. We sort the eigenvalues from greatest to least, taking the desired number of principal components along with the corresponding eigenvectors (we do this by first storing the respective indexes). The principal components are obtained from the `slice_eigen` function.

Finally, the `build_color_matrix` function handles the projection of the original dataset onto the newly chosen basis from the principal components. The `revectors` variable is the transformation matrix $\boldsymbol{P}$, and the transformed data is computed by taking the projection as $\boldsymbol{PP}^T\boldsymbol{X}$.

[1] In the code, we take the transpose of the actual input matrix. This is because in definition 4, the dataset $\boldsymbol{X}$ has the measurement types along the rows, but here we have them along the columns.

Listed below are the results for a chosen number components, along with corresponding sizes.

(a) 1 component (b) 2 components (c) 3 components (d) 5 components (e) 10 components

(f) 30 components (g) 50 components (h) 100 components (i) 150 components (j) 200 components

(k) 300 components (l) 350 components (m) 400 components (n) 450 components (o) 500 components

Figure 2: Images with Variable Number of Components

5.2 Discussion of Example Compression in Python

We also graph the size of the image relative to the number of components. We can quickly loop over all components to find that the smallest image size corresponds to 1 principal component at 23.1KB while the largest image size corresponds to 177 principal components at 85.7KB.

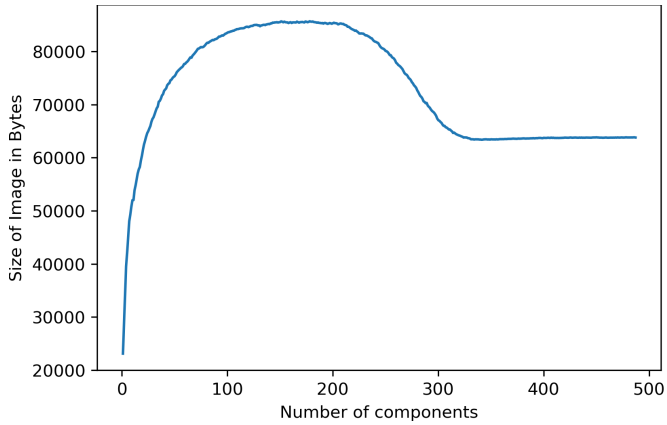


Figure 3: Size of Image vs. Number of Components

Observing the graph, we see that the size of the image levels out at around 300 principal components, with the final size hovering around 63KB. This corresponds to approximately 11% reduction in the size of the image.

As demonstrated, the above application efficiently applies a PCA algorithm to reduce the image size while preserving the original coloring qualities. The choice of measurement type and data points for the image matrices was relatively simple and yielded the desired functionality. Some of the shortcomings of the example above are that it may not be intuitively clear why the columns are chosen to be the measurement types and rows the data points. Here, this choice was made because it was relatively simple to visualize and fit nicely with the definitions presented in the paper.

Theoretically, there may be other methods to split the image into measurement types and data points that may have been more efficient. One such alternative method could've been treating the data points as the colors themselves and each pixel in the image grid as its own measurement type. There are many other algorithms in the literature that build upon this and can be used to further compress the image.

6 Relevance and Conclusion

As inferred from the discussions already illustrated above, the use cases and applications of PCA are extremely versatile. Ultimately, the technique is one of many in a data analyst's toolbox that is applied in various industries. Listed here are a few of these industries (outside of those already mentioned) along with a few of the ways in which PCA is applied.

Bioinformatics: Given a cluster of cell types with various measurements capturing expression over a pool of genes, we would like to discover which cell types are most prevalent for expressing particular genes. PCA is used to select the particular gene of interest and isolate expression levels for that gene, along with a variety of other dimension reduction techniques such as t-SNE (t-distributed stochastic neighbor embedding) and UMAP (uniform manifold approximation and projection) to visualize data in a 2D plot.

Neuroscience: Suppose we would like to measure the effect of a certain stimulus of the action potentials of neurons (this ties in nicely with neural networks). However, the recorded spikes may be recorded multiple times over the course of the experiment by various metrics and/or sensors, and we must also determine which exact neuron fired which spike (i.e. specify each metric/sensor as a measurement type and each neuron as a data point). PCA and various clustering algorithms, are used together to determine the above, as well as model the shape of the action potential.

Quantitative Finance: The number of strategies that PCA could be used for in this field is vast but here are a couple of examples: constructing algorithms for the allocation of assets in a given stock portfolio, assessing and analyzing market risk, generating predictions for market returns, and constructing interest rate models.

In summary, given a dataset that we wish to analyze, we often need to find ways to clean the data for simpler analysis. By computing the covariance matrix and extracting the principal components, we can project the original dataset onto the new basis constructed from the chosen principal components. Assuming the effectiveness of the principal components, this transformed data will reveal hidden trends within the data and remove "dirtiness" that existed in the original dataset. Whenever a dataset has too many variables to analyze, PCA is a viable technique towards simplifying the analysis at hand.

7 Works Cited

1. Schlens, John. “A Tutorial on Principal Component Analysis.” 25 Mar. 2003.
2. Arab, Abbas Harbi, Jamila Abbas, Amel. (2018). Image Compression Using Principle Component Analysis. Al-Mustansiriyah Journal of Science. 29. 141.
3. Alpaydin, Ethem. Introduction to Machine Learning (OIP). Mit Press, 2004.
4. Blitzstein, Joseph K., and Jessica Hwang. Introduction to Probability. CRC Press, 2019.
5. Lay, David C., et al. Linear Algebra and Its Applications, Fifth Edition. Pearson, 2015.
6. Mofarreh-Bonab, Mohammad Mofarreh-Bonab, Mostafa. ”Color Image Compression using PCA.” International Journal of Computer Applications. 111. 5. Feb. 2015.
7. Chen, Huanting. ”Portfolio Construction Using Principle Component Analysis.” Worcester Polytechnic Institute. Aug. 2014.
8. Schlegel, Aaron. “Principal Component Analysis with R Example.” Aaron Schlegel’s Notebook of Interesting Things, 19 Jan. 2017.