COSC 3337 : Data Science I

N. Rizk

College of Natural and Applied Sciences

Department of Computer Science

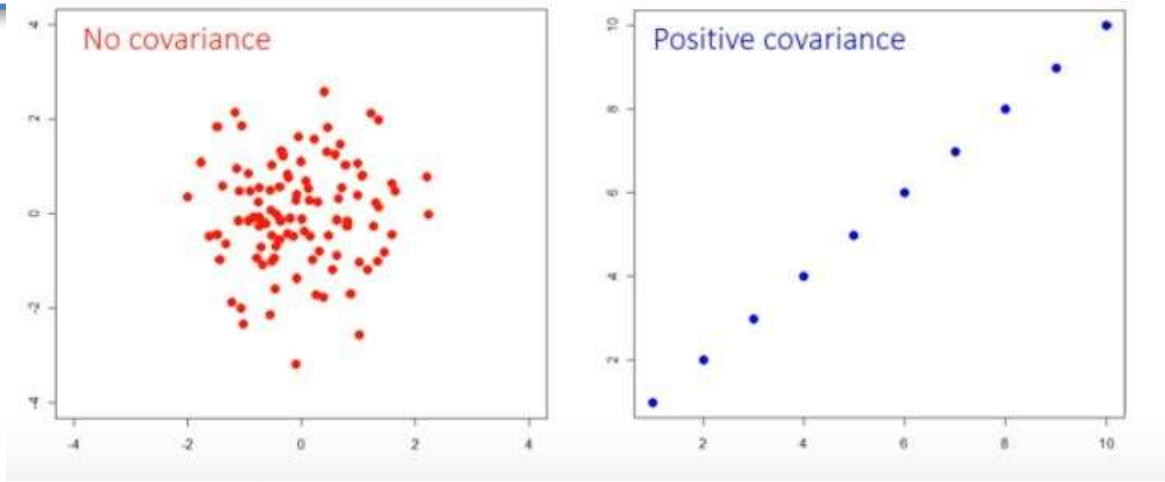
University of Houston

Reduce Dimensionality

A more common way of speeding up a machine learning algorithm is by using Principal Component Analysis **(PCA)**

Recall : co-variance

A measure of how much two variables change together:



Measure how two variables

➔change together if one variable changes the other changes or not

In case of multi-variate e.g measuring 3 variables

➔Use covariance matrix

$$\begin{array}{c} \\ x1 \\ x2 \\ x3 \end{array} \begin{array}{c} \begin{array}{ccc} x1 & x2 & x3 \end{array} \\ \begin{bmatrix} var(x1) & cov(x1,x2) & cov(x1,x3) \\ cov(x2,x1) & var(x2) & cov(x2,x3) \\ cov(x3,x1) & cov(x3,x2) & var(x3) \end{bmatrix} \end{array}$$

It would be easier to rescale features (to remove variance)

Eigenvectors of a Matrix

Square *n*-dimensional matrices have *n* eigenvectors

Let's work with a simpler matrix first:

$$\begin{array}{c} \\ \boldsymbol{x1} \\ \boldsymbol{x2} \end{array} \begin{array}{c} \begin{array}{cc} \boldsymbol{x1} & \boldsymbol{x2} \end{array} \\ \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \end{array}$$

An **eigenvector** is a vector that, when multiplied with the matrix, yields a result that is the eigenvector multiplied by an integer *I*

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} e1 \\ e2 \end{bmatrix} = I \begin{bmatrix} e1 \\ e2 \end{bmatrix}$$

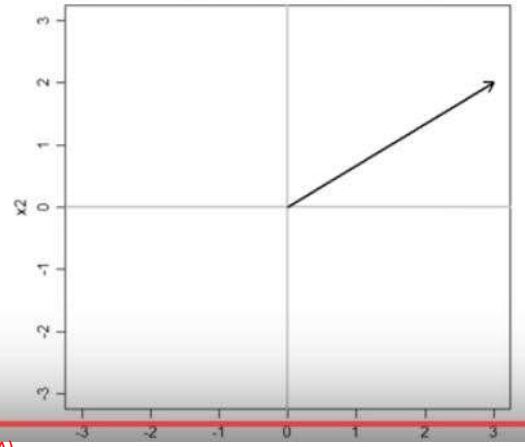
$$\boldsymbol{Av = \lambda v}$$

Here, $\boldsymbol{v}$ is the eigenvector and $\boldsymbol{\lambda}$ is the eigenvalue associated with it.

Eigenvectors of a Matrix

Eigenvectors are typically scaled to have a length of 1

Eigenvector [3,2] displayed graphically



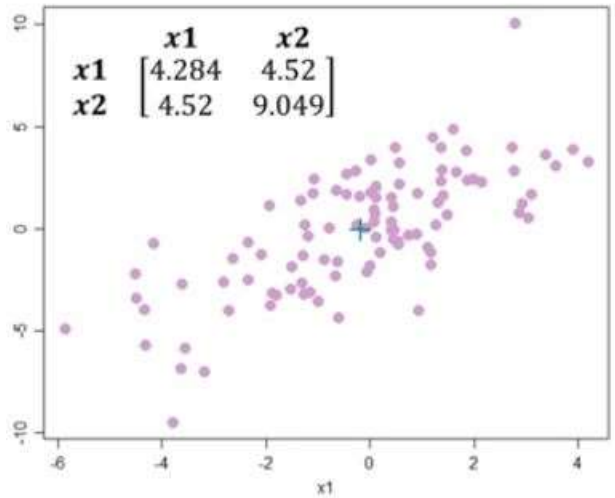
Eigenvector $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$ has a length of $\sqrt{3^2 + 2^2}$, or $\sqrt{13}$

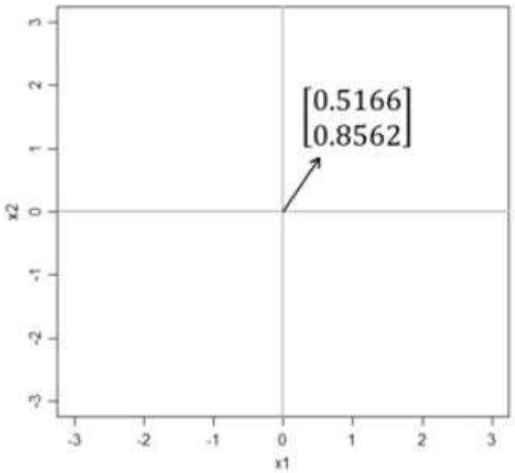
Scaled eigenvector is:

$$\begin{bmatrix} 3/\sqrt{13} \\ 2/\sqrt{13} \end{bmatrix}$$

Eigenvectors of a Matrix

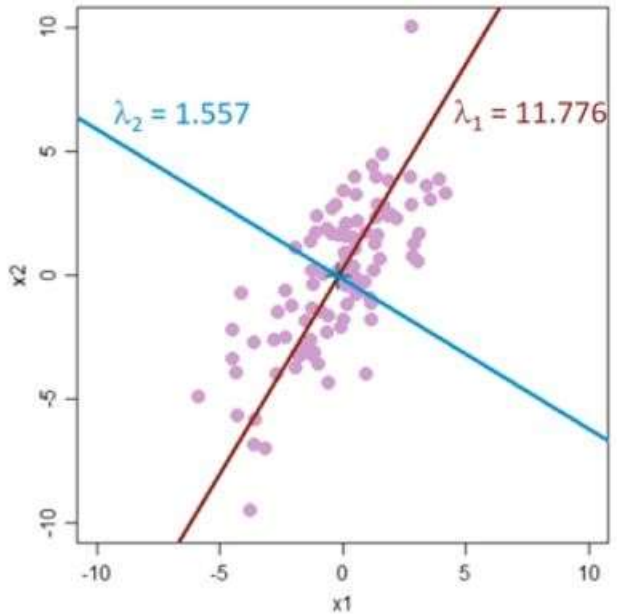
The two scaled eigenvectors of the example dataset are $\begin{bmatrix}0.5166\\0.8562\end{bmatrix}$ and $\begin{bmatrix}-0.8562\\0.5166\end{bmatrix}$. As vectors, they are directions on the plot.





Rescaling to Remove Covariance

Remove covariance by treating each eigenvector as a new axis, shrink axis by $\sqrt{\lambda_i}$, then calculate distance between points



Principal Component Analysis (PCA)

What is it ?

is to reduce the dimensionality of a data set while retaining the variation present in the dataset, up to the maximum extent.

How ?

by transforming the variables to a new set of variables, which are known as the principal components (or simply, the PCs) and are orthogonal

How many ?

ordered PCA such that the retention of variation present in the original variables decreases as we move down in the order. So, in this way, the 1st principal component retains maximum variation that was present in the original components.

Why they are orthogonal?

The principal components are the eigenvectors of a covariance matrix, and hence they are orthogonal.

The dataset on which PCA technique is to be used must be scaled.

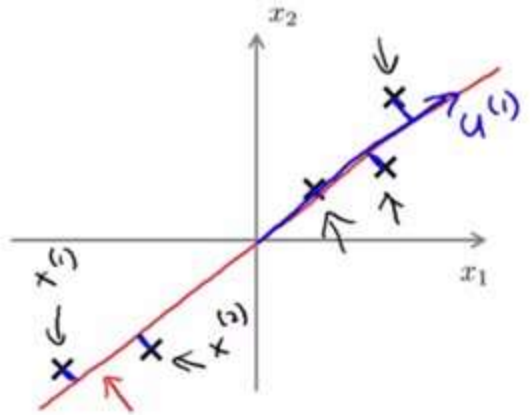
PCA's Benefit

1. Reduce the number of attributes
2. Find out which attribute is more valuable of separating samples (The variation present in the PCs decrease as we move from the 1st PC to the last one, hence the importance)
3. Find out how accurate the split is

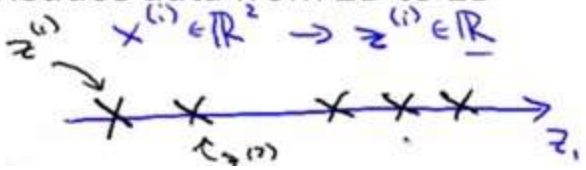
Dimensionality: It is the number of features,

