Principal Component Analysis on Multiband Landsat Imagery

##### AUTHORS

*Muhammad Omer Bin Atique   
2020335 ES 304 B, BCS, Batch 30   
GIKI*

*Syed Ammar Bin Farrukh   
2020468 ES 304 B, BCS, Batch 30   
GIKI*

*Shahryar Mubashar  
2020447 ES 304 B, BCS, Batch 30   
GIKI*

*Sameer Arif Khan   
2020430 ES 304 B, BCS, Batch 30   
GIKI*

*Abstract*—We present the application of Principal Component Analysis (PCA) on 12 bands of Landsat-8 imagery. Each of these bands further has its own bands of RGB (Red Green Blue). The basic idea is to apply PCA and then merge the 12 bands into a single image. PCA basically reduces the dimensions of whatever image or dataset we use it on. It is a machine learning technique which can reduce the no of features for faster processing while keeping enough detail to give optimal results.

A single band of the Landsat dataset is of dimensions (7841, 7701, 3). Applying PCA on such a huge image takes a very long time and requires huge amounts of RAM which most every day student computers do not have. So before applying PCA we reduce the resolution of the image to a much more processable amount. This gives us faster and better results. We can easily then reduce the dimensions of any required image.

Keywords—PCA, Principal Component Analysis, dimensionality, dimensions, resolution, bands, Landsat-8, Landsat, images, features.

# Introduction

## Curse of Dimensionality

PCA is a very powerful machine learning algo which helps with the issue known as the curse of dimensionality. Dimensionality in a dataset becomes a severe setback to achieving a reasonable efficiency for most algorithms. Having more features does not always improve the accuracy. When data does not have enough features, the model is likely to underfit(performs well on training data but poorly on test data) and having more features may cause the model to overfit (performs well on test data but poorly on training data). Hence it is called the curse of dimensionality. The curse of dimensionality is an astonishing paradox for data scientists,

based on the exploding amount of n-dimensional spaces — as the number of dimensions, n, increases.

## Dimensionality Reduction

Dimensionality reduction eliminates a couple of features of the dataset and creates a set of features that contain enough information to predict the target variables more efficiently and accurately. That is where PCA comes in. PCA seeks such a projection of the data which preserves as much information as possible.

##### METHODOLOGY

# i. Principal Component Analysis

## Theory

Principal Component Analysis or PCA is a technique to reduce the dimension of a given set of matrices. PCA decomposes the data matrix into principal components which is a set of linearly uncorrelated variables. All principal components are orthogonal to each other.

Typically, PCA is used for dimensionality reduction. It is a great solution for Curse of dimensionality. PCA is applied through 2 major methods:

* PCA through Eigenvalue Decomposition.
* PCA through Singular Value Decomposition.

Here we will only discuss PCA through Eigenvalue Decomposition. The SVD method will be discussed in section III.

## PCA through Eigenvalue decomposition

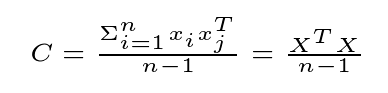
## Usually, EVD method entails a sample covariance matrix C. Assume that we have a Matrix X. Covariance matrix C can be found out by the equations 2.1.

We can also apply the eigenvalue decomposition using eq 2.2.

Here Q is an orthogonal matrix whose columns are the eigenvalues of C. ^ is a diagonal matrix with eigenvalues in the decreasing order on the diagonal.

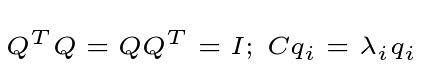
We can find the principal components by multiplying each eigenvalue in ^ with the corresponding eigen vector in Q. It can also be a matrix multiplication of ^ and Q.

## Equations



*Equation 2.1*



*Equation 2.2*

*Equation 2.3*

# SVD

## Theory

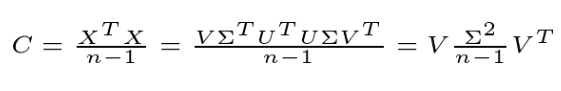
In section II, we have demonstrated how PCA is applied through the covariance matrix and eigenvectors. Another way which is also widely used is the SVD method, which another decomposition method. It decomposes the desired matrix into the dot product of two matrixes U, Vt and a diagonal matrix of singular values denoted by **Σ**.

Consider a matrix X. It can be decomposed further into: U, V, **Σ** such as in eq 3.1. Both U and V are orthogonal and **Σ** is diagonal. Consider a covariance matrix C. We get a form of C very similar to eq 2 in PCA section.

When compared with above eq 2, we know that columns of V are the principal directions. The principal components can be obtained with XV = U**Σ.**

## Equations

*Equation 3.1*



*Equation 3.2*

XV = U**Σ.**

*Equation 3.3*

## Link with PCA

PCA and SVD are directly related approaches and both can be used to decompose any rectangular matrices.

