Lecture 6. PCA. Principal Component Analysis

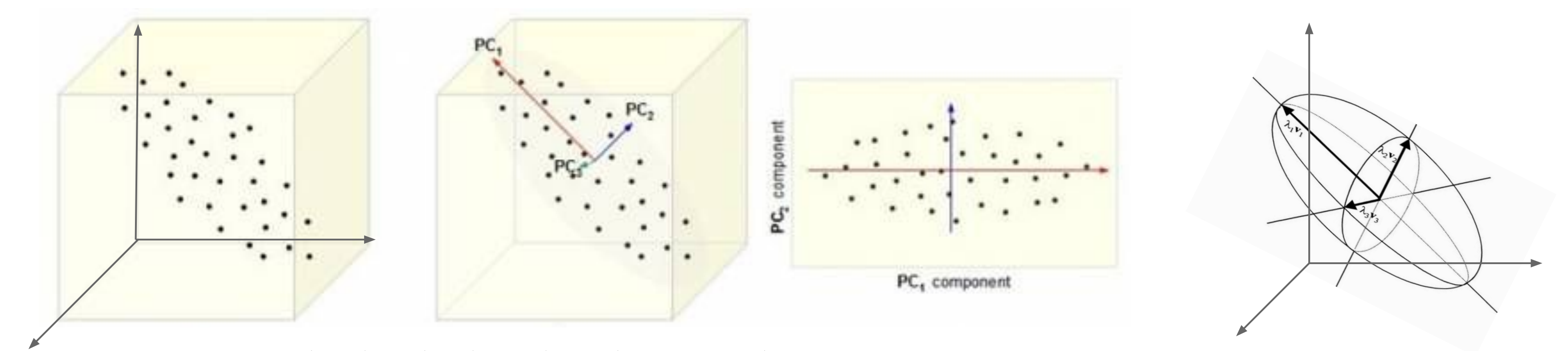
Suthep “Jogie” Madarasmi, Ph.D.

PCA. Principal Component Analysis.

Goal: Reduce the dimensionality of a data set while retaining the variation ("information content") in it. Convert the dataset into new features to reduce covariance. Highly covariant data example: height & weight, height & foot size, GPA & hard working, …

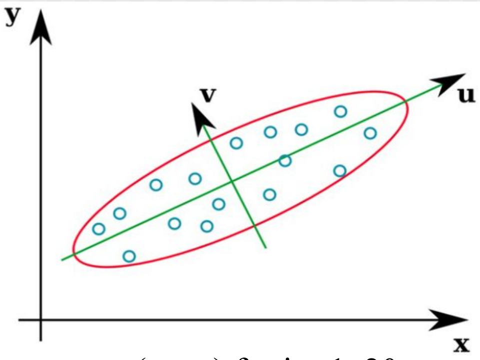
1. Coordinate transformation to fit an ellipsoid. Translate origin to data centroid. Find a new orthonormal coordinate axis to fit an ellipsoid.

2. Eliminate the dimension(s) with the smaller ellipsoid axis.



Example of Reduction of 3D data to 2D data

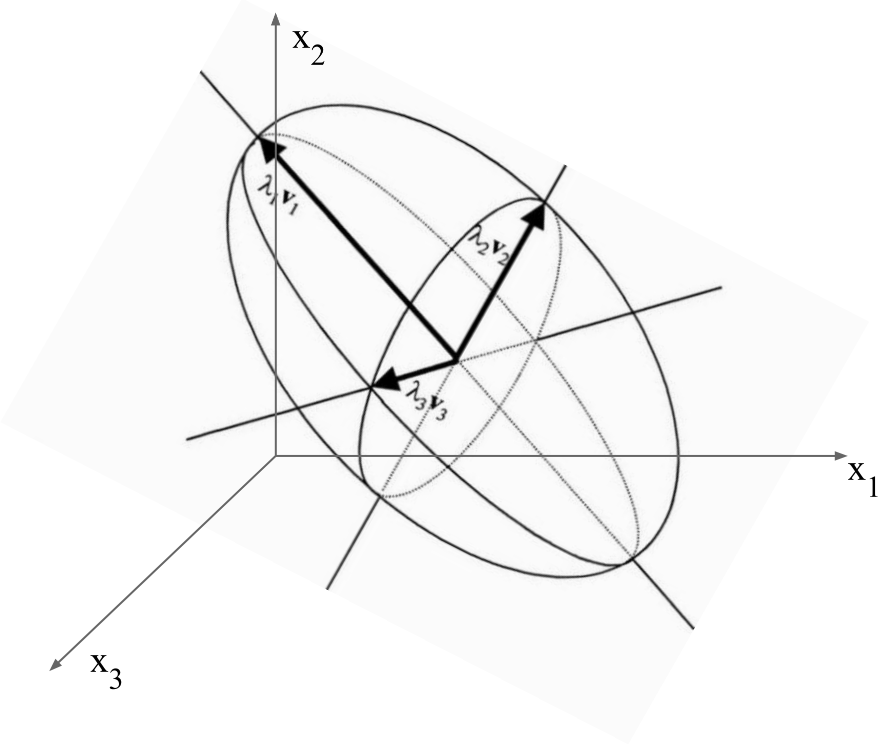
Step 1. Coordinate change by Translation + Rotation



Given a dataset of feature vectors (x_i, y_i) for i = 1..20.

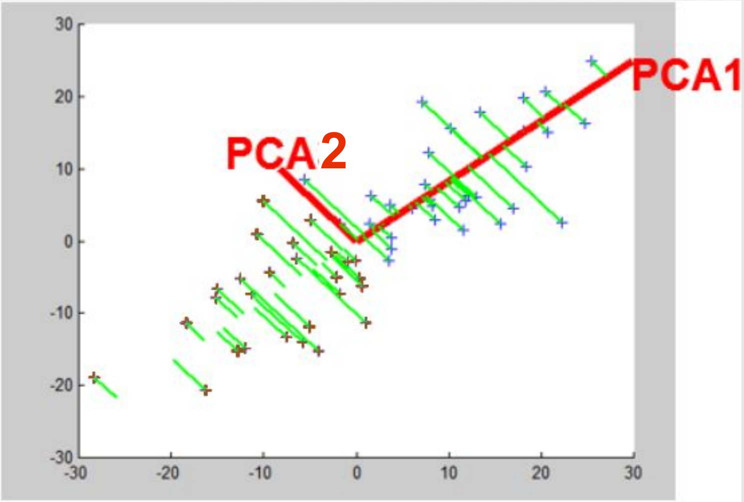
How to find new feature vectors (u_i, v_i)?

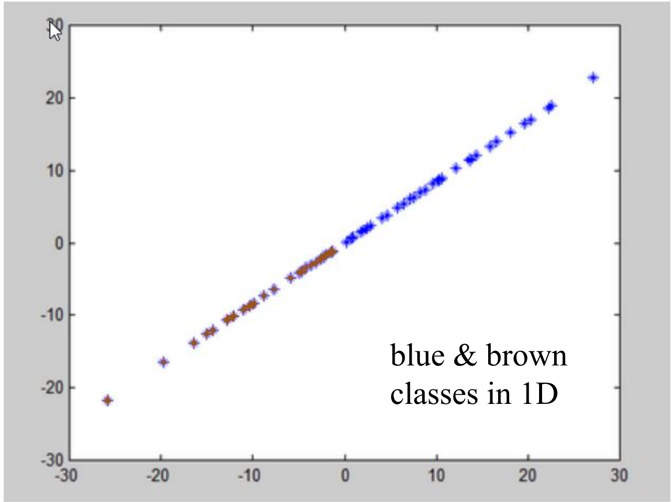
1. Translate to a new origin by subtracting mean (x_{av}, y_{av}) of the dataset. $(x_i, y_i) - (x_{av}, y_{av}) \Rightarrow (x'_i, y'_i)$
2. Find the covariance matrix **C** for the new dataset (x'_i, y'_i).
3. Find Rotation matrix R for the ellipse by finding Eigenvalues and Eigenvectors of **C**. Rotate each (x'_i, y'_i) to (u, v) using eigenvectors as the R.



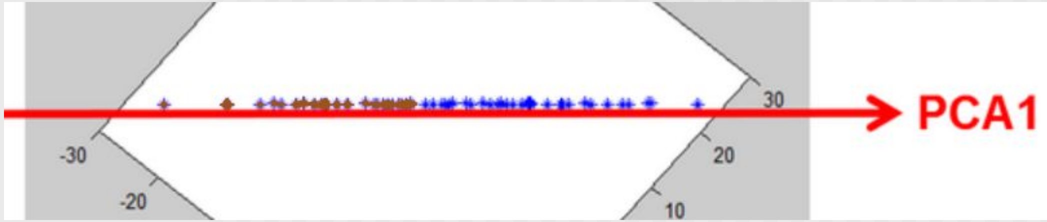
Step 2. Remove unimportant dimensions.

