How to manually calculate PCA: In Data Science and Machine Learning, Principal Component Analysis (PCA), is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process. PCA finds directions of maximal variance of data. It finds directions that are mutually orthogonal. Mutually orthogonal means it's a global algorithm. Global means, that all the directions, all the new features that they find have a big global constraint, namely that they must be mutually orthogonal. Let's see how can we manually compute PCA given some random table of values (see the illustration): Step 1: Standardize the dataset Step 2: Calculate the covariance matrix for the features in the dataset Step 3: Calculate the eigenvalues and eigenvectors for the covariance matrix Step 4: Sort eigenvalues and their corresponding eigenvectors Step 5: Calculate eigenvector for each eigenvalue using Cramer's rule Step 6: Build eigenvectors matrix Step 7: Pick k eigenvalues and form a matrix of eigenvectors Step 8: Transform the original matrix Principle Component Analysis (PCA) vs Linear Discriminant Analysis (LDA) In Data Science and Machine Learning, PCA is an unsupervised dimensionality reduction technique, that ignores the class label. PCA focuses on capturing the direction of maximum variation in the data set. LDA is a supervised dimensionality reduction method, that focuses on finding a feature subspace that maximizes the separability between the groups. PCA performs better in cases where a number of samples per class is less. · LDA works better with large datasets having multiple classes; class separability is an important factor while reducing the dimensionality import matplotlib.image as mpimg img = mpimg.imread('C:\\Users\\u\\Desktop\\Analysis data\\PCA\_Image.jpg') plt.figure(figsize=(16,10)) plt.axis('off') plt.imshow(img) plt.show() A = 2.51579324, 1.0652885, 0.39388704, 0.02503121 solving  $(A - \lambda I)v = 0$  for v with different  $\lambda$  values:  $f_2$   $cov(f_2, f_1)$   $var(f_2)$   $cov(f_2, f_2)$   $cov(f_2, f_4)$ 0.028490  $f_3 = cov(f_3, f_1) = cov(f_3, f_2) = var(f_3) = cov(f_3, f_4)$  $f_4 = cov(f_4, f_1) = cov(f_4, f_2) = cov(f_4, f_2) = var(f_4)$ solving  $(A - \lambda I)v = 0$  for every  $\lambda$  using Cramer's rule  $(-1.0-0)^2 + (0.33-0)^2 +$  $(-1.0-0)^2 + (0.33-0)^2 + (1.33-0)^2$  $\lambda_2 = 1.0652885$ A1 = 2.51579324 μ mean per column  $\lambda_3 = 0.39388704$  $\lambda_4 = 0.02503121$  $v_1 = -0.307071$ σ std per column  $v_1 = -0.917059$  $(-1.0-0) \cdot (0.632456-0) +$  $v_2 = -0.817319$  $v_2 = -0.52404813$  $v_2=0.206922$  $(-0.33 - 0) \cdot (1.264911 - 0) +$  $v_3 = 0.188250$  $v_3 = -0.720099$  $v_2 = -0.58589647$   $v_2 = -0.320539$  $(-1.0-0) \cdot (0.632456-0) +$  $v_4 = 0.449733$  $v_4 = 0.654547$  $v_4 = -0.59654663$  $v_4 = -0.115935$  $(0.33 - 0) \cdot (0.000000 - 0) +$  $(1.33 - 0) \cdot (-1.264911 - 0))$ (6 0.16195986 -0.917059 -0.3070710.196162 -0.52404813 0.206922 -0.58589647 -0.320539 -0.59654663 -0.115935 -0.252980.8 0.51121 0.4945 sort eigenvalues & corresponding eigenvectors 0.03849 0.51121 0.8 0.75236 -0.917059 0.161960 eigenvalues are already sorted -1 -0.63246 0.26062 pick k eigenvalues and form matrix of eigenvectors 0.33333 1.73205 1.56374 -0.585896 -0.320539select top 2 eigenvectors & matrix will look like -0.596547 -0.115935/ v - eigen vector 0.260623 0.16195986 λ - eigen value -0.6324560.000000 0.33333 -1.04249 1.563740 -0.173749 0.333333 1.732051 1.264911 -0.52404813 0.206922 