# Landsat-8 Images (DataSet)

## Landsat-8

Landsat-8 is an American Earth observation satellite launched on February 11, 2013. It is the seventh Landsat spacecraft to successfully enter orbit, making it the eighth overall. NASA and the United States Geological Survey collaborate on what was once known as the Landsat Data Continuity Mission (LDCM). The USGS developed the ground systems and manages ongoing mission operations, while NASA Goddard Space Flight Center in Greenbelt, Maryland, contributed launch vehicle development, mission systems engineering, and acquisition. It consists of the Thermal Infrared Sensor (TIRS), which is used to research global warming as well as the Earth's surface temperature, and the camera of the Operational Land Imager (OLI). Orbital Sciences Corporation, the mission's primary contractor, produced the satellite. Ball Aerospace & Technologies and NASA's Goddard Space Flight Center (GSFC) built the spacecraft's instruments,and United Launch Alliance was hired to launch it (ULA). NASA checked and verified LDCM during its first 108 days in orbit, and on May 30, 2013, when LDCM was formally called Landsat 8, USGS operations took over from NASA.

For our dataset we will use a loction extracted using Earth Explore facilities. The location selected will give an image, which is a combination of 12 separate bands. Each of the bands is furthur devided into 3 RGB bands. So in total we have around 36 bands. Our goal will be to first split each band into its RGB bands and apply PCA on each of the RGB bands and merge them together to again make the band. We will do this for each of the 12 bands, meaning that PCA will be applied 36 times in total.

## Preprocessing

In this section, we will discuss the minimal amount of pre-processing that was used to prepare each of the bands for PCA application.

1. Resize:

Initially when we import our image into the python file and convert it into an array, the dimensions are (7841, 7701, 3). If we were to apply PCA on these dimensions, the processing time would increase exponentially. To avoid this we will resize or rather reduce the resolution of the image to (1000, 1000, 3). This will increase the execution time by a lot. Let it be noted that the 3 in these dimensions means RGB.

1. Splitting:

As it was mentioned before, we have to apply PCA on the individual RGB bands of each of the 12 bands. For this we have to split each RGB into separate bands. Each RGB band will then be (1000, 1000) and result in 3 of these arrays. Then each of these arrays will have PCA applied separately.

## Figures

Over here will show all 12 of the bands that are used as our dataset.

*Figure 1* NOTE: All of the following pictures are of a fixed location having the coordinates (Long:074 43’ 37’’ E) (Lat: 32 53’ 53’’ N) of the city of Akhnur in Indian Occupied Kashmir

# Python Code Implementation

## Included Libraries

First let’s mention the included libraries.

import numpy as np

from sklearn.utils.extmath import svd\_flip

import matplotlib.pyplot as plt

import cv2

import numpy.linalg as la

**Numpy:** is used for mathematical calculations and array operations.

**Sklearn.utils.extmath:** Helped in the implementation of SVD.

**Matplotlib.pyplot:** Helps visualize images in python.

**Cv2:** helps working with images.

**Numpy.linalg:** linalg stands for linear algebra. Helps with calculations involving linear algebra.

## Preprocessing

* *Resize:*

im1 = cv2.resize(image, (1000, 1000),interpolation = cv2.INTER\_LINEAR)

The explanation is given in Section IV.B.I

* *CvtColor:*

img = cv2.cvtColor(im1, cv2.COLOR\_BGR2RGB)

Helps arranging the color bands of an image. Separates RGB bands. Here we arrange them as Blue Green and Red respectively.

## SVD & PCA: Now we discuss the implementation of PCA function using SVD. In code we made a separate function which would return the processed image.

def pcaSVD(X, k):

* *Splitting:*

blue, green, red = cv2.split(X)

The explanation is given in Section IV.B. II. The Function used is a cv2 function called split. This will effectively split the color bands of RGB and store them in separate variable for separate processing.

* *Mean Calculation:*

b\_m = np.mean(blue)

g\_m = np.mean(green)

r\_m = np.mean(red)

We calculate mean as it is required in the calculation of SVD. We use the mean function in the numpy library.

* *SVD:*

The SVD code is basically an imitation of the SVD function in Sklearn library. While most of the code directly corelates with the theory mentioned before, there are still a couple of things which are rather ambiguous. That being said one can still get a good idea of what is happening. We first standardize the RGB arrays by dividing each by 225.

blue = blue/255

    green = green/255

    red = red/255

Initially we subtract the mean calculated in the previous step with the arrays that we got by splitting.

b\_mean = blue - b\_m

    g\_mean = green - g\_m

    r\_mean = red - r\_m

Then we get U, V, **Σ** using the SVD function from linear algebra library.

After which we get the required no of components that we want to preserve. Which is K. The user can manipulate this value accordingly. We used 10.

U\_b, s\_b, Vt\_b = la.svd(b\_mean, full\_matrices = False)

    components\_b = Vt\_b[0:k]

Then svd\_flip function will be used for sign correction on U and V.