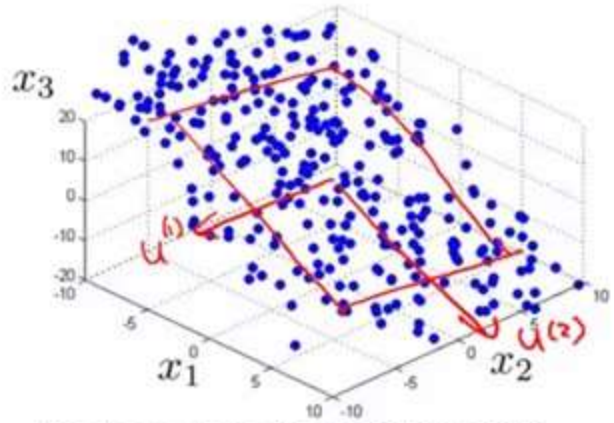
Example of feature reduction

Principal Component Analysis (PCA) algorithm



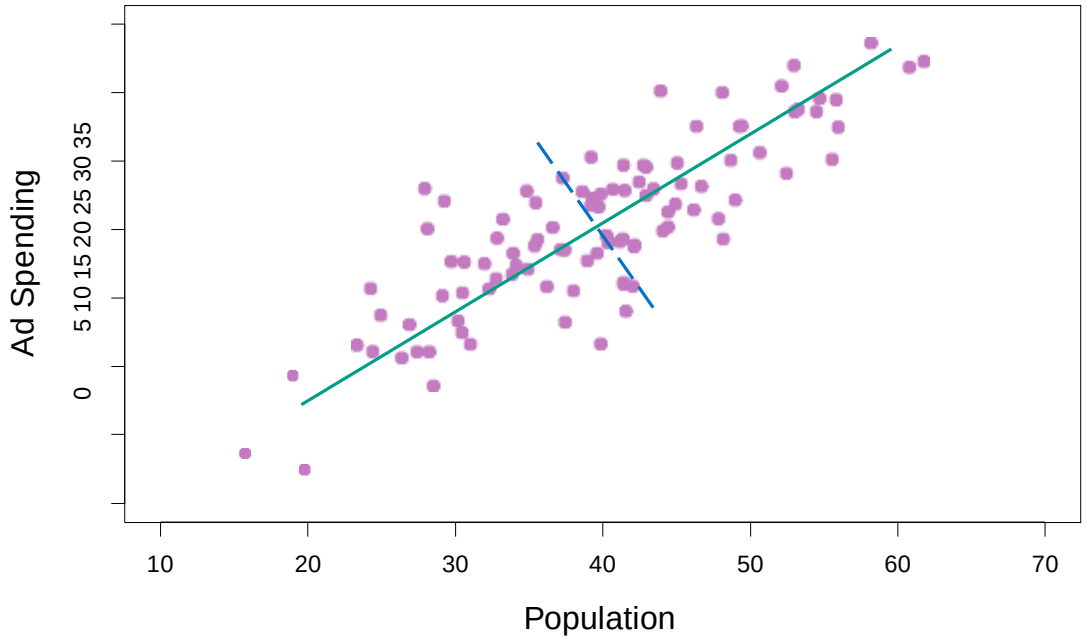
Reduce data from 2D to 1D





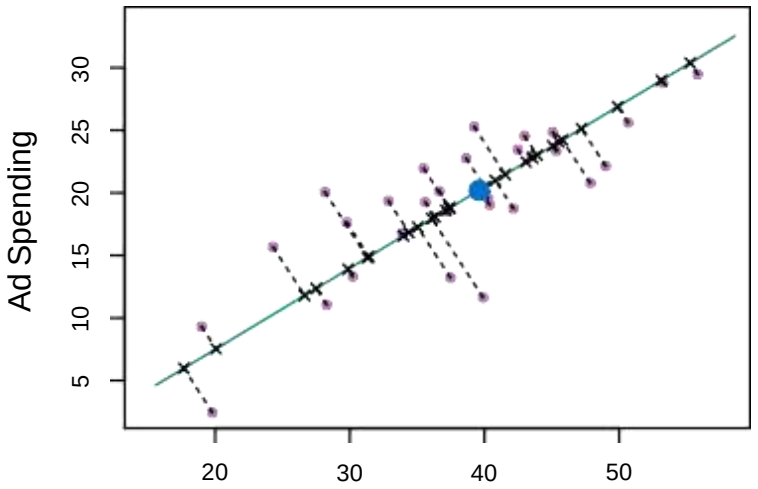
Reduce data from 3D to 2D

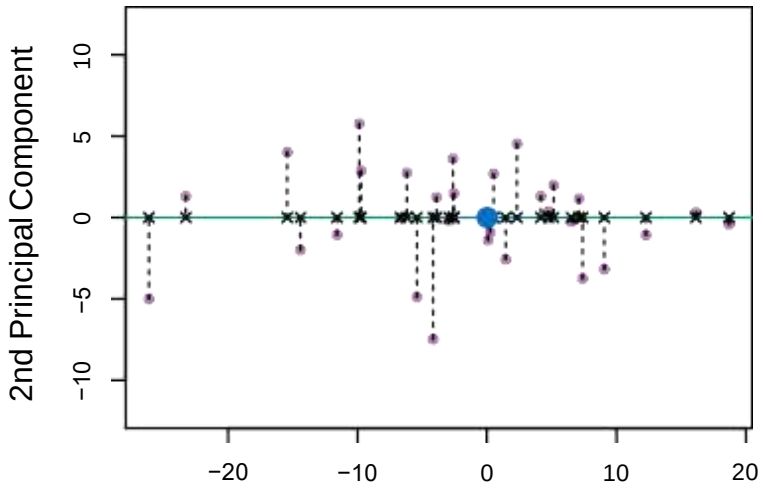
Pictures of PCA



The population size (`pop`*) and ad spending (*`ad`*) for* 100 *different cities are shown as purple circles. The green solid line indicates the first principal component, and the blue dashed line indicates the second principal component.*

Pictures of PCA: continued





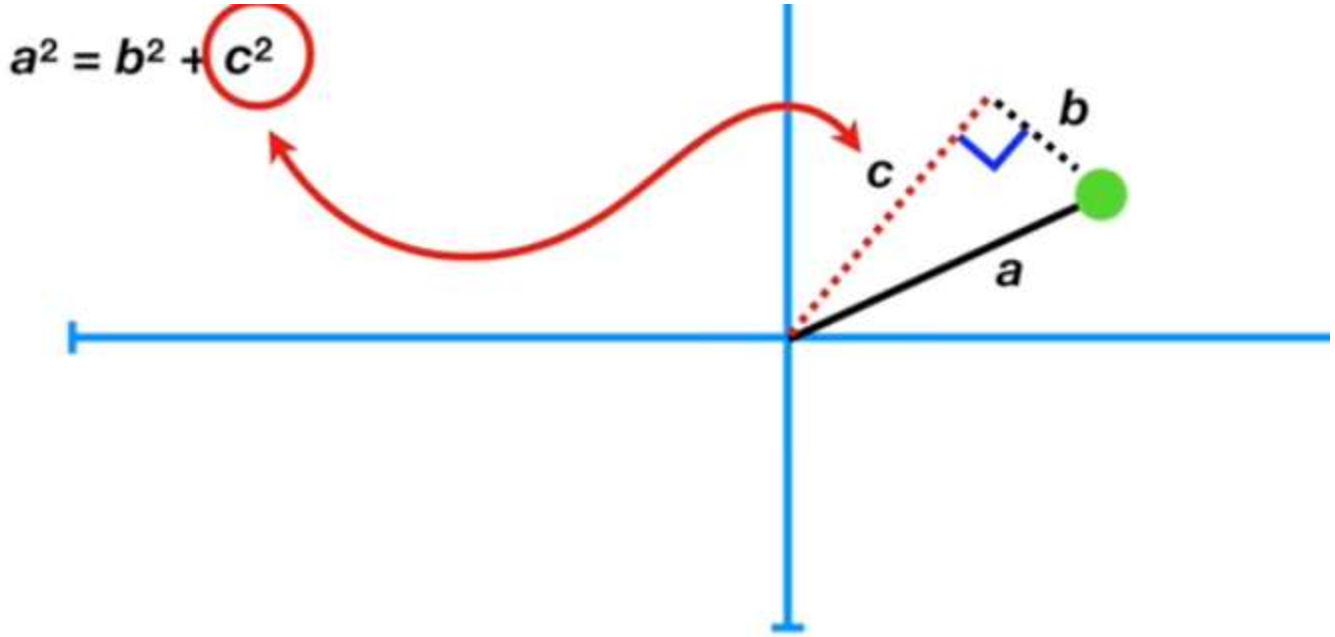
A subset of the advertising data.

Left: *The first principal component, chosen to minimize the sum of the squared perpendicular distances to each point, is shown in green. These distances are represented using the black dashed line segments.*

Right: *The left-hand panel has been rotated so that the first principal component lies on the x-axis.*

PCA1: Line with largest sum of squared distances between the projected points and the origin (slope of this line determine relationships between points slope 0.25 ➔ 4axis 1 y axis