0.632456 -0.577350-0.58589647 -0.320539 -0.59654663 -0.115935/ 1.333333 -1.264911 -0.577350 -0.608121-0.252980.4945 8 0.75236 0.014003 0.755975  $\sum (x_i - \bar{x}) \cdot (y_i - \bar{y})$ -2.556534-0.780432**PCA** -0.14479 0.4945 -0.0514801.253135 1.014150 0.000239 img = mpimg.imread('C:\\Users\\u\\Desktop\\Analysis data\\PCA\_vs\_LDA.jpg') plt.figure(figsize=(16,10)) plt.axis('off') plt.imshow(img) plt.show() Maximizing the variance Maximizing component and minimizing SSE axes for class separation y A  $LD_1$  $LD_2$ **PCA** LDA Classification projection LDA Dimensionality Reduction Principal behind PCA  $PC_1$ Practical Implementation of Principal Component Analysis(PCA) In [4]: import pandas as pd import numpy as np import matplotlib.pyplot as plt from scipy.linalg import eig from numpy.linalg import eig In [5]: # 2-D Data data = np.array([[3,4],[2,8],[6,9],[10,12]])In [6]: data array([[ 3, 4], [ 2, 8], [ 6, 9], [10, 12]]) **#Create DataFrame** df = pd.DataFrame(data, columns = ['ML', 'DL']) In [8]: **2** 6 9 **3** 10 12 In [9]: #Plot it on Scater plot plt.scatter(df['ML'], df['DL']) <matplotlib.collections.PathCollection at 0x132dd97f580> 12 11 10 PCA STEPS: '''1).Standarization of the data (Zero centric) 2).COV matrix 3). Eigen value and Eigen vector 4).Find Principal Component''' '1).Standarization of the data (Zero centric)\n 2).COV matrix\n 3).Eigen value and Eigen vector\n 4).Find Principal Component' Out[11]: array([[ 3, 4], [10, 12]]) In [12]: #Transpose the data data.T array([[ 3, 2, 6, 10], [ 4, 8, 9, 12]]) In [13]: #Find the mean np.mean(data) Out[13]: In [14]: #Mean Row wise np.mean(data, axis = 1)array([ 3.5, 5. , 7.5, 11. ]) In [15]: #Mean Column wise mean\_by\_col = np.mean(data.T, axis = 1) STEP\_01 - Standard Normal Distribution In [16]: scaled\_data = data - mean\_by\_col In [17]: scaled\_data array([[-2.25, -4.25], [-3.25, -0.25], [ 0.75, 0.75], [ 4.75, 3.75]]) #Not doing division by sigma as (sigma = 1) and data set too small STEP\_02 - COV matrix In [19]: #Relation b/W variable cov\_mat = np.cov(scaled\_data.T) STEP 03 - Eigen value and Eigen vector In [20]: eig\_value, eig\_vector = np.linalg.eig(cov\_mat) eig\_value array([21.55203266, 2.28130068]) eig\_vector Out[22]: array([[ 0.74289445, -0.66940857], [ 0.66940857, 0.74289445]]) STEP 04 - Principal Component In [23]: #Transpose the data eig\_vector.T.dot(scaled\_data.T).T array([[-4.51649894, -1.65113213], [-2.58175911, 1.98985424], [ 1.05922727, 0.05511441], [ 6.03903078, -0.39383652]]) #Import PCA from sklearn.decomposition from sklearn.decomposition import PCA pca=PCA() In [25]: #Transform the scaled data pca.fit\_transform(scaled\_data) array([[-4.51649894, -1.65113213], [-2.58175911, 1.98985424], [ 1.05922727, 0.05511441], [ 6.03903078, -0.39383652]]) In [26]: #Create DataFrame for scaled data pd.DataFrame(data = pca.fit\_transform(scaled\_data), columns = ['PC1', 'PC2']) PC2 Out[26]: PC1 **0** -4.516499 -1.651132 **1** -2.581759 1.989854 **2** 1.059227 0.055114 **3** 6.039031 -0.393837 In [27]: **#Inverse The Data** pca.inverse\_transform(pca.fit\_transform(scaled\_data)) array([[-2.25, -4.25], [-3.25, -0.25], [ 0.75, 0.75], [ 4.75, 3.75]]) In [28]: scaled\_data array([[-2.25, -4.25], [-3.25, -0.25], [ 0.75, 0.75], [ 4.75, 3.75]]) #Variance ratio b/w PC1 and PC2 pca.explained\_variance\_ratio\_ array([0.90428109, 0.09571891]) 0.90428109+0.09571891 Out[30]: 1.0 We have to select PC1 as it covers 90% of dataset variation In [ ]: