Principal Component Analysis

Jap Purohit :1940109

Nihar Patel :1940119

Mohit Prajapati :1940171

Raj Gariwala :1940118

Purvam Sheth :1940151

Introduction:

Abstract

In this project we are trying to implement PCA i.e. Principal Component Analysis using Householder reflection. And see whether there is any abnormalities from the graph or not.

Background

Principal Component Analysis or PCA is a dimensionality reduction technique. There are many dimensionality reduction techniques available and for each them exist different approaches. But PCA limits low-dimensional reproduction blunder, however another reasonable target is to amplify the disperse of the projection, under the reasoning that doing so would yield the most useful projection. There are different techniques too for linear dimensionality reduction. We have included Multidimensional scaling as it is associated with PCA. This task centres around acquiring the central parts of a grid. The Principal Component Analysis helps discover the part of the lattice that contribute most essentially. It is utilized for pressure of information without losing any huge data. This task

utilizes numerous ideas of direct variable-based math subsequently expanding our grip and comprehension of the ideas. It likewise encourages us in perceiving how these ideas are utilized in the reality and how it transforms ourselves in manners we can't envision. Our gathering has attempted to diminish the elements of a satellite perception set hence zeroing in on the parts that truly matter and, in any event, consolidating various segments to shape one solitary variable.

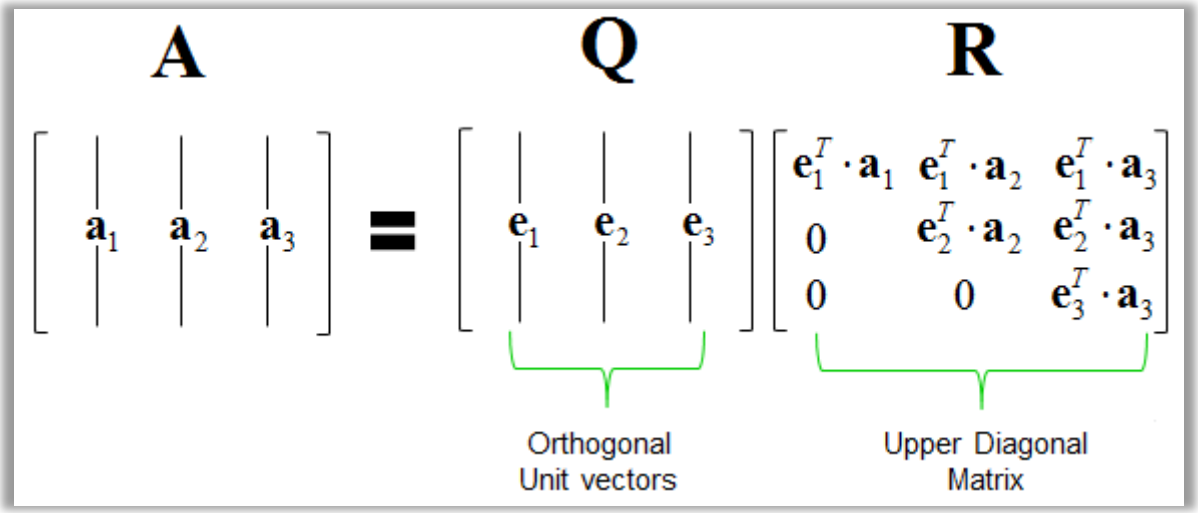
Motivation and Overview

In the era of modern technology where data is produced in the enormous amount it is important to analyse this data and produce some useful result which can be used in further innovations. For this project we decide to work on a dataset of a satellite observation and column representing the pixels. So observing large dataset it is impossible to draw inferences and observations from the same. So in order to draw inference we need to simply the data and reduce the variables which affects the least. This is where Principal Component Analysis comes in. It helps in reducing the dimensionality of the data set while keeping in mind that there is no significant loss of information. The relation between different factors is based on a form of matrix. It analyses the data and tells us what can be clubbed together or in a crucial time even rejected for then.

Our dataset contains 5100 x 36 data points to work on. With each row representing a different image and each column representing a different pixel of dataset.

Keywords

- PCA (Principal Component Analysis) and data reduction
- Householder Reflection and QR Decomposition of matrix



Approach

Description and the Approach used

- **QR Decomposition of matrix**
  To compute with matrixes QR Decomposition is a stable method in comparison with other methods. Matrix **A**, in system of equation in matrix form **Ax=b** is converted to **A=QR** and which is further written as $\mathbf{Rx=Q^{-1}b}$, which is much easier to compute.
  In QR Decomposition, **Q** is the orthonormal matrix which means $\mathbf{Q^{-1}=Q^{T}}$, whereas **R** is the upper diagonal matrix. So, we can also rewrite the equation as $\mathbf{Rx=Q^{T}b.}$

  $\mathbf{Rx=Q^{T}b}$ is easier to solve because right hand side is just a vector. And R is an upper triangular. So, this can be easily solved by back-substitution.

- **Householder Reflection**
  Householder reflection is one of the important aspects in finding QR decomposition.

Householder reflection technique is used in finding reflection of any vector in any dimension which respect to any plane in the given subspace. $\mathbf{H = (\mathit{I} - 2(vv^T))}$, this equation is known as householder reflection equation, where **v** is the orthonormal vector to the plane and **H** is the Householder reflector matrix.

$$\begin{aligned} x' &= Hx \\ &= x - 2(xv)v \\ &= x - 2v(xv) \\ &= \mathit{I}x - 2v(v^Tx) \\ &= \mathit{I}x - 2(vv^T)x \\ &= (\mathit{I} - 2(vv^T))x \end{aligned}$$

$$H = (I - 2(vv^T))$$

Given an n * m invertible matrix A:

Q an orthogonal matrix n * m

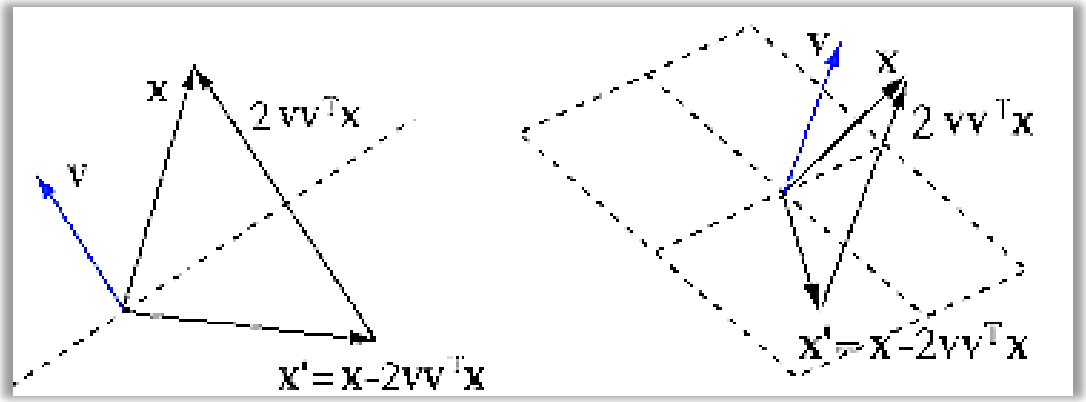
R is an upper triangular m * m matrix

Such that QR = A

$$A_{(n*m)} = Q_{(n*m)} * R_{(m*m)}$$

$$Q_{(n*m)} = (H_{(n-1)} * H_{(n-2)} * H_{(n-3)} * \ldots * H_{(2)} * H_{(1)})^T$$

$$R_{(m*m)} = Q^T * A$$



Representation of vector and it's reflection with respect to a plane.

Simulation

Description:

1. **About the dataset:**
   The satellite dataset comprises of features extracted from satellite observations. In particular, each image was taken under four different light wavelengths, two in visible light (green and red) and two infrared images. The task of the original dataset is to classify the image into the soil category of the observed region.
2. **Content**
   The dataset has 36 columns and 5100 rows collected from satellite observations. And each column is pixel of the image.