How? Keep dimensions (eigenvectors) with highest Eigenvalues to preserve information content.





After dimension reduction to preserve enough variance, the dataset is generally still separable.



The PCA1 axis is the eigenvector with highest eigenvalue becomes the new x-axis for the transformed 2D to 1D feature vectors.

The Iris Flower Dataset: *setosa, versicolor, virginica.*

	sepal length	sepal width	petal length	petal width
0	-0.900681	1.032057	-1.341272	-1.312977
1	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.385353	0.337848	-1.398138	-1.312977
3	-1.506521	0.106445	-1.284407	-1.312977
4	-1.021849	1.263460	-1.341272	-1.312977

(x_1, x_2, x_3, x_4)

4D to 2D PCA

	principal component 1	principal component 2
0	-2.264542	0.505704
1	-2.086426	-0.655405
2	-2.367950	-0.318477
3	-2.304197	-0.575368
4	-2.388777	0.674767

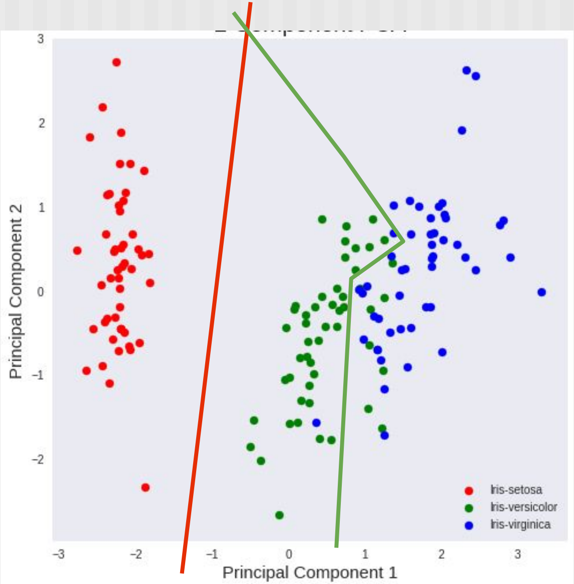
(x_1', x_2')

	target
0	Iris-setosa
1	Iris-setosa
2	Iris-setosa
3	Iris-setosa
4	Iris-setosa

Even after dimension reduction from 4D to 2D, the 3 Iris classes still appear to be separable. The red *setosa* is easily separable. The blue *virginica* and green *versicolor* are starting to become more difficult to separate, but it's still reasonably easy.

Dimension reduction can:

- **help visualization**
- **speed up the learning algorithm**
- **save memory and disk space.**

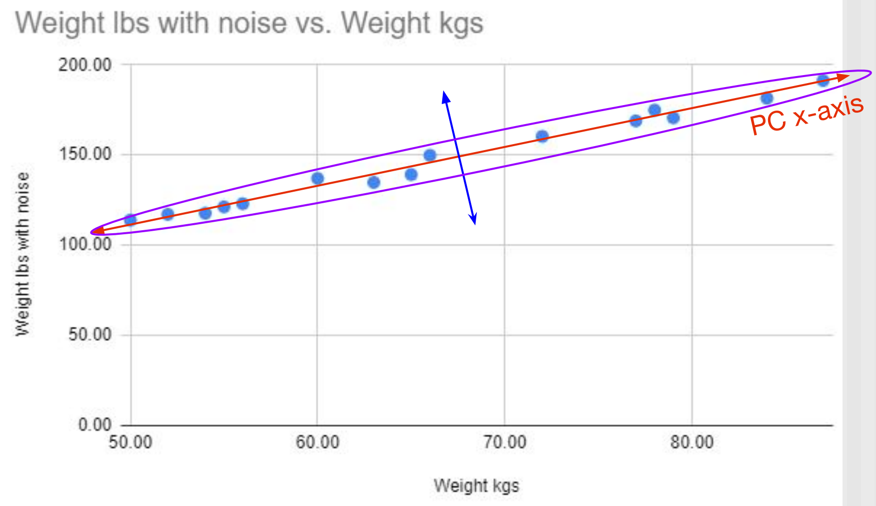


Extreme Data Correlation to help understand

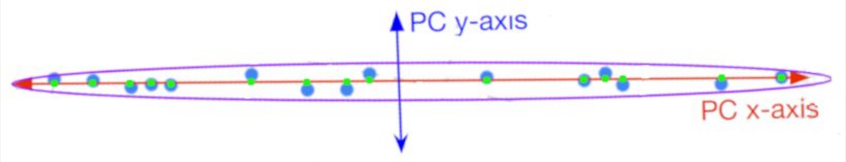
Highly (fully) correlated feature: (Weight in kgs, Weight in lbs). Some measurement noise.

`=B4*2.20462 +(rand() - 0.5)*10`

Weight kgs	Weight in lbs	Weight lbs with noise
56.00	123.46	123.02
54.00	119.05	117.71
72.00	158.73	160.35
84.00	185.19	181.56
79.00	174.16	170.66
65.00	143.30	139.17
52.00	114.64	117.12
55.00	121.25	121.29
87.00	191.80	191.36
63.00	138.89	134.85
78.00	171.96	174.96
66.00	145.50	149.84
77.00	169.76	168.96
60.00	132.28	137.05
50.00	110.23	113.95



The new “PC” coordinates of a fitting ellipse is shown. The PC y-axis is very small and can be eliminated without much loss of information. Each feature will have a new value along the PC x-axis shown in green.



General Case

Features are highly correlated if one can predict the other. Examples of highly correlated:

- height and weight. Weight can quite accurately be determined by (height, waist size)
- ethnicity and skin color
- blurry image and unclear image. Processing cost = a*blurry + b*unclear + c*size + d*featureless + e

Fitting becomes a problem when features are highly correlated, so parameters keep changing:

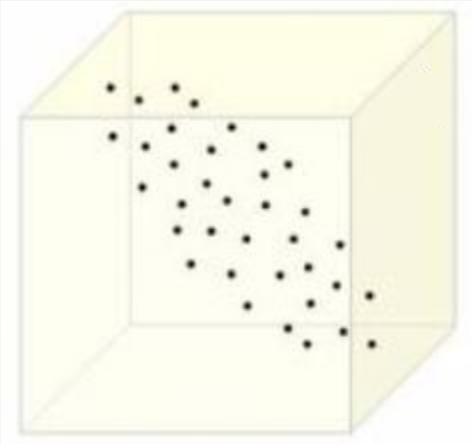
Cost = 2.0*blurry + 1.0*unclear + 0.5*size + 3.0*featureless + 1.5

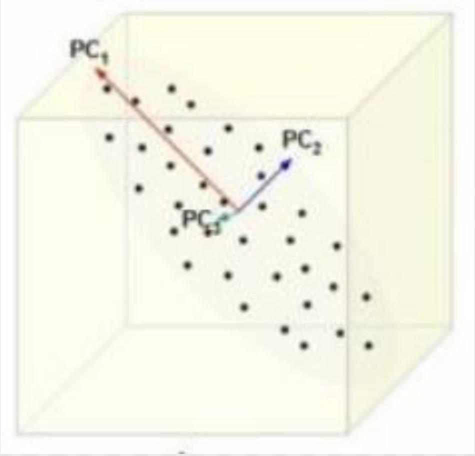
Cost = 2.8*blurry + 0.2*unclear + 0.5*size + 3.0*featureless + 1.5

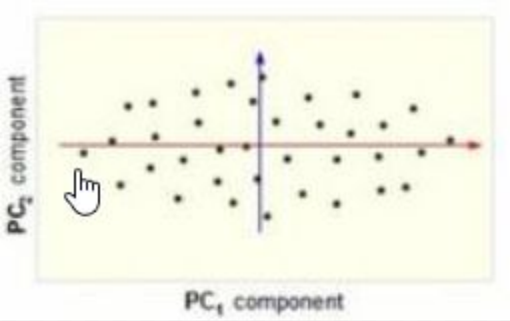
Cost = 4.0*blurry - 1.0*unclear + 0.5*size + 3.0*featureless + 1.5

Thus, it's a good idea to reduce the number of features to only those that are not highly correlated via PCA. Dimensional reduction to 2 or 3 dimensions can also help visualize the dataset.