U\_b, Vt\_b = svd\_flip(U\_b, Vt\_b)

Then **Σ** is diagonalized. When we first initialize **Σ**, we get the eigenvalues in a row matrix. For further calculations we need a scalar matrix with eigenvalues on its main diagonal.

Sigma\_b = np.diag(s\_b)

Then we basically apply a dot product between K no of components of U and Σto get the principal components. This corresponds to equation 3.3.

 pcab = np.dot(U\_b[:, 0:k], Sigma\_b[0:k, 0:k])

After applying the above operation, we get a matrix with dimension (1000, k) but we have to restore the image to its original (1000, 1000) dimensions, while maintaining the k number of features. For that we use the following line of code.

pca\_b = np.dot(pcab, components\_b) + b\_m

* *Visualizing RGB bands:*

Here we will visualize the RGB bands of each band after the application of PCA:

fig = plt.figure(figsize = (15, 7.2))

fig.add\_subplot(131)

plt.title("PCAb Image")

plt.imshow(pca\_b)

fig.add\_subplot(132)

plt.title("PCAg Image")

plt.imshow(pca\_g)

fig.add\_subplot(133)

plt.title("PCAr Image")

plt.imshow(pca\_r)

plt.show()

The result:

*Figure 2-RGB bands of first band after PCA application*

* *Merging RGB bands:*

This formula will effectively merge the 3 RGB bands and return us the reduced image.

img\_reduced =(pca\_b/3)+(pca\_g/3)+(pca\_r/3)

return img\_reduced

## Merging Bands:

## Here we will finally merge all 12 bands after the application of PCA to get the final Image.

## Merging:

## We will have a loop running for all 12 images and will apply PCA on each of the 12 images and then use the formula below to merge them into one final image.

merImg = merImg + (img\_reduced[i])/no\_img

* *Original and Reduced:*

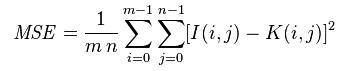
*Figure 3-Original first band vs the Reduced first band after PCA application*

## Final Image

The final Image is obtained after each of the 12 bands has been reduced and all the reduced bands are also merged into one single image.

*Figure 4-Final reduced image obtained after the combination of all 12 reduced bands*

# Error Analysis

 For error analysis we have used Mean Squared Error, the equation of which is given below:

The code for the calculation of MSE is given below:

 err = np.sum((img.astype("float")-img\_reduced[i].astype("float"))\*\*2)

    err /= float(img.shape[0] \* img.shape[1])

##### RESULTS

As evident from the final image displayed in Section 4.E, the Image has been compressed and some details have definitely been removed. Hence, we can say that PCA has been applied successfully. The amount of detailed that one wants to retain can be edited easily by changing the value of K.

##### TASK DISTRIBUTION

*2020335—*Python coding, researching PCA and SVD, report writing.

*2020468*—Python coding, researching PCA and required python libraries, report writing.

*2020430*---Report writing, Landsat research, dataset downloading and configuration, researching PCA.

*2020447*---Report writing, researching Mean Squared Error and its implementation, researching PCA and SVD.

##### References

`

1. https://en.wikipedia.org/wiki/Earth\_observation\_satellite
2. https://en.wikipedia.org/wiki/Remote\_sensing
3. https://www.un-spider.org/links-and-resources/data-sources/daotm-landcover
4. https://en.wikipedia.org/wiki/Multispec–tral\_imaging
5. https://en.wikipedia.org/wiki/Landsat\_program
6. https://landsat.gsfc.nasa.gov/
7. https://gisgeography.com/usgs-earth-explorer-download-free-landsat-imagery/
8. https://en.wikipedia.org/wiki/Principal\_component\_analysis
9. <https://www.ieee.org/conferences/publishing/templates.html>
10. <https://towardsdatascience.com/svd-in-machine-learning-pca-f25cf9b837ae>
11. <https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8>
12. <https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa>
13. <https://towardsdatascience.com/dimensionality-reduction-of-a-color-photo-splitting-into-rgb-channels-using-pca-algorithm-in-python-ba01580a1118>
14. Steven L. Brunto, J.Nathan Kutz “Data Driven Science & Engineering Machine Learning, Dynamical Systems and Control” University of Washington, 2017.
15. <https://www.youtube.com/watch?v=gXbThCXjZFM&&ab_channel=SteveBrunton>
16. <https://www.youtube.com/watch?v=nbBvuuNVfco&&t=283s&&ab_channel=SteveBrunton>
17. <https://youtu.be/xy3QyyhiuY4?list=PLMrJAkhIeNNSVjnsviglFoY2nXildDCcv>
18. <https://youtu.be/WmDnaoY2Ivs?list=PLMrJAkhIeNNSVjnsviglFoY2nXildDCcv>
19. https://youtu.be/QQ8vxj-9OfQ?list=PLMrJAkhIeNNSVjnsviglFoY2nXildDCcv