Approach of coding strategy:

1. **Filtering the data**
   The first thing to do while working with any data this big is to analyse it. We need to see if their any null value then we should try to remove it or set the value equal to zero. If there is any string value then remove that column.
2. **Forming the covariance matrix**

```
self.mu = mean(X)
X = X - self.mu
```

```
# Eigen Decomposition of the covariance matrix
C = X.T @ X / (n-1)
self.eigenvalues, self.eigenvectors = eigen_decomposition(C)
```

3. **Centring around the mean:**
   Mean is across the column

```
def mean(a):
    result=[]

    a= np.array(a)
    for i in range(a.shape[1]):
        sum = 0
        counter = 0
        for j in range(a.shape[0]):
            sum = sum + a[j][i]
            counter = counter + 1
        mean = sum/counter
        result.append(mean)
    return np.array(result)
```

```
    # subtract off the mean to center the data.
    self.mu = mean(X)
    X = X - self.mu
```

4. Eigenvalues and eigen vectors:

For finding the eigenvalues and eigen vectors we are using the Householder reflection and QR decomposition.

```
def householder_reflection(a, e):
    '''
    Given a vector a and a unit vector e,
    (where a is non-zero and not collinear with e)
    returns an orthogonal matrix which maps a
    into the line of e.
    '''
    #just to be safe
    assert a.ndim == 1
    assert np.allclose(1, sumfunction(e**2))

    '''
    a and norm(a) * e are of equal length so
    form an isosceles triangle. Therefore the third side of the triangle is
    perpendicular to the line that bisects the angle
    between a and e. This third side is given
    by a - ||a|| e, which we will call u.

    Since u lies in the plane spanned by a and e its clear that u is actually
    orthogonal to a plane equadistant to both a and ||a|| e. This is our
    plane of reflection. We normalize u to v to because a unit vector is
    required in the next step.
    '''
    u = a - signfunction(a[0]) * normalise(a) * e
    v = u / normalise(u)

    '''
      Derivation of the matrix form of a reflection:
      x - 2<x, v>v ==
      x - 2v<x, v> ==
      Ix - 2 v (v^T x) ==
      Ix - 2 (v v^T) x ==
```

```
        (I - 2v v^T) x == H x
    '''
    H = eyes(len(a)) - 2 * outer(v, v)

    return H


def qr_decomposition(A):
    '''
    Given an n x m invertable matrix A, returns the pair:
        Q an orthogonal n x m matrix
        R an upper triangular m x m matrix
    such that QR = A.
    '''
    n, m = A.shape
    assert n >= m


    # Q starts as a simple identity matrix.
    # R is not yet upper-triangular, but will be.
    Q = eyes(n)
    R = A.copy()

    # if the matrix is square, we can stop at m-1
    # because there are no elements below the pivot
    # in the last column to zero out. Otherwise we
    # need to do all m columns.
    for i in range(m - int(n==m)):

        # we don't actually need to construct it,
        # but conceptually we're working to update
        # the minor matrix R[i:, i:] during the i-th
        # iteration.
        # the first column vector of the minor matrix.

        r = R.iloc[i:, i]

        # if r and e are already co-linear then we won't
        # be able to construct the householder matrix,
        # but the good news is we won't need to!

        if np.allclose(r[1:], 0):
            continue

        # e is the i-th basis vector of the minor matrix.
        e = zeros(n-i)
        e[0] = 1

        # The householder reflection is only
        # applied to the minor matrix - the
        # rest of the matrix is left unchanged,
        # which we represent with an identity matrix.
        # Note that means H is in block diagonal form
        # where every block is orthogonal, therefore H
        # itself is orthogonal.
        H = eyes(n)
        H[i:, i:] = householder_reflection(r, e)

        # QR = A is invariant. Proof:
        # QR = A, H^T H = I =>
        # Q H^T H R = A =>
        # Q' = Q H^T, R' = H R =>
        # Q' R' = A. QED.
```

```
            # By construction, the first column of the
            # minor matrix now has zeros for all
            # subdiagonal matrix. By induction, we
            # have that all subdiagonal elements in
            # columns j<=i are zero. When i=N, R
            # is upper triangular.
            Q = (Q @ H.T)
            R = H @ R
            return Q, R

    def eigen_decomposition(A, max_iter=100):
        A_k = A
        Q_k = eyes(A.shape[1])

        for k in range(max_iter):
            Q, R = qr_decomposition(A_k)
            Q_k = Q_k @ Q
            A_k = R @ Q

        eigenvalues = diag(A_k)
        eigenvectors = Q_k
        return eigenvalues, eigenvectors
```