Example 3D to 2D dimension reduction







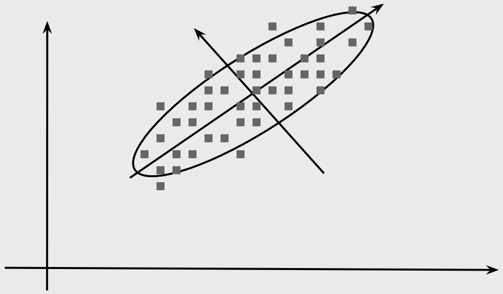
3D feature such as (x, y, z) = (length, grayscale color, weight)

New 3D feature such as where each (x’, y’, z’) are linear combinations of the original (x, y, z).

Only (x’, y’) dimensions of the 2 largest ellipse axis are kept with z’ removed. They contain more information.

Principal Component Analysis - PCA

- Procedure to transform an observation data into a value data with new vector space by creating a new coordinate system.
- PCA: direction with maximum variance. All are orthogonal.
- Eigenvectors show the direction of axes
- The larger the Eigenvalue, the more significant the Eigenvector axis
- Reduce the dimension of data by choosing the significant Eigenvectors

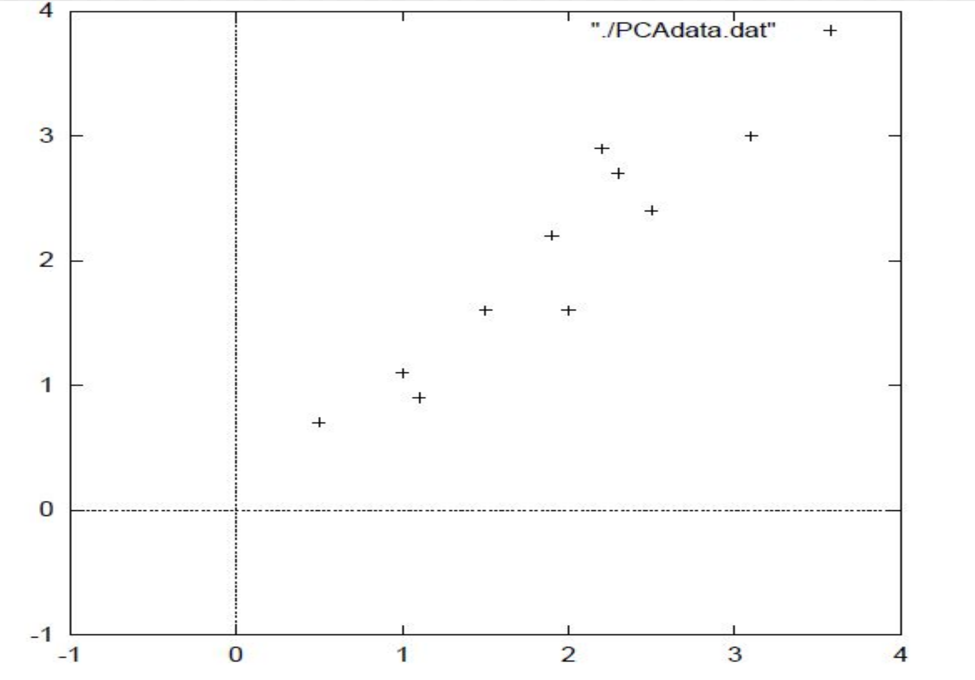


PCA Method (1). Original data.

Step 1. Start with Original Data. Here 2-D. Will reduce it to 1-D.

$\mathbf{x}_j \rightarrow \boldsymbol{\phi}_j \rightarrow \boldsymbol{\phi}_j'$

x	y
2.50	2.40
0.50	0.70
2.20	2.90
1.90	2.20
3.10	3.00
2.30	2.70
2.00	1.60
1.00	1.10
1.50	1.60
1.10	0.90



PCA Method (2). Translate the origin to data centroid.

Step 2. Subtract the mean to get:

- $(x'_i, y'_i) = (x - x_\mu,\ y - y_\mu)$
- Means: $x_\mu = 1.81$, $y_\mu = 1.91$.

$\mathbf{x}_j \rightarrow \boldsymbol{\phi}_j \rightarrow \boldsymbol{\phi}_j'$

x	y	$x' = x - x_\mu$	$y' = y - y_\mu$
2.50	2.40	0.69	0.49
0.50	0.70	-1.31	-1.21
2.20	2.90	0.39	0.99
1.90	2.20	0.09	0.29
3.10	3.00	1.29	1.09
2.30	2.70	0.49	0.79
2.00	1.60	0.19	-0.31
1.00	1.10	-0.81	-0.81
1.50	1.60	-0.31	-0.31
1.10	0.90	-0.71	-1.01
1.81	1.91	←Mean	

PCA Method (3). Find Covariance matrix.

Step 3: Calculate the covariance matrix for (x', y'):

Shorthand matrix notation for C: $$C = \frac{1}{n}\sum_{i=1}^{n}\phi_i\phi_i^T = \frac{1}{n}\sum_{i=1}^{n}\begin{bmatrix} x_i' \\ y_i' \\ z_i' \end{bmatrix}\begin{bmatrix} x_i' & y_i' & z_i' \end{bmatrix}$$

$$C(r,\, c) = \frac{1}{n}\sum_{i=1}^{n}(r_i - r_\mu)(c_i - c_\mu)$$

Cov(r, c)	x	y	z
x	Cov(x, x)	Cov(x, y)	Cov(x, z)
y	Cov(y, x)	Cov(y, y)	Cov(y, z)
z	Cov(z, x)	Cov(z, y)	Cov(z, z)

- Covariance can analyze relationships between pairwise feature dimensions.
- **Ex**: In 3-D data set (x,y,z), we will calculate Cov(x,y), Cov(x,z), Cov(y,z) , Cov(x,x) = Var(x), Cov(y,y) = Var(y), Cov(z,z) = Var(z).
- **Note:** Cov(a, b) = Cov(b, a); Cov(a, a) = Var(a).

Covariance in matrix notation

In Matrix notation, for n data points each feature after subtracting feature mean is $\phi_i = (x'_i, y'_i)$ which is, say, 2x1. This when multiplied by its transpose which is 1x2 results in a 2x2 matrix.

$$C = \frac{1}{n}\sum_{i=1}^{n}\phi_i\phi^T = \frac{1}{n}\sum_{i=1}^{n}\begin{bmatrix} x'_i \\ y'_i \end{bmatrix}\begin{bmatrix} x'_i & y'_i \end{bmatrix} = \frac{1}{n}\sum_{i=1}^{n}\begin{bmatrix} x_i'^2 & x'_i y'_i \\ x'_i y'_i & y_i'^2 \end{bmatrix} = \begin{bmatrix} \sum_i^n x_i'^2 & \sum_i^n x'_i y'_i \\ \sum_i^n x'_i y'_i & \sum_i^n y_i'^2 \end{bmatrix}$$

$$= \begin{bmatrix} \sum_i^n (x_i - x_\mu)^2 & \sum_i^n (x_i - x_\mu)(y_i - y_\mu) \\ \sum_i^n (x_i - x_\mu)(y_i - y_\mu) & \sum_i^n (y_i - y_\mu)^2 \end{bmatrix}$$

$$C(r, c) = \frac{1}{n}\sum_{i=1}^{n}(r_i - r_\mu)(c_i - c_\mu)$$

Find the Covariance Matrix

$$C = \frac{1}{n}\sum_{i=1}^{n}\phi_i\phi^T; \qquad \text{Equivalently:} \quad C(r,\, c) = \frac{1}{n}\sum_{i=1}^{n}(r_i - r_\mu)(c_i - c_\mu)$$

Example of the covariance matrix for 6-D features:

6-D Features	F1	F2	F3	F4	F5	F6
F1	Cov(F1, F1)	Cov(F1, F2)	Cov(F1, F3)	Cov(F1,F4)	Cov(F1,F5)	Cov(F1,F6)
F2	Copy	Cov(F2, F2)	Cov(F2, F3)	Cov(F2,F4)	Cov(F2,F5)	Cov(F2,F6)
F3	Copy	Copy	Cov(F3, F3)	Cov(F3,F4)	Cov(F3,F5)	Cov(F3,F6)
F4	Copy	Copy	Copy	Cov(F4,F4)	Cov(F4,F5)	Cov(F4,F6)
F5	Copy	Copy	Copy	Copy	Cov(F5,F5)	Cov(F5,F6)
F6	Copy	Copy	Copy	Copy	Copy	Cov(F6,F6)

Find the Covariance Matrix

$\mathbf{x}_j \rightarrow \boldsymbol{\phi}_j \rightarrow \boldsymbol{\phi}_j'$

6-D Features	F1	F2	F3	F4	F5	F6
F1	Cov(F1, F1)	Cov(F1, F2)	Cov(F1, F3)	Cov(F1,F4)	Cov(F1,F5)	Cov(F1,F6)
F2	Copy	Cov(F2, F2)	Cov(F2, F3)	Cov(F2,F4)	Cov(F2,F5)	Cov(F2,F6)
F3	Copy	Copy	Cov(F3, F3)	Cov(F3,F4)	Cov(F3,F5)	Cov(F3,F6)
F4	Copy	Copy	Copy	Cov(F4,F4)	Cov(F4,F5)	Cov(F4,F6)
F5	Copy	Copy	Copy	Copy	Cov(F5,F5)	Cov(F5,F6)
F6	Copy	Copy	Copy	Copy	Copy	Cov(F6,F6)

The **goal of PCA** is to transform each data point $\mathbf{x}_j$ to $\boldsymbol{\phi}_j'$ so that the covariance matrix for the transformed data is a diagonal matrix with variances: λ^2 and covariances: 0. Note λ is n-dimensional where λ_i is the i-th dimension variance.