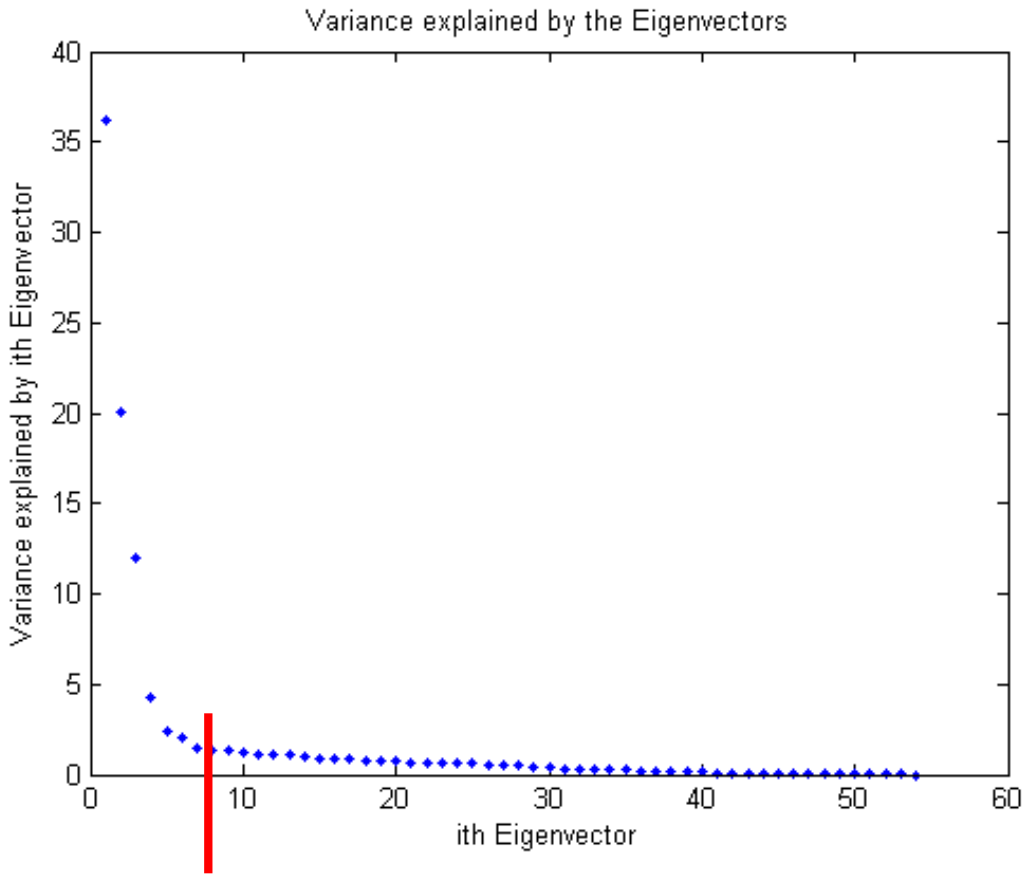
Best fit line dotted red



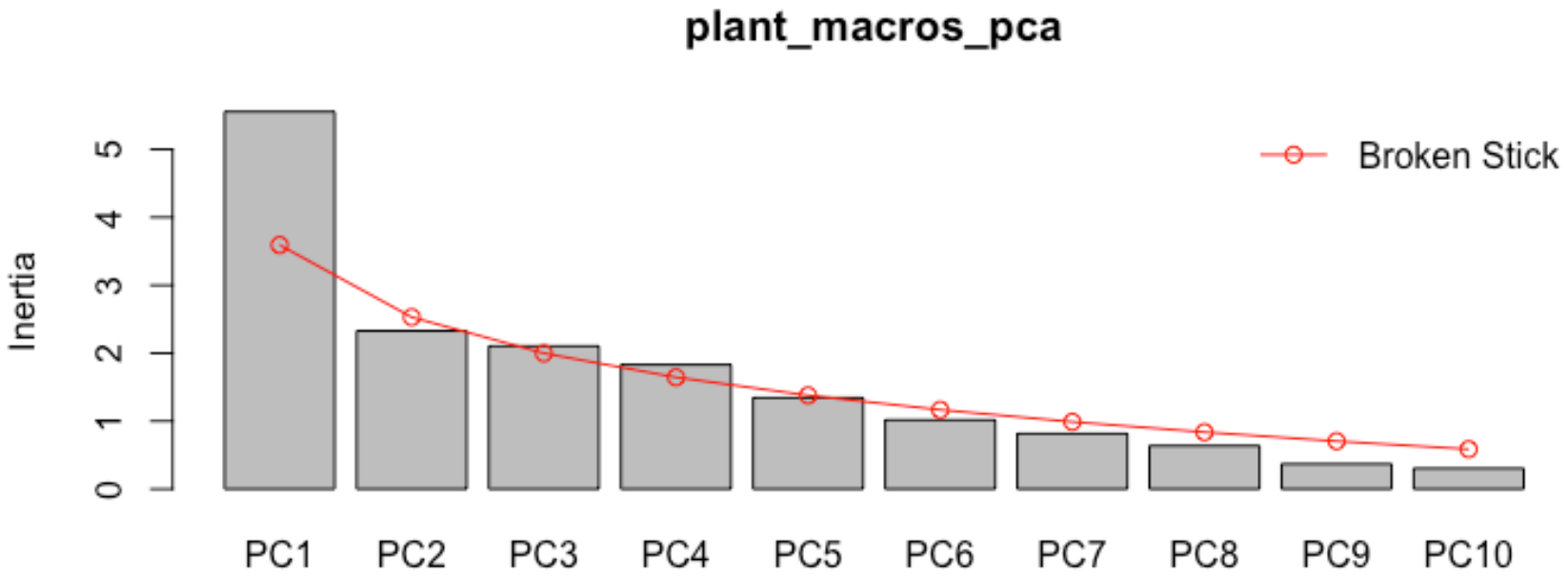
Purpose better fit = maximize sum of squares distance c
or minimize sum of squares distance b

How many components?

- Check the distribution of eigen-values (scree plot)
- Take enough many eigen-vectors to cover 80-90% of the variance



Screeplot histogram of PCAs



Implementing PCA Algorithm on a 2D dataset

- Step 1: Normalize the data
- by subtracting the respective means from the numbers in the respective column. So if we have two dimensions X and Y, all X become $\boldsymbol{x}$- and all Y become $\boldsymbol{y}$-. This produces a dataset whose mean is zero.

Implementing PCA Algorithm

- Step 2: Calculate the covariance matrix

Since the dataset we took is 2-dimensional, this will result in a 2x2 Covariance matrix.

$$Matrix(Covariance) = \begin{bmatrix} Var[X_1] & Cov[X_1, X_2] \\ Cov[X_2, X_1] & Var[X_2] \end{bmatrix}$$

Note

Var[X1] = Cov[X1,X1] and Var[X2] = Cov[X2,X2].

$$Cov(x,y) = \frac{\Sigma(x_i-\bar{x})(y_i-y)}{N-1}$$

Implementing PCA Algorithm on a 2D dataset

Step 3: Calculate the eigenvalues and eigenvectors

λ is an eigenvalue for a matrix A if it is a solution of the characteristic equation:

- $det(\lambda I - A) = 0$
- For each eigenvalue λ, a corresponding eigen-vector v, can be found by solving:
- $(\lambda I - A)v = 0$

Implementing PCA Algorithm on a 2D dataset

Step 4: Choosing components and forming a feature vector:

Order the eigenvalues from largest to smallest => the highest eigenvalue is the principal component of the dataset

To reduce the dimensions, choose the first *p* eigenvalues and ignore the rest.

Since we just have 2 dimensions➔ Feature Vector = (eig1, eig2)
= Singular vector decomposition SVD
=eig1 and eig2 eigen values or loading scores

Another way
to compute Eigen value of PC1: sum of squares of distances from origin to projected data
to compute singular value of pca1 : squareroot of the above

Implementing PCA Algorithm on a 2D dataset

Step 5: Forming Principal Components:

- $NewData = FeatureVector^T \times ScaledData^T$

- *NewData* is the Matrix consisting of the principal components,
- *FeatureVector* is the matrix we formed using the eigenvectors we chose to keep, and
- *ScaledData* is the scaled version of original dataset

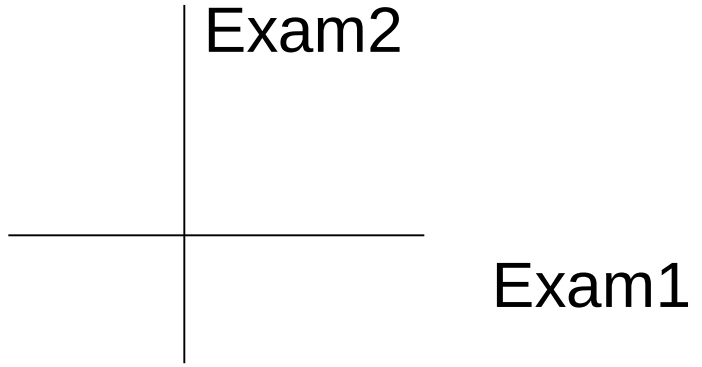
Summary PCA Algorithm

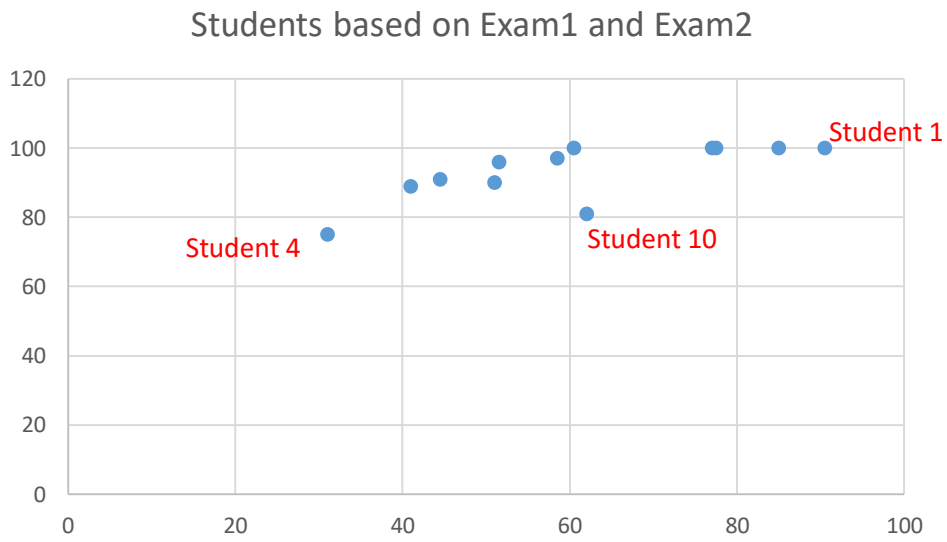
- PCA algorithm:
  - 1. X ← Create N x d data matrix, with one row vector x_n per data point
  - 2. X subtract mean *x* from each row vector x_n in X
  - 3. Σ ← covariance matrix of X
  - Find eigenvectors and eigenvalues of Σ
  - PC's ← the M eigenvectors with largest eigenvalues

PCA: Yet Another Explanation

Example

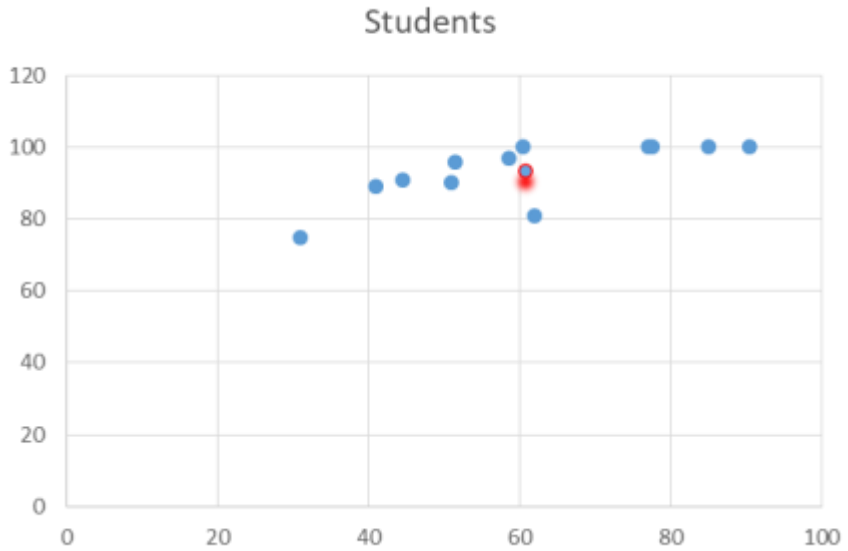
Student ID	Exam1	Exam2
Student 1	90.5	100
Student 2	44.5	91
Student 3	41	89
Student 4	31	75
Student 5	58.5	97
Student 6	77.5	100
Student 7	85	100
Student 8	51	90
Student 9	60.5	100
Student 10	62	81
Student 11	77	100
Student 12	51.5	96



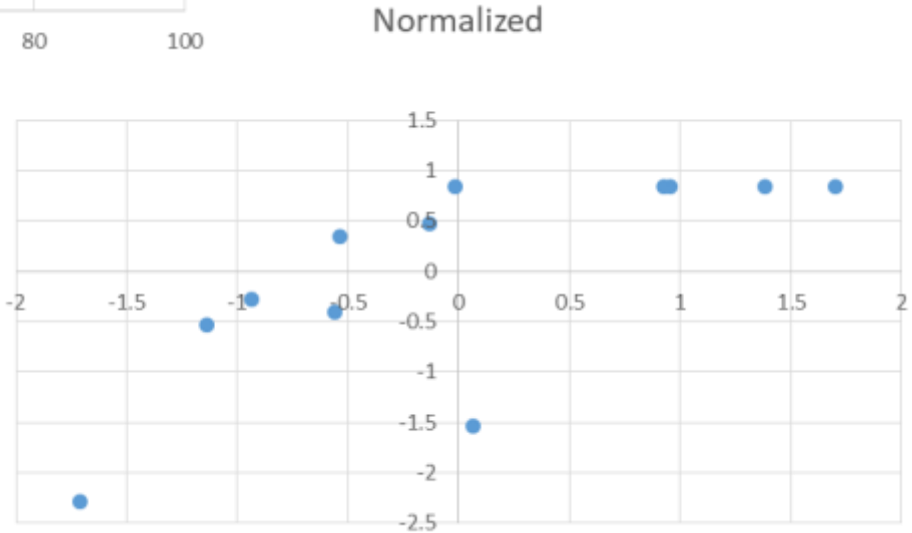


Average point (60.8,93.2)

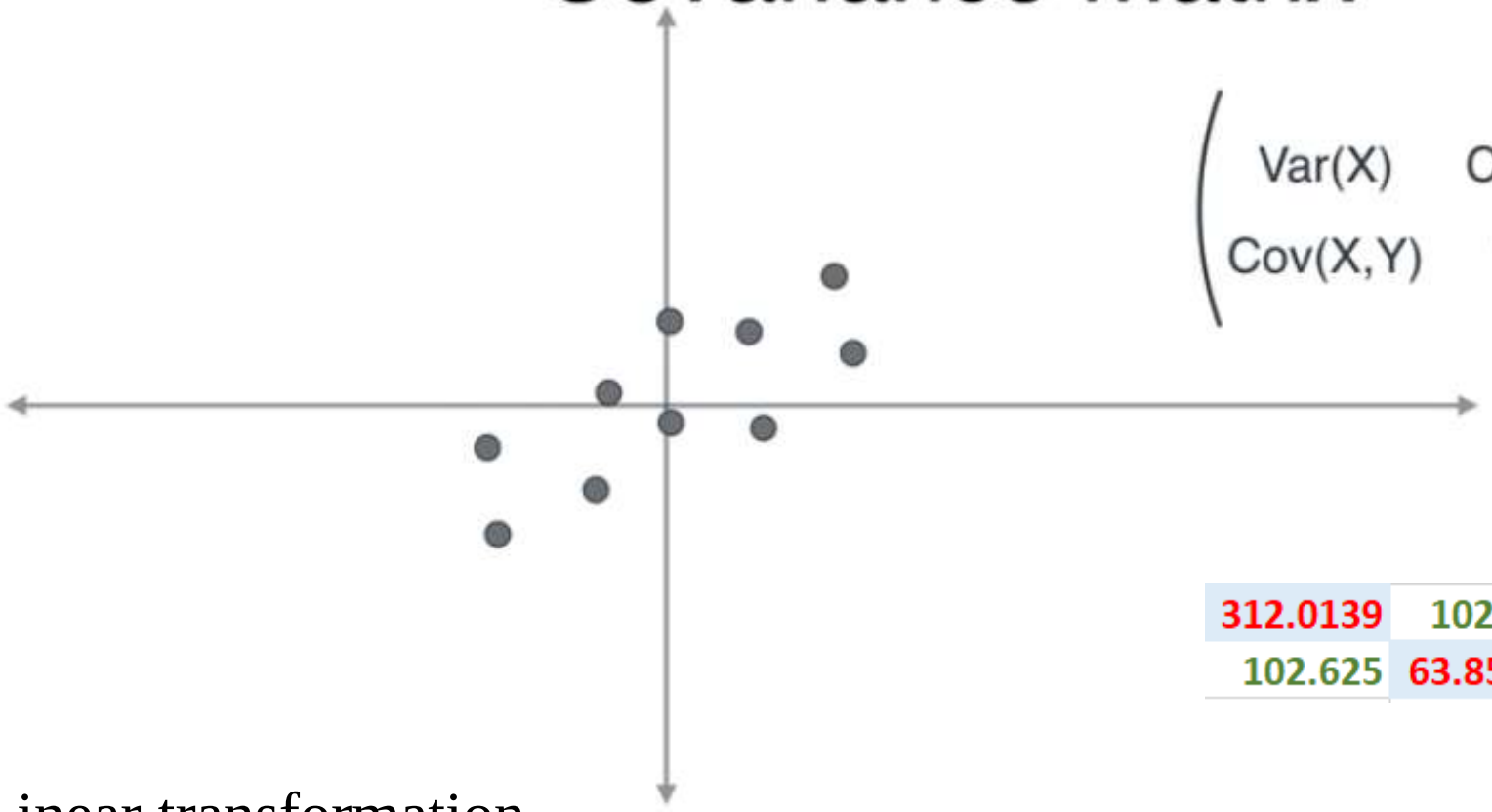
Z-scores



Student ID	Exam1	Exam2
Student 1	1.699392104	0.848913
Student 2	-0.935620372	-0.28297
Student 3	-1.136110451	-0.5345
Student 4	-1.708939251	-2.29521
Student 5	-0.133660053	0.471618
Student 6	0.954714665	0.848913
Student 7	1.384336264	0.848913
Student 8	-0.563281652	-0.40874
Student 9	-0.019094293	0.848913
Student 10	0.066830027	-1.54062
Student 11	0.926073225	0.848913
Student 12	-0.534640212	0.345853



Covariance matrix

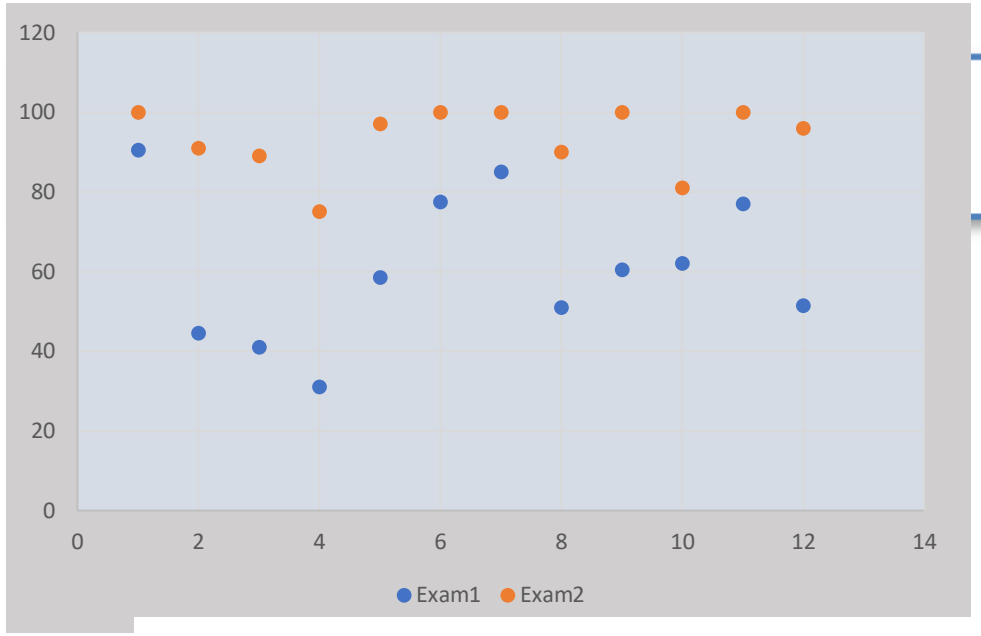


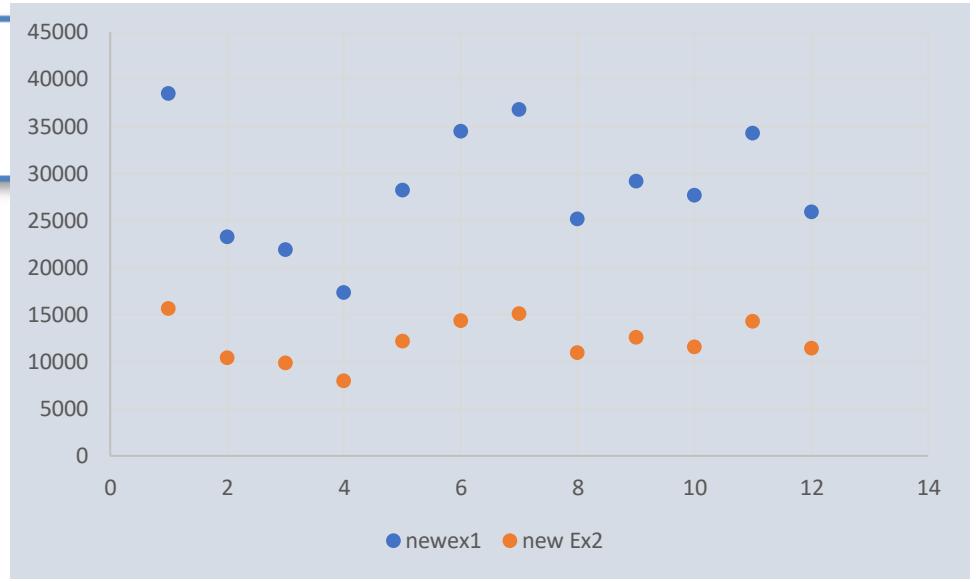
$$\begin{pmatrix} \text{Var}(X) & \text{Cov}(X,Y) \\ \text{Cov}(X,Y) & \text{Var}(Y) \end{pmatrix}$$