5. **Sorting the eigen values and the vectors**

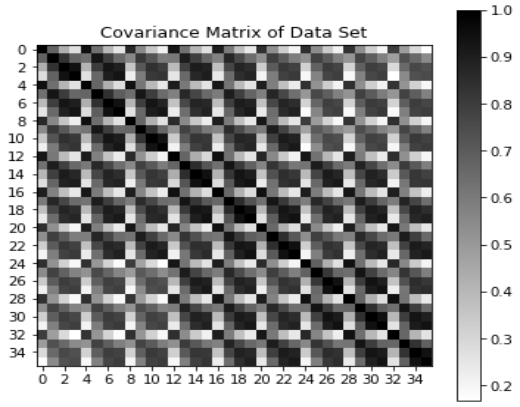
   The QR algorithm tends to puts eigenvalues in descending order but is not guaranteed to. To make sure we use argsort.

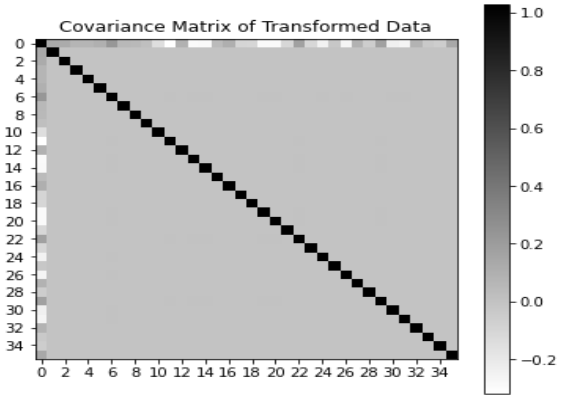
```
def flip(a):
    answer=a[::-1]
    return np.array(answer)
```

```
# the QR algorithm tends to puts eigenvalues in descending order
# but is not guarenteed to. To make sure, we use argsort.
descending_order = flip(np.argsort(self.eigenvalues))
self.eigenvalues = self.eigenvalues[descending_order]
self.eigenvectors = self.eigenvectors[:, descending_order]
```

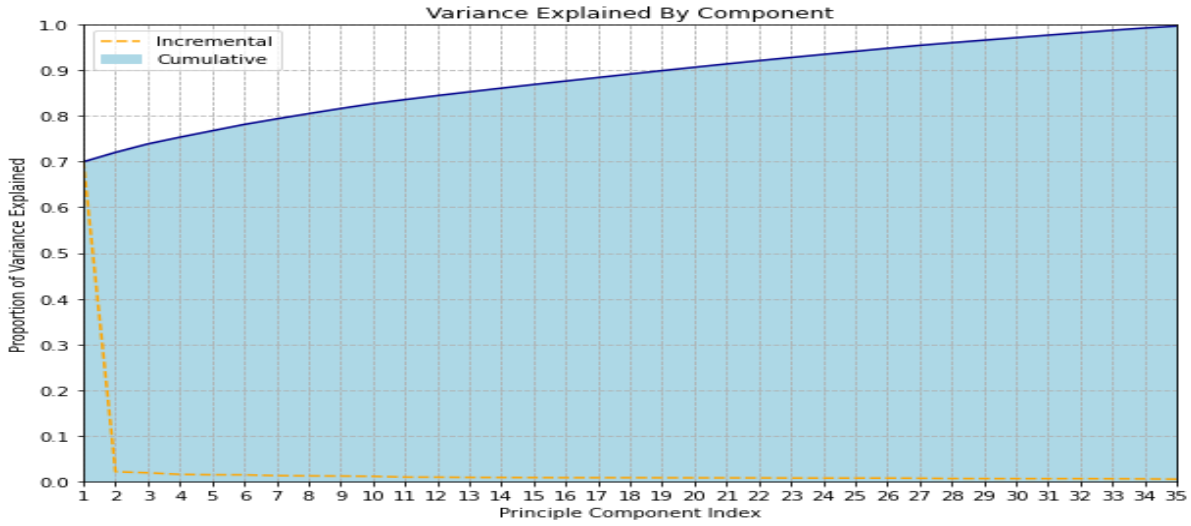
Simulations and outputs:

Covariance of Data before PCA Covariance of Data after PCA





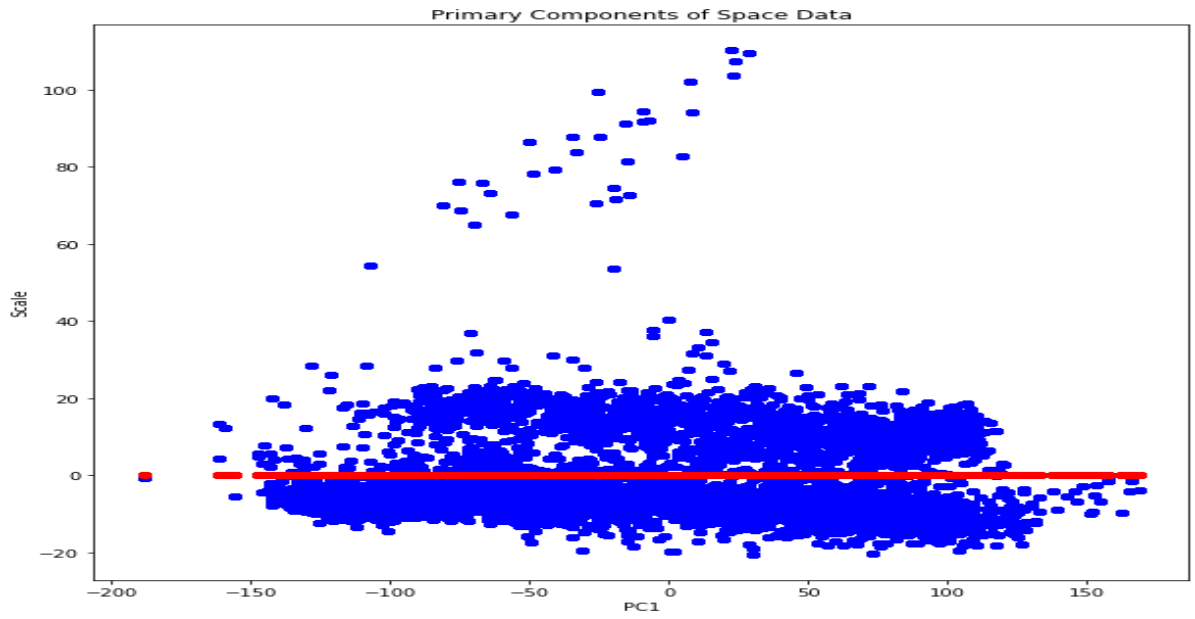
Variance Explained of PCA




```
Proportion Variance [0.69966203 0.02071008 0.01837908 0.01481108 0.01400703 0.01391626
 0.012085   0.01158484 0.01124596 0.01066508 0.00886475 0.00874324
 0.0082401  0.00811041 0.00789912 0.00768168 0.00765897 0.00764854
 0.00750869 0.00736256 0.00729322 0.00719095 0.00687327 0.00674181
 0.00673837 0.00672412 0.00639813 0.0057584  0.00559436 0.00553592
 0.00548161 0.00547384 0.00526689 0.00494884 0.0042875  0.00290825]
```

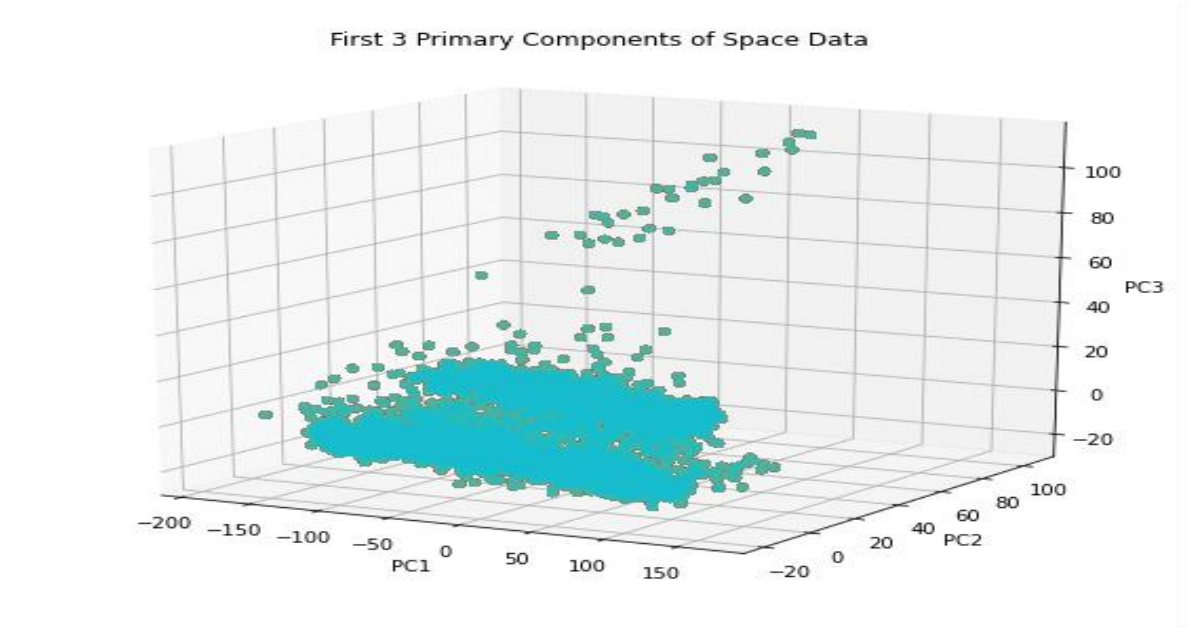
We can see that after performing the PCA we have significance of around 70%. And maximum data variance is across the principle component 1.

Graphs along Principle Component 1



From the graph we can see that the dataset has some outliers

Following graphs is across 3 principle component



Conclusion

On a closing note this project was a great chance for us to learn the principles of linear algebra such as QR Decomposition, Householder Reflection. In addition to the fact that we learned substantially more about them and applied it. This was an extraordinary hand on movement for us and we saw how the ideas are applied, considering all things. In addition to this, there are many other elements of linear algebra that we still have to discover and apply. But apart from that, engaging with the team was a remarkable opportunity, and getting their input on how to go forward to tackle a specific challenge and eventually finished the project.

Each member of the group made an equal contribution to the analysis of the project. With Jap and Purvam being the better coder implemented most of the code. The dataset to be used for the project was cleaned up by Mohit. Nihar and Raj first solved the problems on paper, which Jap and Purvam then implemented. Mohit did the formatting of the report. Overall, the report was a collaborative activity by the whole group, and because of the group work, it was possible.

Reference:

Patel, Ankur. "Hands-On Unsupervised Learning Using Python." *O'Reilly Online Learning*, www.oreilly.com/library/view/hands-on-unsupervised-learning/9781492035633/ch04.html. Accessed 6 Nov. 2020.

Likebupt. "PCA-Based Anomaly Detection: Module Reference - Azure Machine Learning." *Microsoft Docs*, 22 Feb. 2020, docs.microsoft.com/en-us/azure/machine-learning/algorithm-module-reference/pca-based-anomaly-detection.

"Principal Component Analysis." *Dr. Sebastian Raschka*, 27 Jan. 2015, sebastianraschka.com/Articles/2015_pca_in_3_steps.html.

"Implementing a Principal Component Analysis (PCA)." *Dr. Sebastian Raschka*, 13 Apr. 2014, sebastianraschka.com/Articles/2014_pca_step_by_step.html.

Dataman. "Anomaly Detection with PyOD! - Towards Data Science." *Medium*, 17 Oct. 2020, towardsdatascience.com/anomaly-detection-with-pyod-b523fc47db9.

Hui, Jonathan. "Machine Learning — Singular Value Decomposition (SVD) & Principal Component Analysis (PCA)." *Medium*, 8 Feb. 2020, jonathan-hui.medium.com/machine-learning-singular-value-decomposition-svd-principal-component-analysis-pca-1d45e885e491.

"Principal Component Analysis in Python | Basics of Principle Component Analysis Explained | Edureka." *YouTube*, uploaded by edureka!, 4 Oct. 2019, www.youtube.com/watch?v=n7npKX5zIWI.

"Householder Transformation Introduction." *YouTube*, uploaded by Poujh, 26 Mar. 2019, www.youtube.com/watch?v=oOHNgFtBFUo&list=PLOW1obrRCUQlqXcKV2tNG5QgKY_v3XNKb.