312.0139	102.625
102.625	63.85417

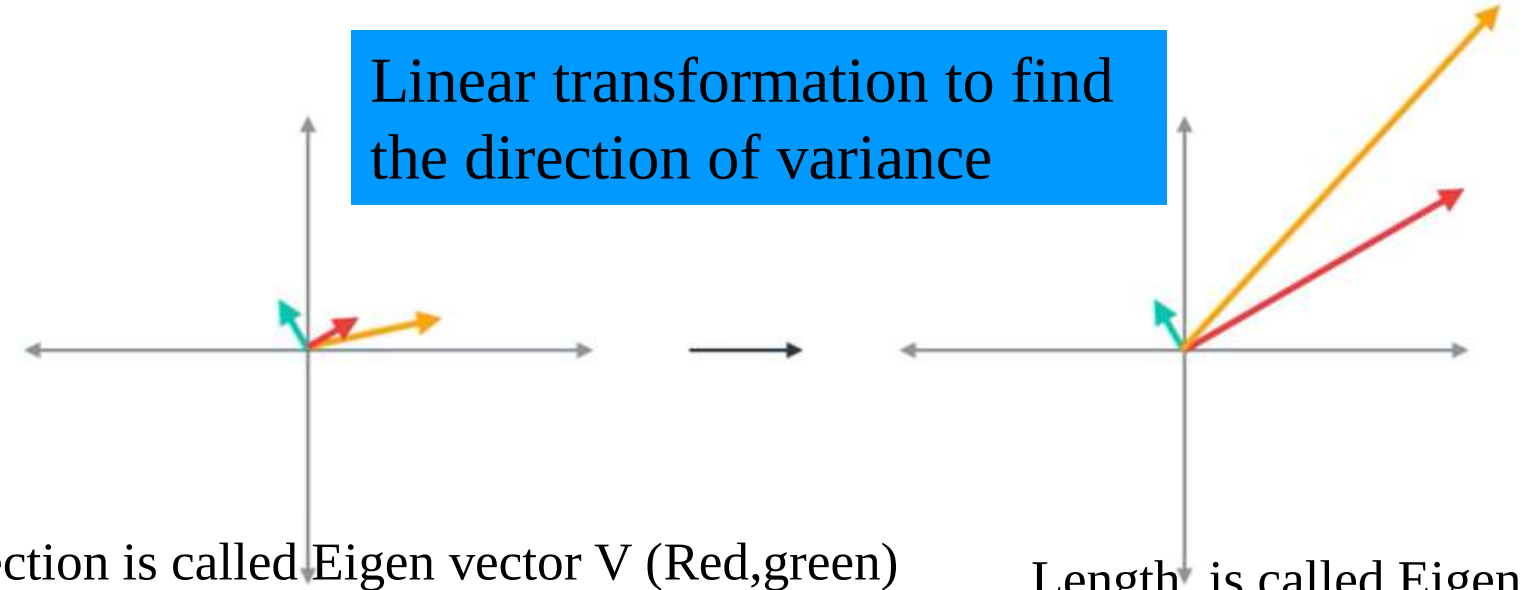
Linear transformation

- ➔ Student 2 =(44.5,91)
- ➔ ----------➔ (312.01*44.5+102,625*91,
  102.625*44.5+63.85*91)





Linear transformation to find the direction of variance



Direction is called Eigen vector V (Red,green)
Red is (10,6) Green (-1,1)

Length is called Eigen value Lambda e.g10

- Linear transformation matrix does not change the direction of 2 particular vectors but scaling them

312.0139	102.625
102.625	63.85417

$*V = \lambda * V$

$\det(\lambda - 312)(\lambda - 63) - (-102 * (-102))$

Find $\lambda 1$ and $\lambda 2$ ➔ $\lambda^2 - 63\lambda - 312\lambda + 312*63 + 102*102$

Replace in the above equation to find v

www.wolframalpha.com

$$\lambda_1 = \frac{3}{2}\left(125 + \sqrt{11513}\right)$$

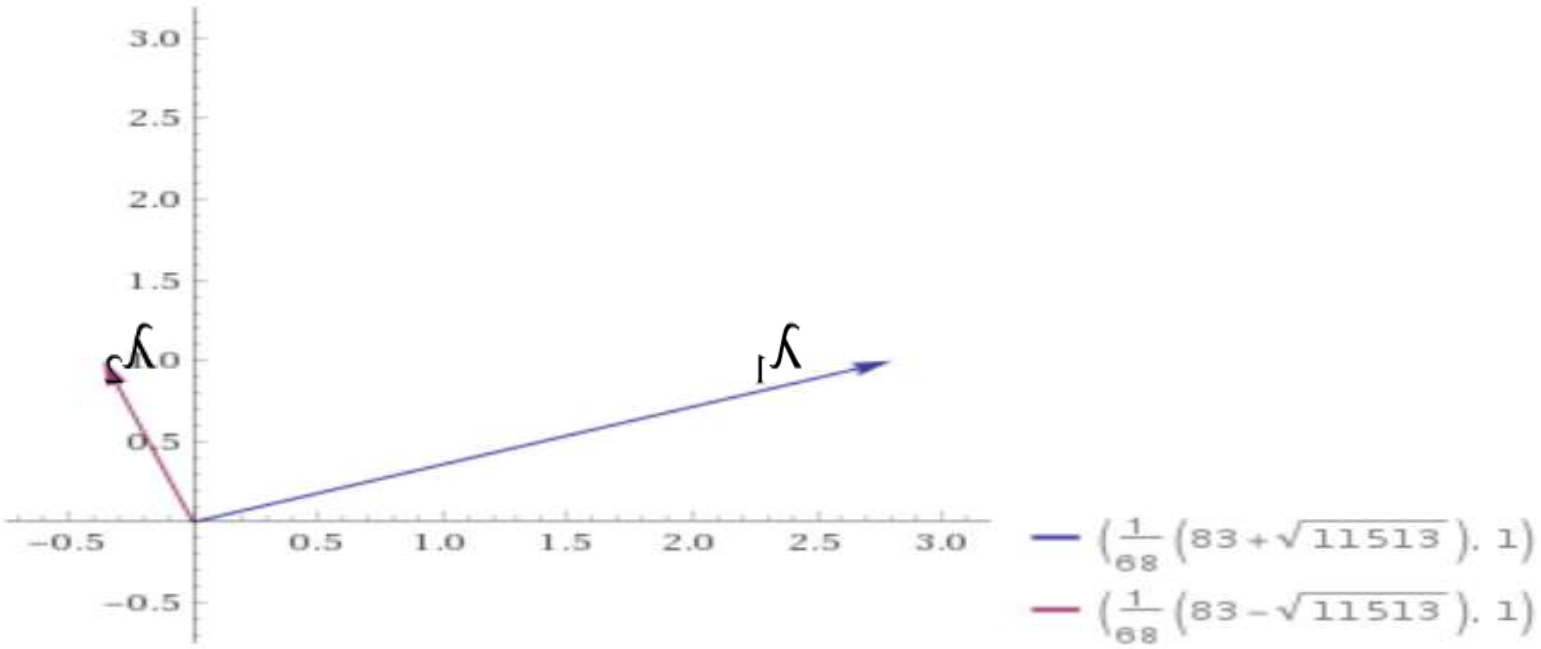
$$\lambda_2 = -\frac{3}{2}\left(\sqrt{11513} - 125\right)$$

One root or solution is: 258.888

and the other root or solution is: 116.112

Eigen Values (length of the vector)

Plot of eigenvectors:



Eigen Vectors

312.0139	102.625
102.625	63.85417

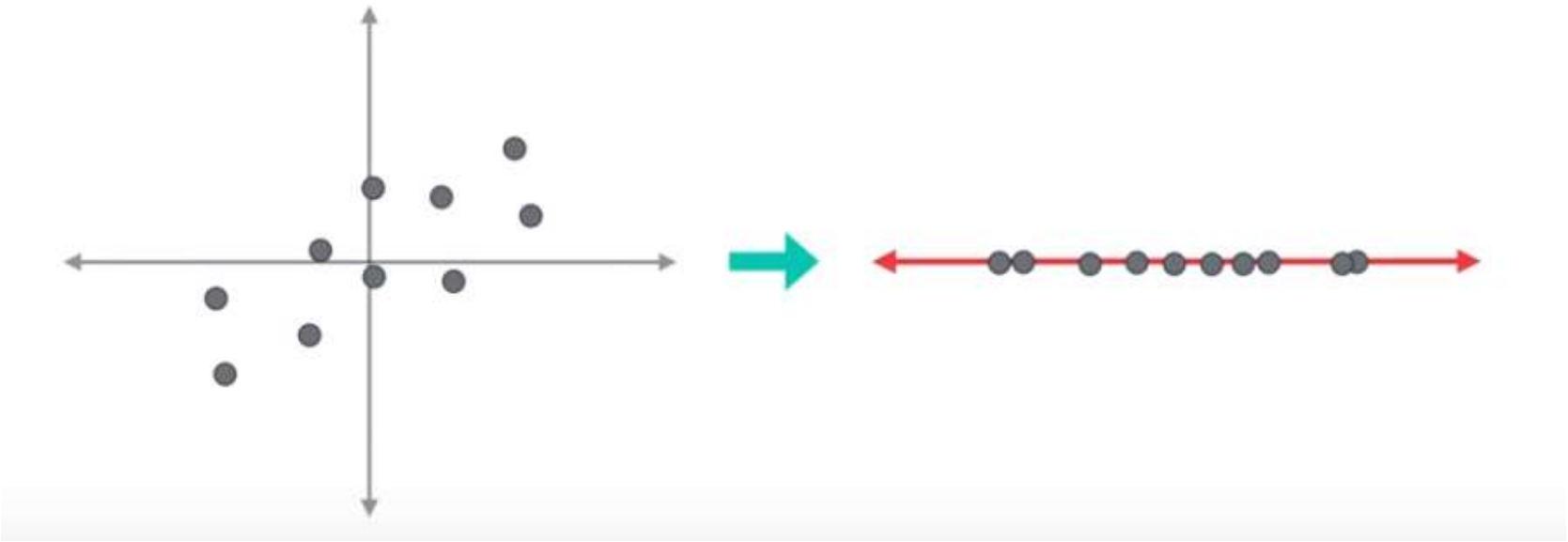
$*V = 258.88*V$

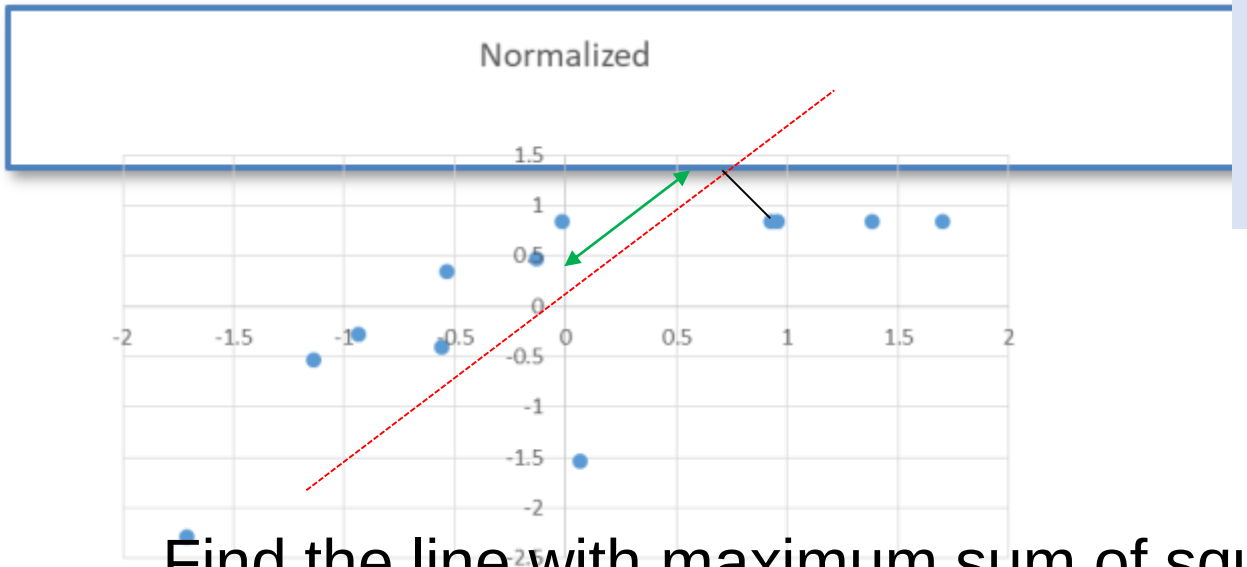
312.0139	102.625
102.625	63.85417

$*V = 116.112*V$

$$V1 \begin{vmatrix} 2.79 \\ 1 \end{vmatrix} \qquad V2 \begin{vmatrix} -0.35 \\ 1 \end{vmatrix}$$

Before transformation After





PCA1 is a linear combination of Exam1 and Exam2

Find the line with maximum sum of squares distances
Projection of samples on the red dotted line =PCA 1

PCA1= a*Exam1 +b*Exam2

(a>b =➔PCA1 more represented by the x axis)
PCA1=4Exam1+2Exam2
Pca1^2= 4^2 +2^2
➔exam1 is more influential than exam2 on data spread

Scaling

The Eigen vector of PCA1
Is this singular vector

Pca1^2= 4^2 +2^2
PCA1= SQRT(4^2 +2^2)
=SQRT(20) = 4.47

➔Length of the

PCA

4

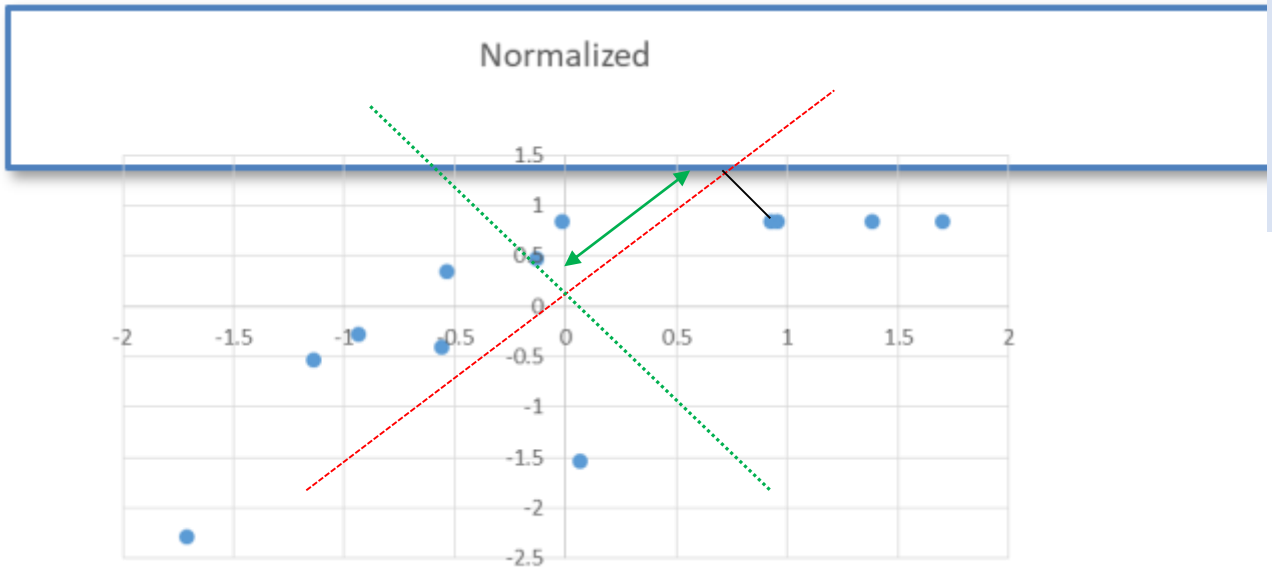
SCALE TO GET PCA=1 →DIVIDE BY 4.47

1= a^2 +b^2
a=4/4.47=0.89
b=2/4.47=0.47

decomposition of
0.89 %of Exam1 and
0.47% of Exam 2
As loading scores

SS(distances for PC1) = Eigenvalue for PC1

$\sqrt{\text{Eigenvalue for PC1}} = \text{Singular Value for PC1}$



PCA2 is a linear combination of Exam1 and Exam2

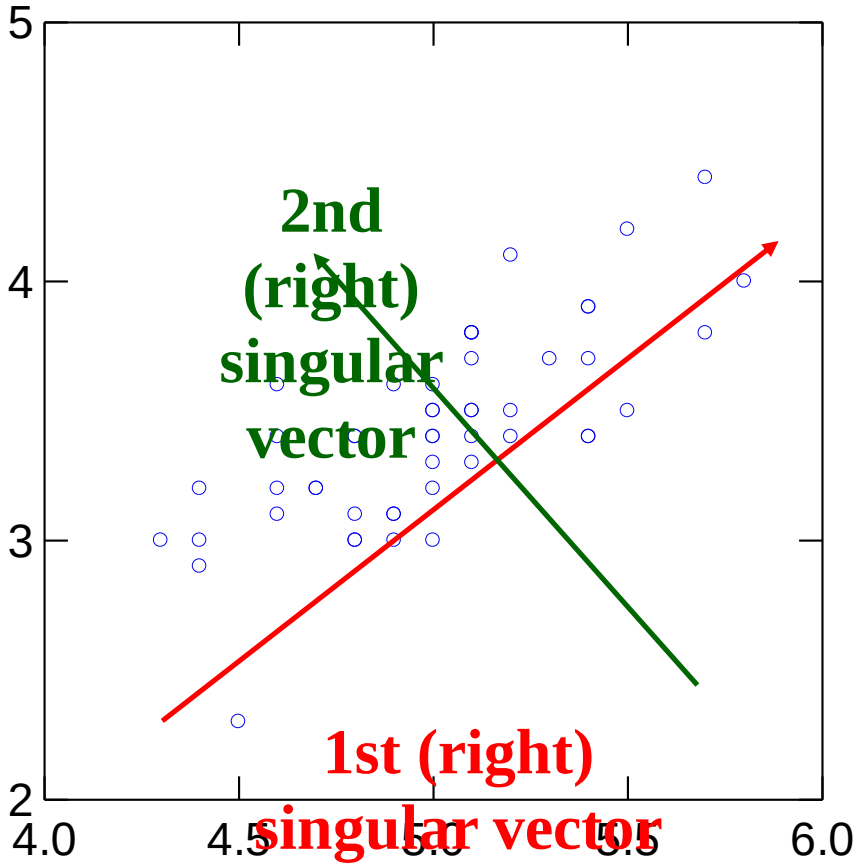
PCA2 is Orthogonal to PCA1

Projection of samples on the green dotted line =PCA 2

PCA2= -0.47*Exam1 +0.89*Exam2

➔Exam2 is more influential than exam1 when data is projected on PCA2

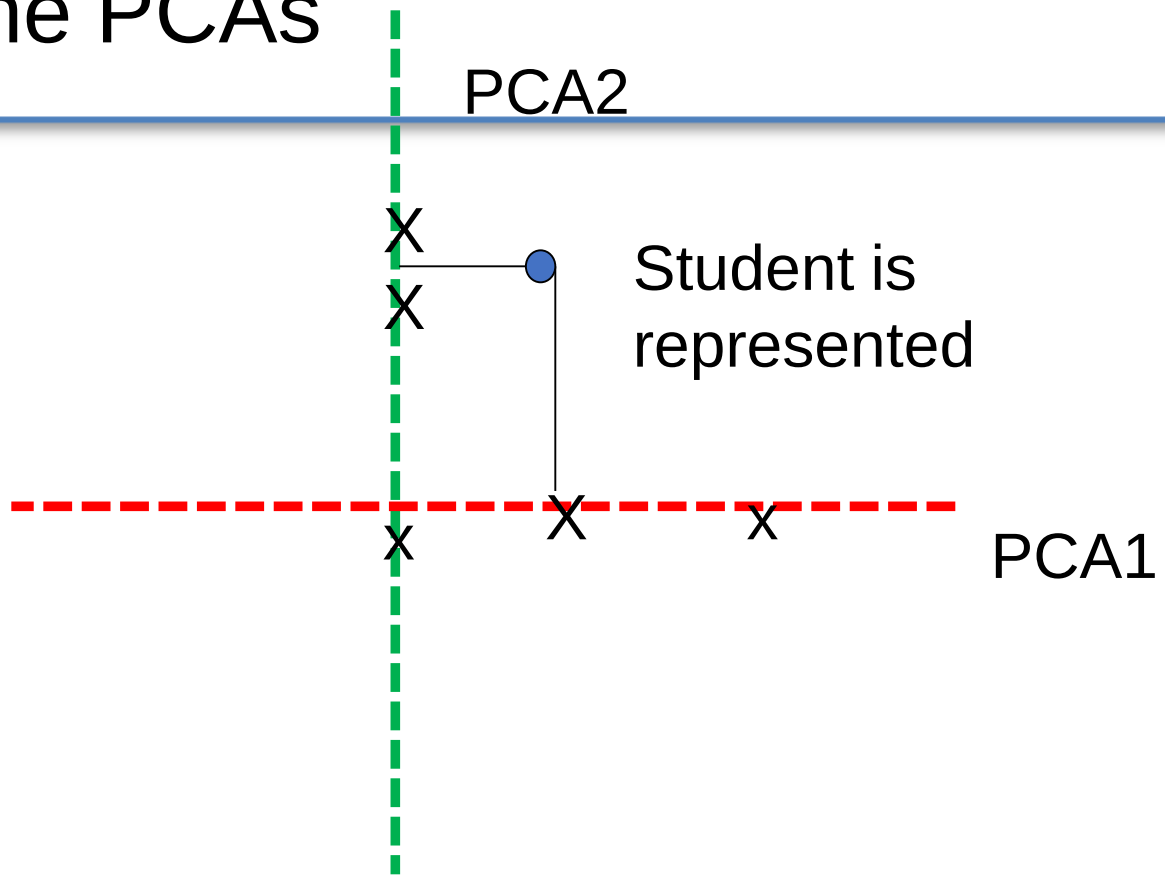
PCA is a special case of SVD on the centered covariance matrix.



PCA1= a*Exam1 +b*Exam2

PCA2= -0.47*Exam1 +0.89*Exam2

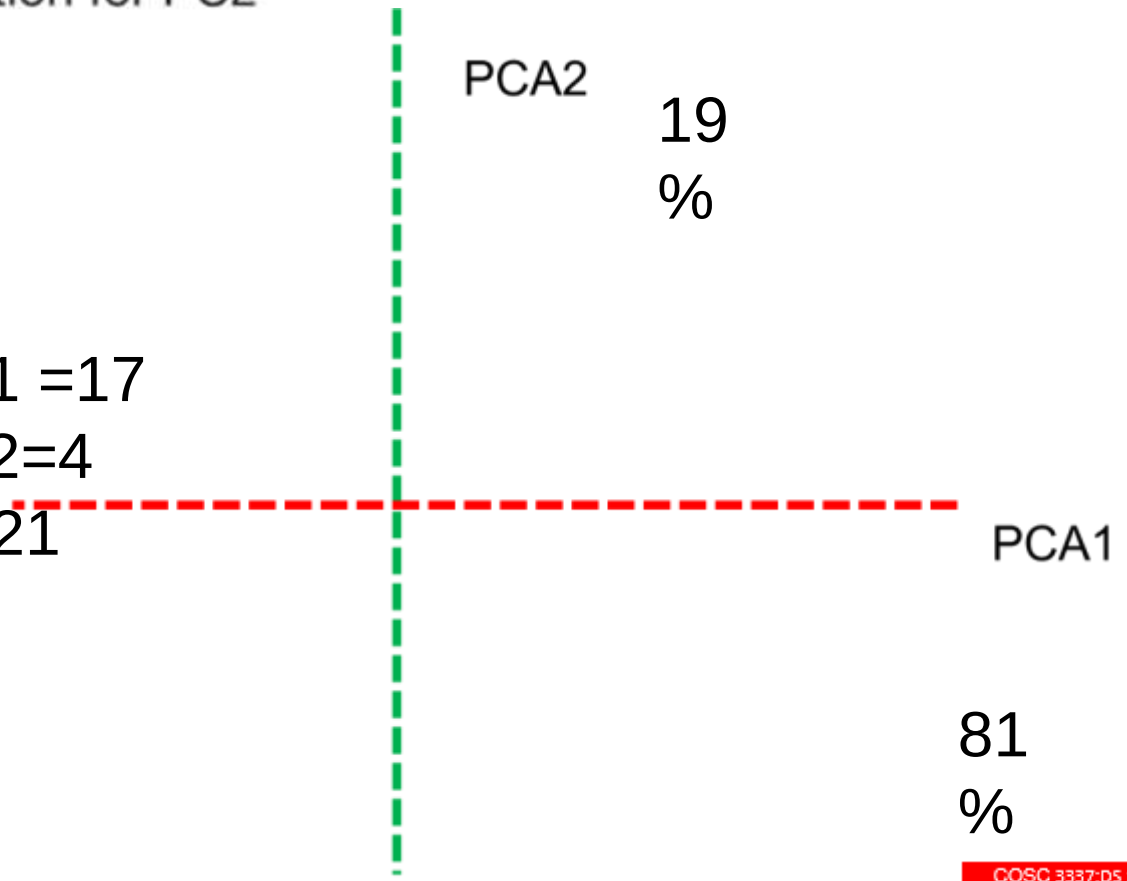
Rotate the PCAs



Variation

$$\frac{SS(\text{distances for PC1})}{n-1} = \text{Variation for PC1}$$

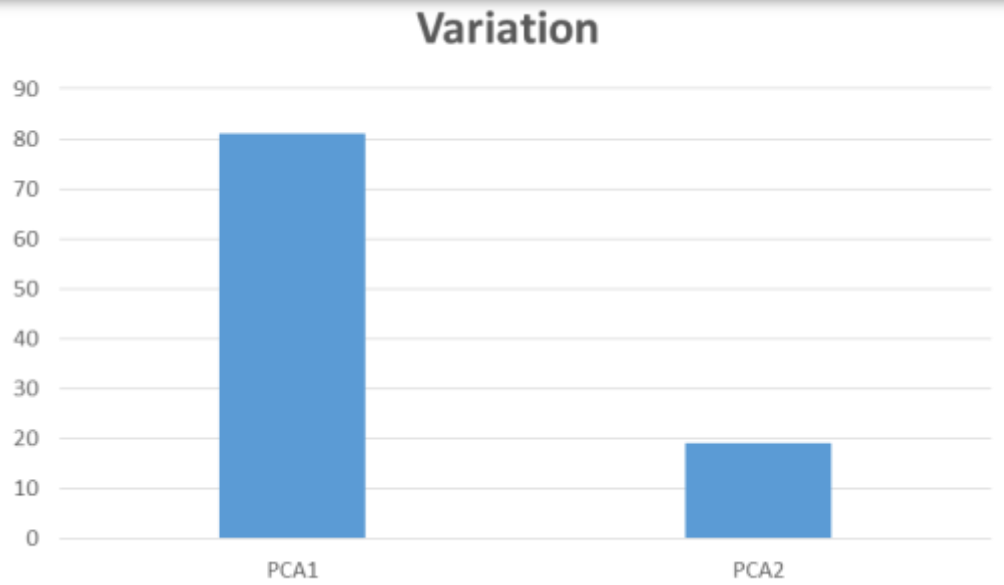
$$\frac{SS(\text{distances for PC2})}{n-1} = \text{Variation for PC2}$$



If variation of PCA1 =17
variation of PCA2=4
➔total variation 21

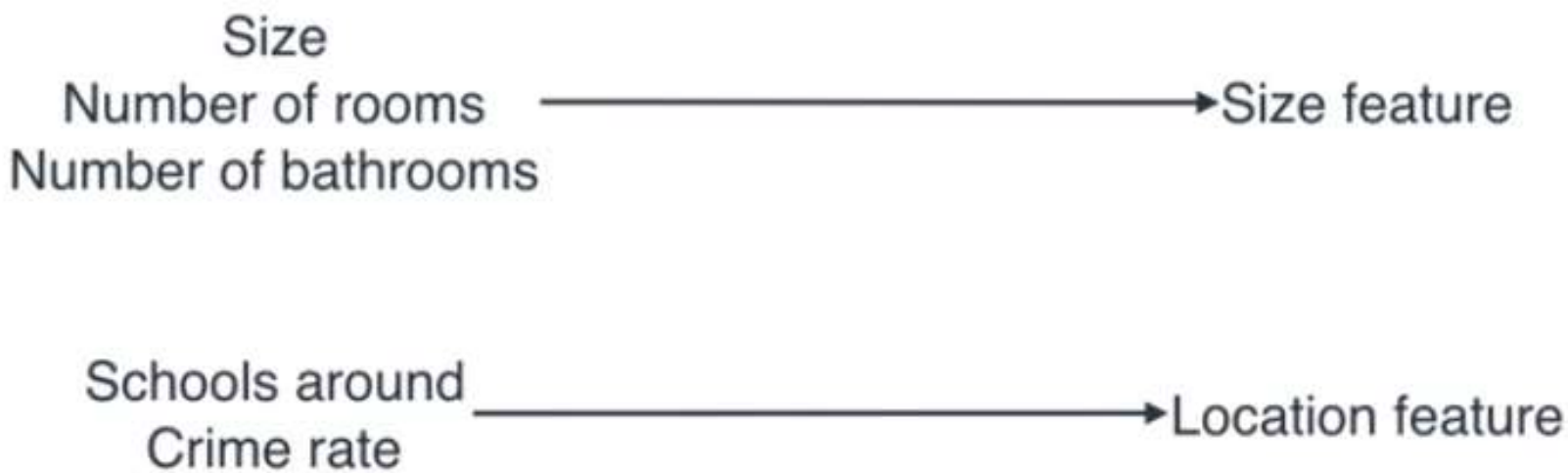
17/21=0.81
4/21= 0.19

Scree Plot

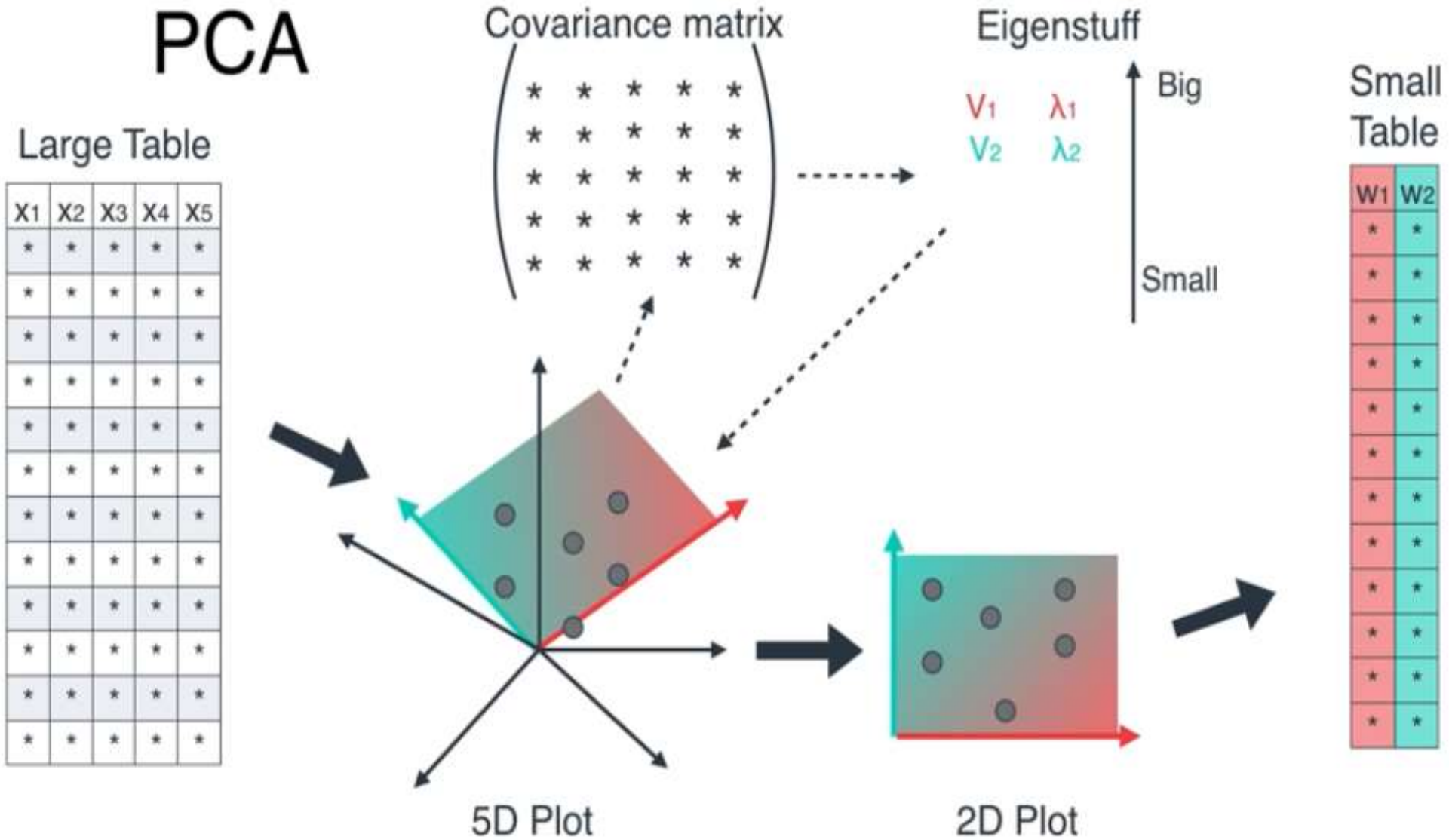


➔Exam1 is an important feature in the data set

Housing Data

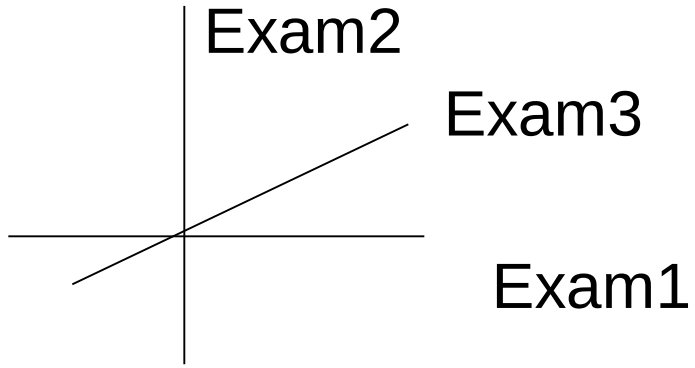


PCA


Large Table
X1 X2 X3 X4 X5
Covariance matrix
Eigenstuff
V1 λ1
V2 λ2
Big
Small
Small Table
W1 W2
5D Plot
2D Plot

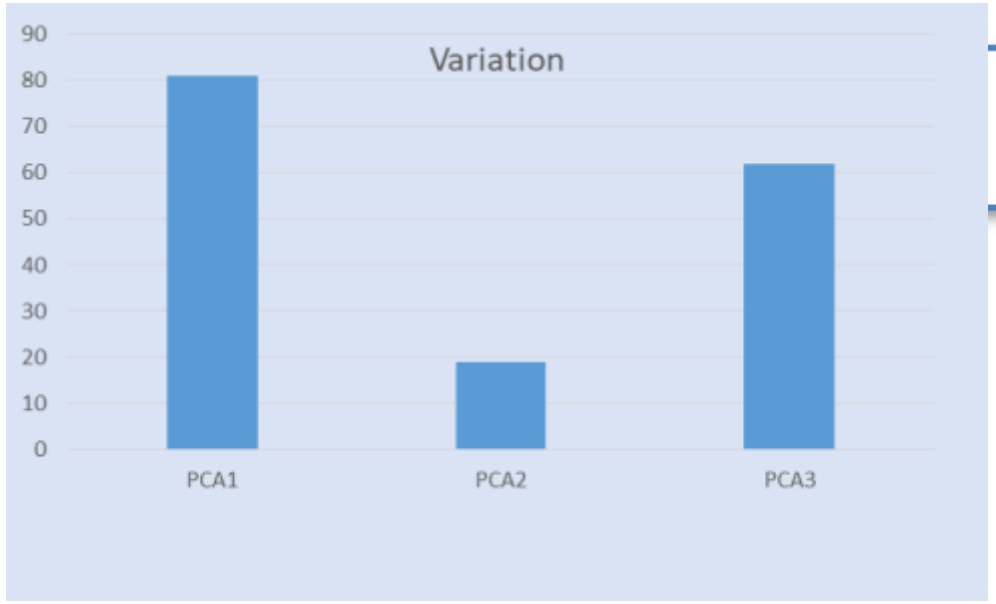
Adding a 3rd dimension ……

Student ID	Exam1	Exam2	Exam 3
Student 1	90.5	100	92
Student 2	44.5	91	72
Student 3	41	89	40
Student 4	31	75	29
Student 5	58.5	97	75
Student 6	77.5	100	89
Student 7	85	100	99
Student 8	51	90	44
Student 9	60.5	100	69
Student 10	62	81	40
Student 11	77	100	72
Student 12	51.5	96	74

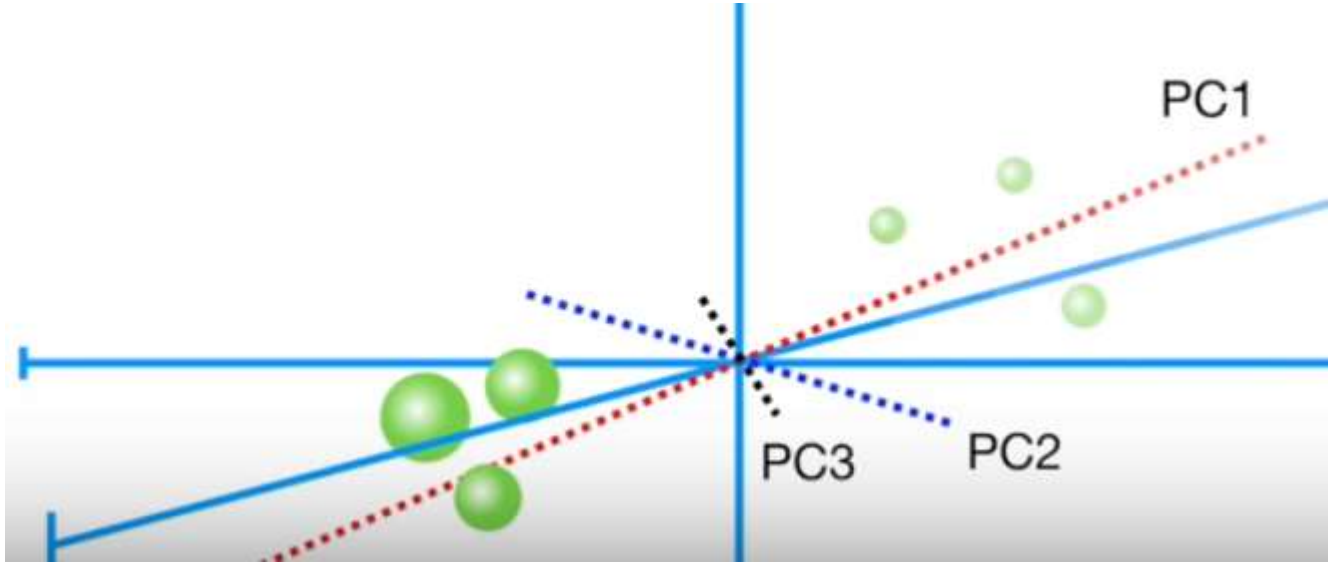


Repeat all the steps of normalization finding pca3 orthogonal to the other

=>PCA3= a*Exam1 +b*Exam2 +c*Exam3



Using a 2D-graph of PCA1 and PCA3 is a good approximation of the whole data



Compute the Eigen values (sum of squares)to find out the proportion of the var of each component

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
%matplotlib inline
```

In [ ]:
```
#load iris
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
df = pd.read_csv(url
                 , names=['sepal length','sepal width','petal length','petal width','target'])
```

In [ ]:
```
df.head()
```

In [ ]:
```
#Standardize the Data mean 0 variance 1
features = ['sepal length', 'sepal width', 'petal length', 'petal width']
x = df.loc[:, features].values
```

In [ ]:
```
 y= df.loc[:,['target']].values
```

In [ ]:
```
x = StandardScaler().fit_transform(x)
```

In [ ]:
```
#print data after standarization
pd.DataFrame(data = x, columns = features).head()
```

In [ ]:
```
pca = PCA(n_components=2)
```

In [ ]:
```
principalComponents = pca.fit_transform(x)
```

In [ ]:
```
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
```

In [ ]:
```
principalDf.head(5)
```

In [ ]:
```
df[['target']].head()
```

In [ ]:
```
finalDf = pd.concat([principalDf, df[['target']]], axis = 1)
finalDf.head(5)
```

In [ ]:
```
#Visualize 2D Projection
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 Component PCA', fontsize = 20)


targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['target'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```

In [ ]:
```
#The explained variance tells us how much information (variance) can be attributed to each of the principal components
```

In [ ]:
```
pca.explained_variance_ratio_
```

In [ ]:
```
#Together, the first two principal components contain 95.80% of the information.
#The first principal component contains 72.77% of the variance and the second principal component contains 23.03% of the variance
#The third and fourth principal component contained the rest of the variance of the dataset.
```

PCA to Speed-up Machine Learning Algorithms

- Using the IRIS dataset would be impractical here as the dataset only has 150 rows and only 4 feature columns. The MNIST database of handwritten digits is more suitable as it has 784 feature columns (784 dimensions), a training set of 60,000 examples, and a test set of 10,000 examples.

In [ ]:
```
#handwritten images
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
```

In [ ]:
```
#split data
from sklearn.model_selection import train_test_split
# test_size: what proportion of original data is used for test set
train_img, test_img, train_lbl, test_lbl = train_test_split( mnist.data, mnist.target, test_size=1/7.0, random_state=0)
```

In [ ]:
```
#standardize data
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit on training set only.
scaler.fit(train_img)
# Apply transform to both the training set and the test set.
train_img = scaler.transform(train_img)
test_img = scaler.transform(test_img)
```

In [ ]:
```
#import and apply PCA
from sklearn.decomposition import PCA
# Make an instance of the Model
pca = PCA(.95)
```

In [ ]:
```
#Fit PCA on training set
pca.fit(train_img)
```

In [ ]:
```
#Apply the mapping (transform) to both the training set and the test set.
train_img = pca.transform(train_img)
test_img = pca.transform(test_img)
```

In [ ]:
```
#Apply Logistic Regression to the Transformed Data
#Step 1: Import the model you want to use
from sklearn.linear_model import LogisticRegression
```

In [ ]:
```
#Step 2: Make an instance of the Model.
# all parameters not specified are set to their defaults
# default solver is incredibly slow which is why it was changed to 'lbfgs'
logisticRegr = LogisticRegression(solver = 'lbfgs')
```

In [ ]:
```
#Step 3: Training the model on the data, storing the information learned from the data

#Model is learning the relationship between digits and labels
logisticRegr.fit(train_img, train_lbl)
```

In [ ]:
```
#Step 4: Predict the labels of new data (new images)
# Predict for One Observation (image)
logisticRegr.predict(test_img[0].reshape(1,-1))
```

In [ ]:
```
# Predict for One Observation (image)
logisticRegr.predict(test_img[0:10])
```

In [ ]:
```
#Measuring Model Performance
logisticRegr.score(test_img, test_lbl)
print(score)
```

In [ ]:
```
#Number of Components, Variance, Time Table
pd.DataFrame(data = [[1.00, 784, 48.94, .9158],
                     [.99, 541, 34.69, .9169],
                     [.95, 330, 13.89, .92],
                     [.90, 236, 10.56, .9168],
                     [.85, 184, 8.85, .9156]],
             columns = ['Variance Retained',
                        'Number of Components',
                        'Time (seconds)',
                        'Accuracy'])
```