Find Covariance matrix

Step 3. Calculate the covariance matrix for (x’, y’):

x	y	x'	y'	x' * x'	x' * y'	y' * x'	y' * y'
2.50	2.40	0.69	0.49	0.48	0.34	0.34	0.24
0.50	0.70	-1.31	-1.21	1.72	1.59	1.59	1.46
2.20	2.90	0.39	0.99	0.15	0.39	0.39	0.98
1.90	2.20	0.09	0.29	0.01	0.03	0.03	0.08
3.10	3.00	1.29	1.09	1.66	1.41	1.41	1.19
2.30	2.70	0.49	0.79	0.24	0.39	0.39	0.62
2.00	1.60	0.19	-0.31	0.04	-0.06	-0.06	0.10
1.00	1.10	-0.81	-0.81	0.66	0.66	0.66	0.66
1.50	1.60	-0.31	-0.31	0.10	0.10	0.10	0.10
1.10	0.90	-0.71	-1.01	0.50	0.72	0.72	1.02
1.81	1.91	←Mean	Mean →	0.555	0.554	0.554	0.645

2 Features	**x**	**y**
x	Cov(x, x)	Cov(x, y)
y	Cov(x,y)	Cov(y, y)

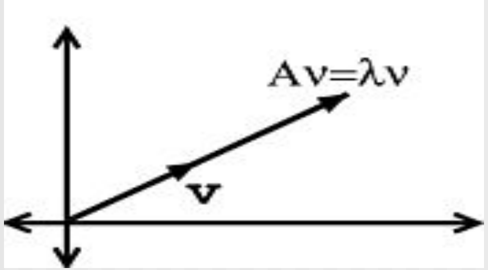
$$C = \begin{bmatrix} 0.555 & 0.554 \\ 0.554 & 0.645 \end{bmatrix}$$

$$C(r, c) = \frac{1}{n}\sum_{i=1}^{n}(r_i - r_\mu)(c_i - c_\mu)$$

PCA Method (5). Find Eigen λ, v of Cov matrix

Step 4. Find the eigenvectors $\mathbf{v}_i$, eigenvalues λ_i of the covariance matrix.

- A is an $n \times n$ matrix, A is a linear operator on vectors in R^n
- Definition Eigenvector of A is a vector $\mathbf{v} \in R^n$



$$A\mathbf{v} = \lambda\mathbf{v}$$ Note: $A_{nxn} * \mathbf{v}_{nx1} = \lambda * \mathbf{v}_{nx1}$

where λ is the eigenvalue, **v** is the eigenvector.

For *n* dimensional features, there will be:

- *n* eigenvalues
- *n* eigenvectors

Find Eigenvalues λ_i of Covariance matrix

Finding eigenvalues:

$$AX - \lambda X = 0$$

or

$$(A - \lambda I)\,X = 0$$

$$\det(\lambda I - A) = 0$$

If a transformation matrix transforms the vector **v** to zero, then its determinant must be zero. Here **v** = λI - A.

$$A = \begin{bmatrix} 5 & -10 & -5 \\ 2 & 14 & 2 \\ -4 & -8 & 6 \end{bmatrix}$$

$$\det\left(\lambda \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 5 & -10 & -5 \\ 2 & 14 & 2 \\ -4 & -8 & 6 \end{bmatrix}\right) = 0$$

$$\det \begin{bmatrix} \lambda - 5 & 10 & 5 \\ -2 & \lambda - 14 & -2 \\ 4 & 8 & \lambda - 6 \end{bmatrix} = 0$$

$$(\lambda - 5)(\lambda - 10)^2 = 0$$

$\lambda_1 = 5, \lambda_2 = 10$ and $\lambda_3 = 10.$

Find Eigenvectors v_i of Covariance matrix

Finding the unit eigenvector $||v|| = 1$ for each eigenvalue. Here for $\lambda_1 = 5$:

$$A = \begin{bmatrix} 5 & -10 & -5 \\ 2 & 14 & 2 \\ -4 & -8 & 6 \end{bmatrix} \quad \lambda_1 = 5, \lambda_2 = 10 \text{ and } \lambda_3 = 10.$$

$$\left(5 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 5 & -10 & -5 \\ 2 & 14 & 2 \\ -4 & -8 & 6 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 10 & 5 \\ -2 & -9 & -2 \\ 4 & 8 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \left[\begin{array}{ccc|c} 0 & 10 & 5 & 0 \\ -2 & -9 & -2 & 0 \\ 4 & 8 & -1 & 0 \end{array} \right] \quad \left[\begin{array}{ccc|c} 1 & 0 & -\frac{5}{4} & 0 \\ 0 & 1 & \frac{1}{2} & 0 \\ 0 & 0 & 1 & 0 \end{array} \right]$$

$$X_1 = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -\frac{5}{4} \\ \frac{1}{2} \\ 1 \end{bmatrix} = \begin{bmatrix} -5 \\ 2 \\ 4 \end{bmatrix}$$

$$||X_1|| = \sqrt{45} = 6.708$$

$$v_1 = \begin{bmatrix} -0.7454 \\ 0.2981 \\ 0.5963 \end{bmatrix}$$

Decomposition of the Covariance matrix

$$\lambda_2 = 0.04417; \qquad \lambda_1 = 1.15562$$

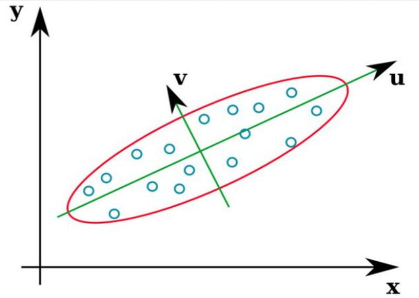
$$v_2 = \begin{bmatrix} -0.73518 \\ 0.67787 \end{bmatrix}; \quad v_1 = \begin{bmatrix} 0.67787 \\ 0.73518 \end{bmatrix}$$

$$R = \begin{bmatrix} v_1^T \\ v_2^T \end{bmatrix} = \begin{bmatrix} 0.67787 & 0.73518 \\ -0.73518 & 0.67787 \end{bmatrix}; \; R^{-1} = R^T$$

$$D = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = \begin{bmatrix} 1.15562 & 0 \\ 0 & 0.04417 \end{bmatrix}$$

$$C = RDR^T = \begin{bmatrix} 0.67787 & 0.73518 \\ -0.73518 & 0.67787 \end{bmatrix} \begin{bmatrix} 1.15562 & 0 \\ 0 & 0.04417 \end{bmatrix} \begin{bmatrix} 0.67787 & -0.73518 \\ 0.73518 & 0.67787 \end{bmatrix}$$

$$= \begin{bmatrix} 0.55489 & 0.55390 \\ 0.55390 & 0.64490 \end{bmatrix}$$



In our 2D Example, the Cov matrix:

$$C = \begin{bmatrix} 0.5549 & 0.5539 \\ 0.5539 & 0.6449 \end{bmatrix}$$

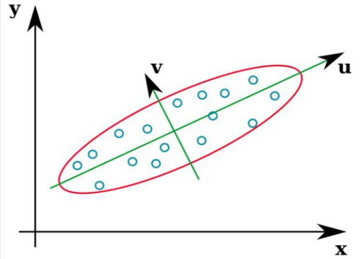
PCA Method (9). Transform Features.

Step 5. Find the transformed (translated + rotated) features.

$$C(r, c) = \frac{1}{M}\sum_{i=1}^{M}(r_i - r_\mu)(c_i - c_\mu);\ r, c \in 1..N$$

$$R = \begin{bmatrix} v_1^T \\ v_2^T \end{bmatrix} = \begin{bmatrix} 0.67787 & 0.73518 \\ -0.73518 & 0.67787 \end{bmatrix}$$

$\mathbf{x}_j \rightarrow \boldsymbol{\phi}_j \rightarrow \boldsymbol{\phi}_j'$



$$\phi'_{2xn} = R_{2x2}\phi_{2xn} = \begin{bmatrix} 0.67787 & 0.73518 \\ -0.73518 & 0.67787 \end{bmatrix}\begin{bmatrix} x_1 - x_\mu & x_2 - x_\mu & \dots & x_n - x_\mu \\ y_1 - y_\mu & y_2 - y_\mu & \dots & y_n - y_\mu \end{bmatrix};\ \phi'_i \text{ :transformed points.}$$

$$\phi'_i = R\,\phi_i = \begin{bmatrix} v_1^T \\ v_2^T \end{bmatrix}\begin{bmatrix} x_i - x_\mu \\ y_i - y_\mu \end{bmatrix}$$

The original data is first translated to the centroid.

It is then rotated by R so that the new x-axis is v_1 and y-axis is v_2.

The covariance matrix of the transformed data ϕ' is now a diagonal matrix with eigenvalue square as its diagonal. All cov(r, c) is removed to become 0's.

PCA Method (10). Dimension Reduction.

Step 6. Choose number of features to preserve p = 90% of data variance.

$$\text{variance}_{k-dim} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i}$$; Assumes λ_i are sorted in decreasing order.

Example for a 7-dimensional feature

We will reduce to k = 4 dimensions by discarding the last 3 dimensions of the transformed data. This roughly means k = 4 dimensions will preserve 91% of the data. If k = 7 will preserve 100%, but only transform the feature's coordinate axis.

k	λ_i	Sum (λ_i) i = 1..k	Variance p% in k dimensions
1	10.00	10.00	33%
2	8.00	18.00	59%
3	7.00	25.00	81%
4	3.00	28.00	91%
5	2.00	30.00	98%
6	0.50	30.50	99%
7	0.25	30.75	100%

PCA Method (11). Transforming back.

How to get the old data back once transformed?

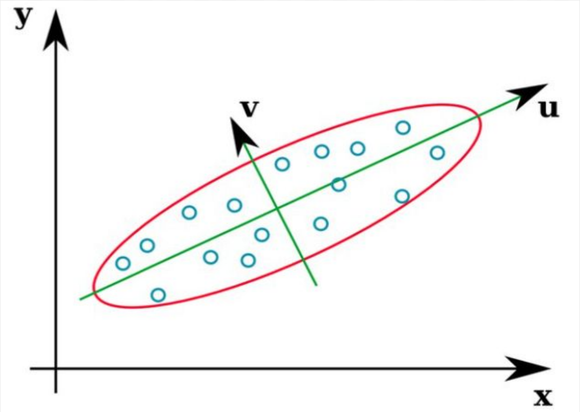
- Rotate back the axis first by R^{-1}.
- Then translate to previous origin by adding (x_μ, y_μ).

$$\phi_i' = R\,\phi_i\,; \quad R^{-1} = R^T$$

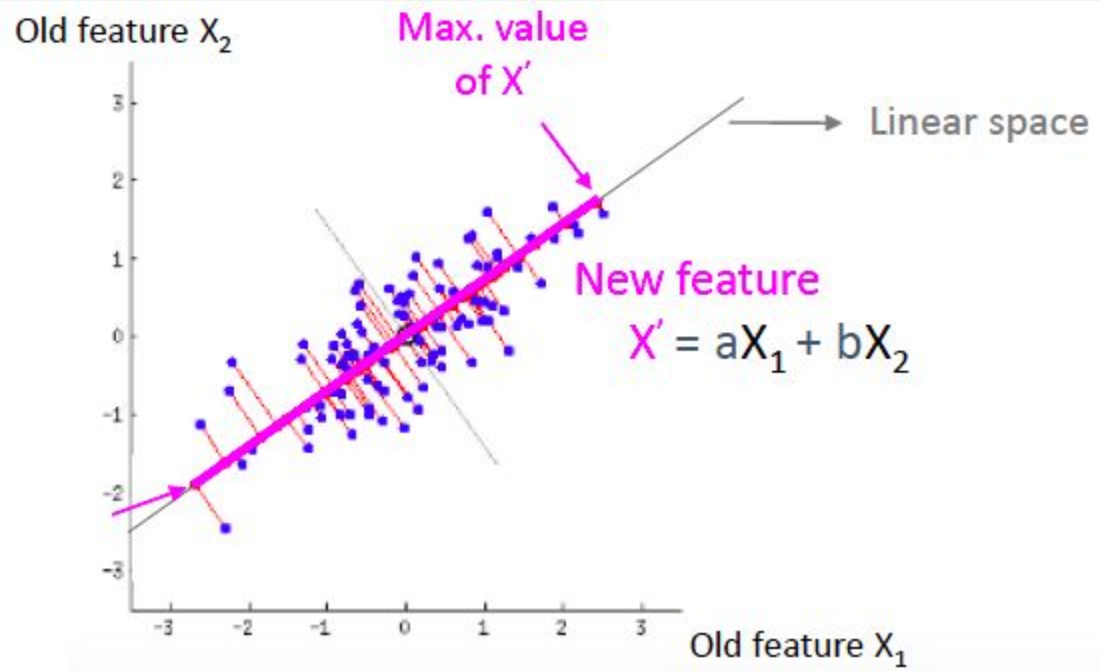
$$R^T\phi_i' = R^T R\phi_i; \qquad \phi_i = R^T\phi_i'$$

$$\begin{bmatrix} x_i - x_\mu \\ y_i - y_\mu \end{bmatrix} = R^T\phi_i'$$

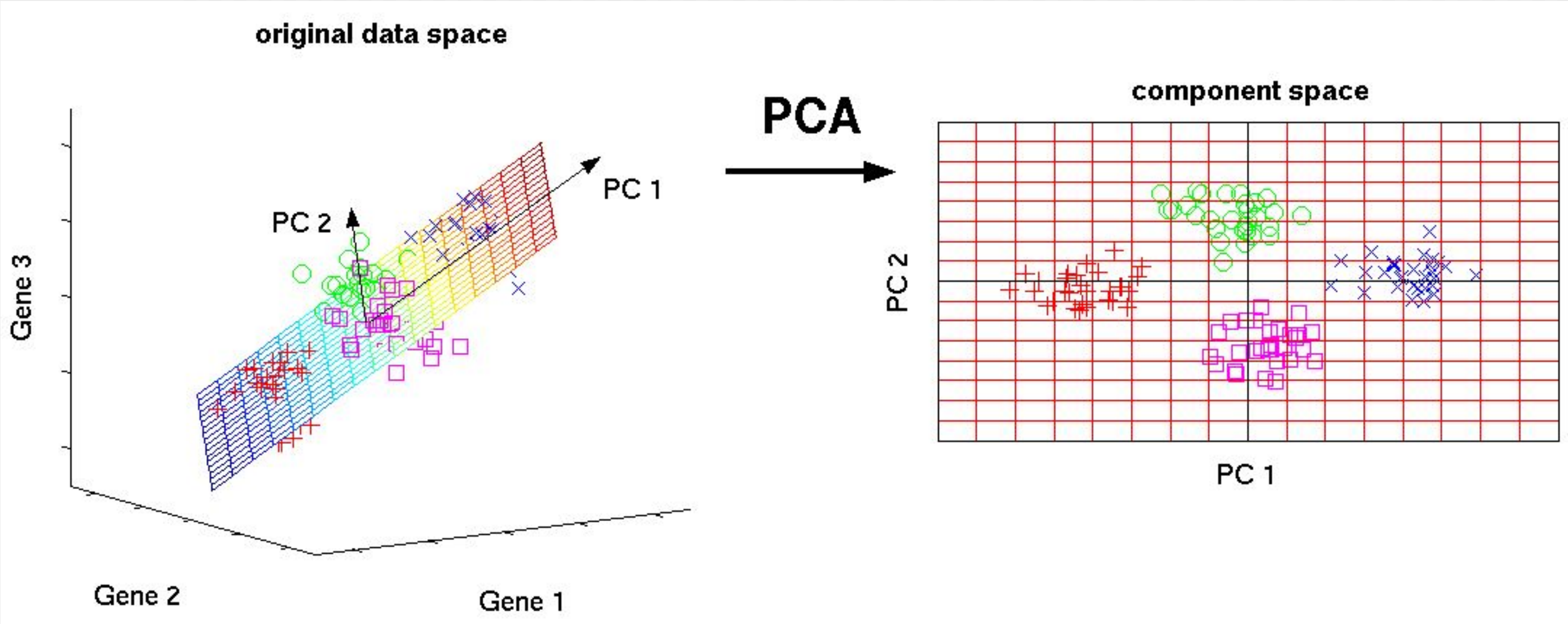
$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = R^T\phi_i' + \begin{bmatrix} x_\mu \\ y_\mu \end{bmatrix}$$



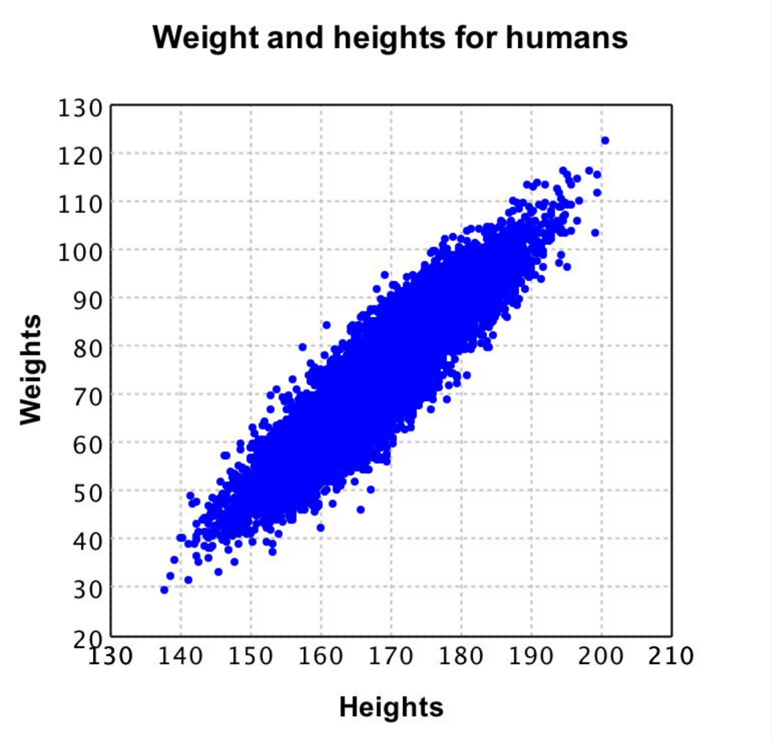
New Feature is Linear Combination of old

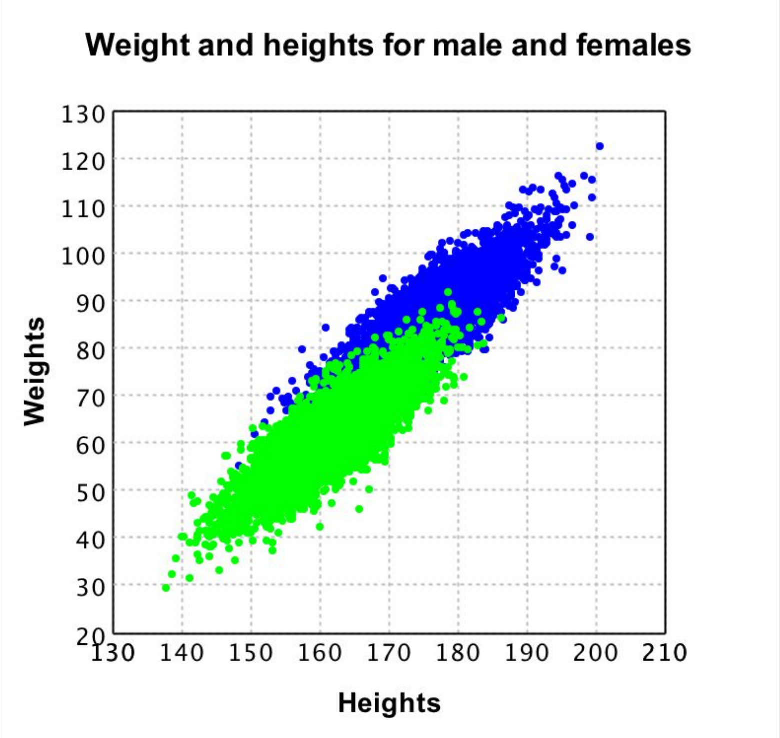


Reduce along Principal Component



Ex.: Height and Weight Correlated





PCA Examples

PCA Examples.ipynb

File Edit View Insert Runtime Tools Help Last saved at 12:38 AM

+ Code + Text

PCA

```
#copied mainly from https://machinelearningmastery.com/pr
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
```

```
# Load dataset
winedata = load_wine()
X = winedata['data']
y = winedata['target']
print("X shape:", X.shape)
```

1. dimension reduction for visualization
2. improving machine learning speed/convergence

Drawing Decision Boundaries

Decision Boundaries.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

DECISION BOUNDARIES

```python
from itertools import product
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
from sklearn.inspection import DecisionBoundaryDisplay
import matplotlib.pyplot as plt
```

```python
# Loading some example data
iris = datasets.load_iris()
```

Exercises: Quiz 7.

King Mongkut's University of Technology **Name:** ______________________

MEE 673 Machine Learning **I.D. Number:** ________________

Suthep Madarasmi, Ph.D.

Take Home Quiz 7 Due 21-Mar-2024 **Score:** ______ **/ 65**

Use the "PCA Examples.ipynb" for the following:

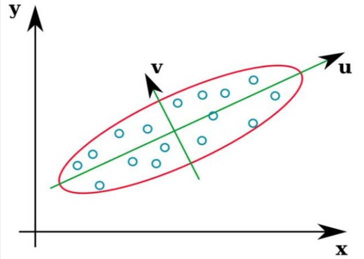
1. 15 points. 1 hour. Using the Wine dataset, use PCA to reduce it to 2 PCA features.
   - 1.1. Visualize the first 2 columns as a scatter plot using all (100%) of the data.
   - 1.2. In another graph visualize the scatter plot for the 2 PCA features also using all the data.
   - 1.3. What % of information ("explained variance") is preserved in the 2 PCA features?
2. 10 points. 1 hour. Use 25% for testing for the Wine dataset. Plot a graph of the accuracy of the samples when using 1, 2, 3, …, 13 PCA features along with showing how much % variance is preserved (p%) for each? Use logistic regression.

PCA for Face Recognition

Quick Review using 2D scatter

Transform features from (x, y) to principal component coordinates (u, v)

$$C(r, c) = \frac{1}{n}\sum_{i=1}^{n}(r_i - r_\mu)(c_i - c_\mu);\ r, c \in 1..2$$



$$R = \begin{bmatrix} v_1^T \\ v_2^T \end{bmatrix} = \begin{bmatrix} 0.67787 & 0.73518 \\ -0.73518 & 0.67787 \end{bmatrix}$$

$$\phi'_{2\text{xn}} = R_{2\text{x}2}\phi_{2\text{xn}} = \begin{bmatrix} 0.67787 & 0.73518 \\ -0.73518 & 0.67787 \end{bmatrix}\begin{bmatrix} x_1 - x_\mu & x_2 - x_\mu & \dots & x_n - x_\mu \\ y_1 - y_\mu & y_2 - y_\mu & \dots & y_n - y_\mu \end{bmatrix};\ \phi'_i\ \text{:transformed points.}$$

$$\phi'_i = R\,\phi_i = \begin{bmatrix} v_1^T \\ v_2^T \end{bmatrix}\begin{bmatrix} x_i - x_\mu \\ y_i - y_\mu \end{bmatrix}$$

The original data is first translated to the centroid.

It is then rotated by R so that the new x-axis is u and y-axis is v.

1. Prepare a training set of face images. Normalized to same size (r × c). Each image is treated as one vector with N = r × c elements. All training are in matrix T, where each column of the matrix is an image.
2. Subtract the mean. The average image has to be calculated and then subtracted from each original image in T. Cov matrix is between pixel (a, b) with a, b in 1..N

$$C(r,\, c) = \frac{1}{M}\sum_{i=1}^{M}(r_i - r_\mu)(c_i - c_\mu);\; r,\, c\; \in 1..\, N$$

$$R = [v_1 \quad v_2 \;\ldots\; v_N]^T;\; N - \text{number features};\; M - \text{number data samples (faces)}$$

$$\phi'_{\text{NxM}} = R_{\text{NxN}}\phi_{\text{NxM}} = \begin{bmatrix} v_1^T \\ v_2^T \\ \ldots \\ v_N^T \end{bmatrix} \begin{bmatrix} x_{1,1} - \bar{x}_1 & x_{1,2} - \bar{x}_1 & \ldots & x_{1,M} - \bar{x}_1 \\ x_{2,1} - \bar{x}_2 & x_{2,2} - \bar{x}_2 & \ldots & x_{2,M} - \bar{x}_2 \\ \vdots & \vdots & \ldots & \vdots \\ x_{N,1} - \bar{x}_N & x_{N,2} - \bar{x}_N & \ldots & x_{N,M} - \bar{x}_N \end{bmatrix}$$

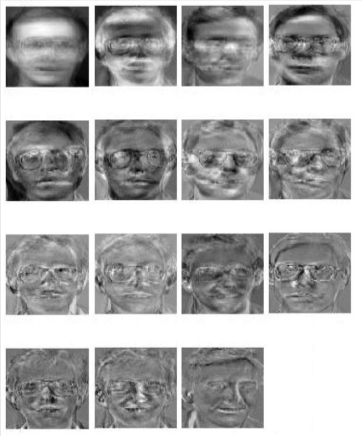
$$\phi'_{i,\,\text{Nx1}} = R_{\text{NxN}}\,\phi_{i,\,\text{Nx1}} = \begin{bmatrix} v_1^T \\ v_2^T \\ \ldots \\ v_N^T \end{bmatrix} \begin{bmatrix} x_1 - \bar{x}_1 \\ x_2 - \bar{x}_2 \\ \vdots \\ x_N - \bar{x}_N \end{bmatrix};\; \phi'_{i,\,\text{4x1}} = R_{\text{4xN}}\,\phi_{i,\,\text{Nx1}} = \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \\ v_4^T \end{bmatrix} \begin{bmatrix} x_1 - \bar{x}_1 \\ x_2 - \bar{x}_2 \\ \vdots \\ x_N - \bar{x}_N \end{bmatrix}$$

3. Calculate the eigenvectors and eigenvalues of the covariance matrix C. There are N such ordered eigenvectors each of N dimensions. The top eigenvectors can be converted to pixel values to view as top eigenfaces.
4. Choose k principal components to preserve p% of the variance. For 100 x 100 image with N = 10,000, we find that k = 100 is usually enough.
5. For training and testing samples convert the transformed face image with N = r x c dimensions to only k dimensions as new features.
6. Use the reduced features for classification such as template matching, MLP, logistic regression.

Why use PCA for face recognition?

- Strengths:
  - Reduce the number of dimension without much information loss
  - High compression rate
- Weakness:
  - Images Limited to same illumination variation with a whole face

Original Training Images

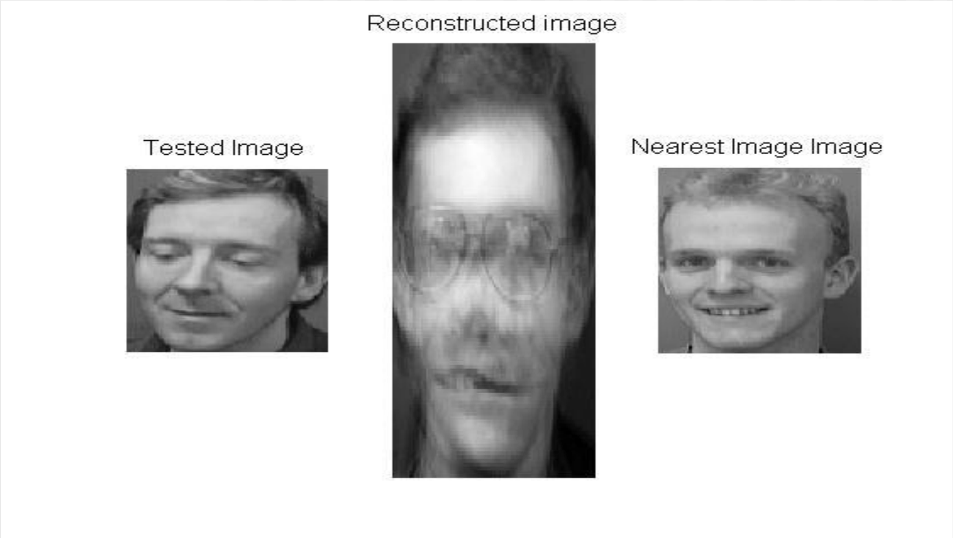


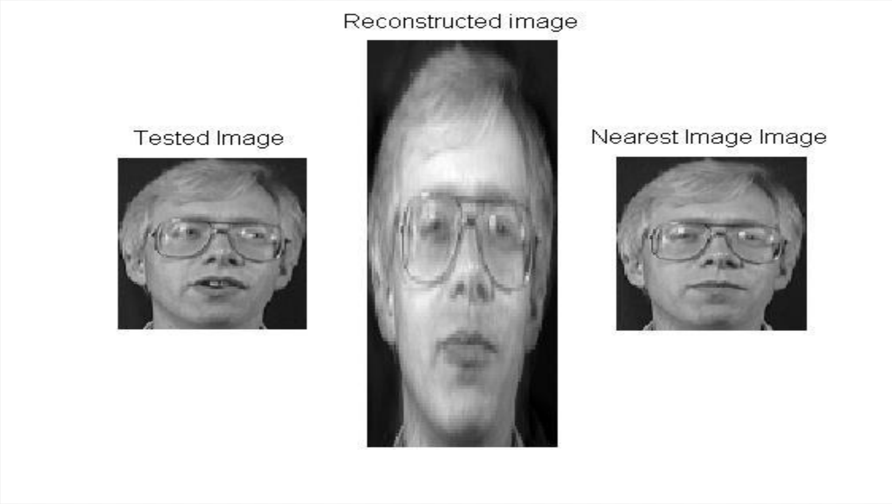
PCA axis transformed images (New features)

PCA

The test image is transformed to lower dimensional features.

Best match by template matching using transformed images of lower dimension.





What are Eigenfaces?

- Eigenfaces are the eigenvectors of the covariance matrix of the probability distribution of the vector space of human faces

- For a 100x100 image there are 10,000 eigenfaces. But only around 100 eigenfaces are needed to preserve 90% of the variance.

Face Recognition using PCA

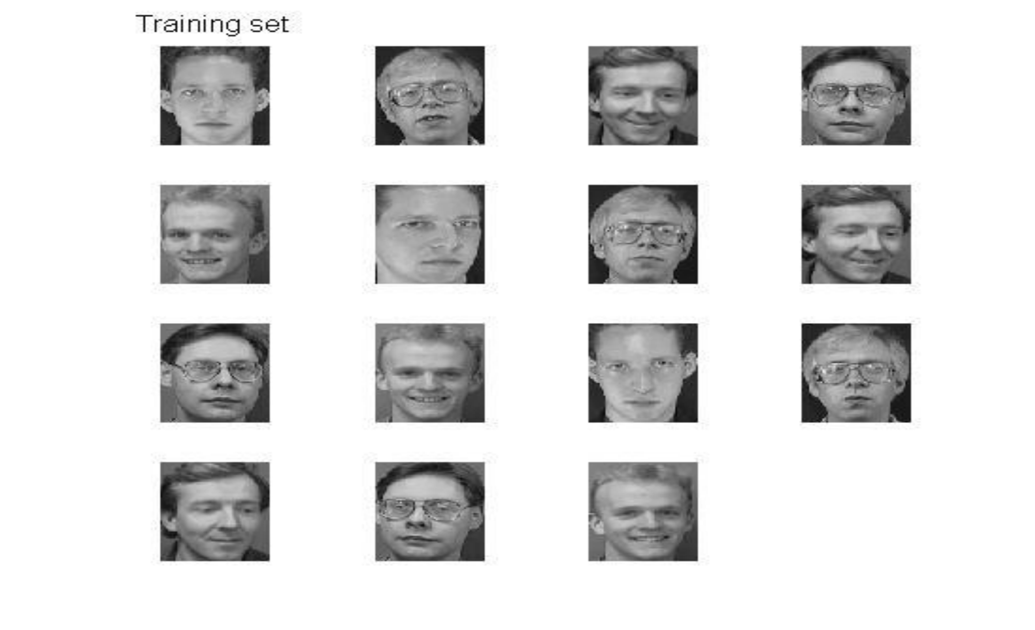
- Training Steps
  - Training Image Set
  - Preprocessing
  - PCA / Eigenfaces
  - Dimensionality Reduction
  - Calculation Weight

- Testing Steps
  - Testing Image
  - Preprocessing
  - Transformed into Eigenface Components
  - Finding minimum of the Euclidean distance (template matching)

Training Steps (1). Same size inputs.

Training Image Set

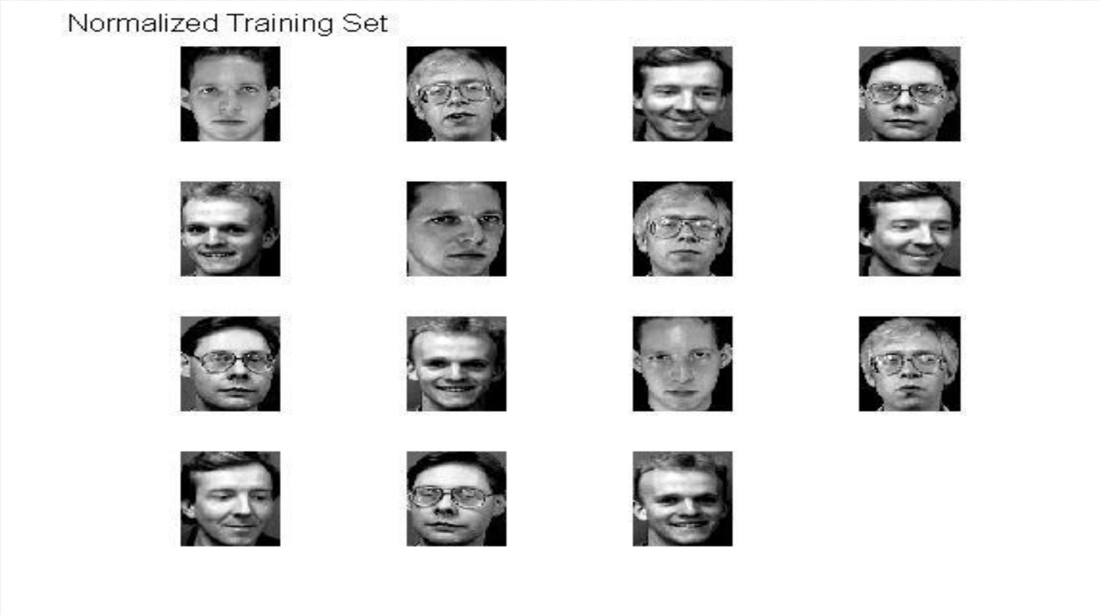
- All Image same size: Row * Col = r x c
- Training Images: M = 15 Images



Training Steps (2). Preprocessing.

Preprocessing

- Normalized Training Image Set
- Reduce noise
- Reduce lighting variation



Training Steps (3). Find Eigen λ, v.

PCA / Eigenfaces

- Each Image is transformed into a vector of size N. N = r x c.
- Obtain a set Γ of M faces data, each vectorized to N = r x c.

$$\Gamma = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_M\}$$

1	2	3	…
4	5	6	…
7	8	9	…
…	…	…	…

N = r x c

1
2
3
:
N

N x 1

Γ

1	1	..	1
2	2	..	2
3	3	..	3
:	:	:	:
N	N	..	N

M faces, each with N features

Training Steps (4). Find Mean Face.

PCA / Eigenfaces

- Find the Mean Image $\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$ Average of M faces

Training Steps (5). Subtract Mean Face.

PCA / Eigenfaces

- Find the difference between one face input image and mean image

$$\Phi_i = \Gamma_i - \Psi_i$$

Φ

1	1	..	1
2	2	..	2
3	3	..	3
:	:	:	:
N	N	..	N

M faces data,
each with mean subtracted.

Training Steps (6). Find Covariance Matrix

PCA / Eigenfaces

- Find the Covariance matrix C

$$C = \frac{1}{M}\sum_{i=1}^{M}\phi_i\phi_i^T$$

$$C(r,\, c) = \frac{1}{M}\sum_{i=1}^{M}(r_i - r_\mu)(c_i - c_\mu);\; r,\; c \in 1..N$$

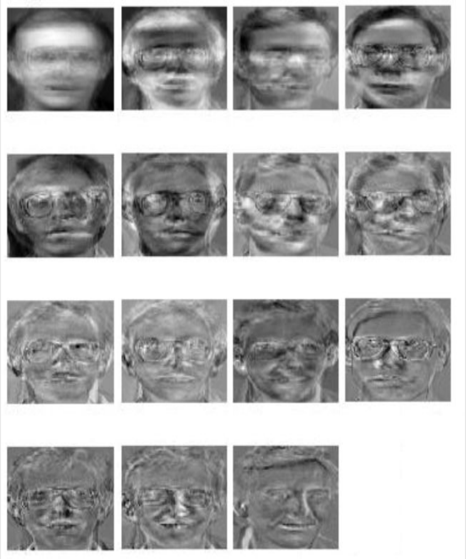
Training Steps (7)

PCA / Eigenfaces

- Find Eigenvectors of the Covariance matrix C. Create the R matrix.
- Transform all M = 15 faces to new rotated coordinates.

Φ

Φ'



$$\phi'_{\text{NxM}} = R_{\text{NxN}} \phi_{\text{NxM}}$$

$$= \begin{bmatrix} v_1^T \\ v_2^T \\ \dots \\ v_N^T \end{bmatrix} \begin{bmatrix} x_{1,1} - \overline{x}_1 & \dots & x_{1,M} - \overline{x}_1 \\ x_{2,1} - \overline{x}_2 & \dots & x_{2,M} - \overline{x}_2 \\ \vdots & \dots & \vdots \\ x_{N,1} - \overline{x}_N & \dots & x_{N,M} - \overline{x}_N \end{bmatrix}$$

Training Steps (8)

Reduce Dimensions

- New Features as weights of Eigenfaces

$$R = [v_1 \quad v_2 \quad \ldots \quad v_N]^T$$

$$\phi'_i = R\,\phi_i \text{ for i} = 1..M \text{ faces}$$

- The weight describes the contribution of each eigenface in representing the training image set.
- Choose dimension k is dimension reduction, where k ≤ N.

$$R = [v_1 \quad v_2 \quad \ldots \quad v_k]^T$$

$$\phi'_{i,\,\mathrm{kx1}} = R_{\mathrm{kxN}}\,\phi_{i,\,\mathrm{Nx1}} \text{ for i} = 1..M \text{ faces}$$

Testing Steps (1)

- Testing Image: Γ
- Preprocessing
- Transformed into Eigenface Components

$$R = [v_1 \quad v_2 \quad \ldots \quad v_k]^T$$

$$\Gamma' = R(\Gamma - \Psi)$$

$$R = [v_1 \quad v_2 \quad \ldots \quad v_k]^T$$

$$\phi'_{i,\,\mathrm{kx1}} = R_{\mathrm{kxN}}\,\phi_{i,\,\mathrm{Nx1}} \text{ for i} = 1..M \text{ faces}$$

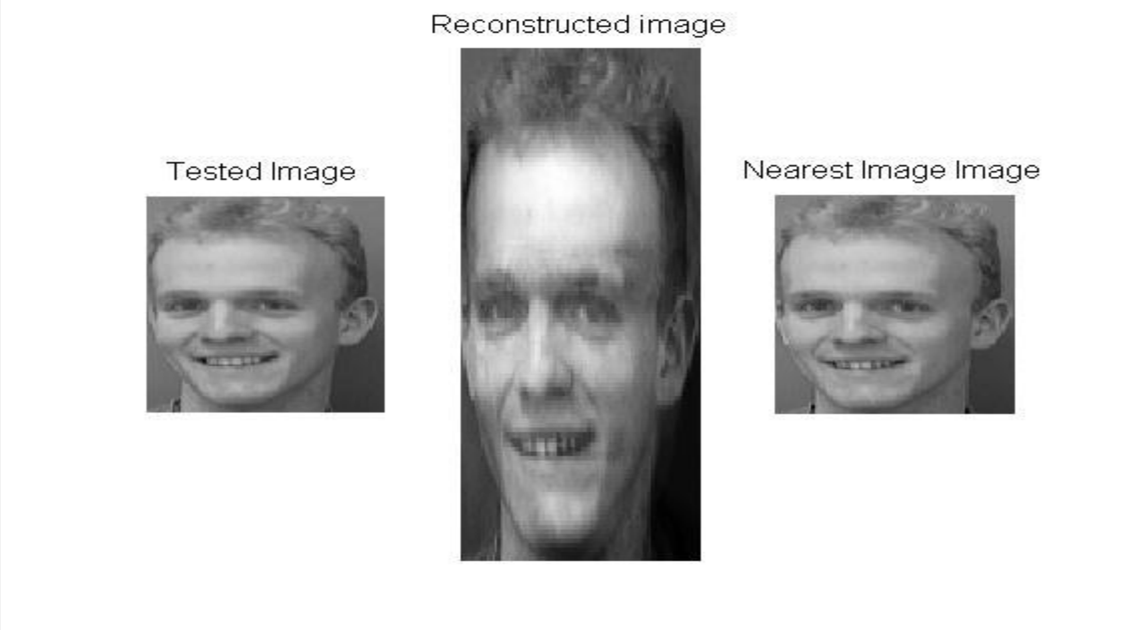
- Note that k ≤ N dimensions is used in the test image as well. These values are the weights of the k-best Eigenvectors.

Testing Steps (2)

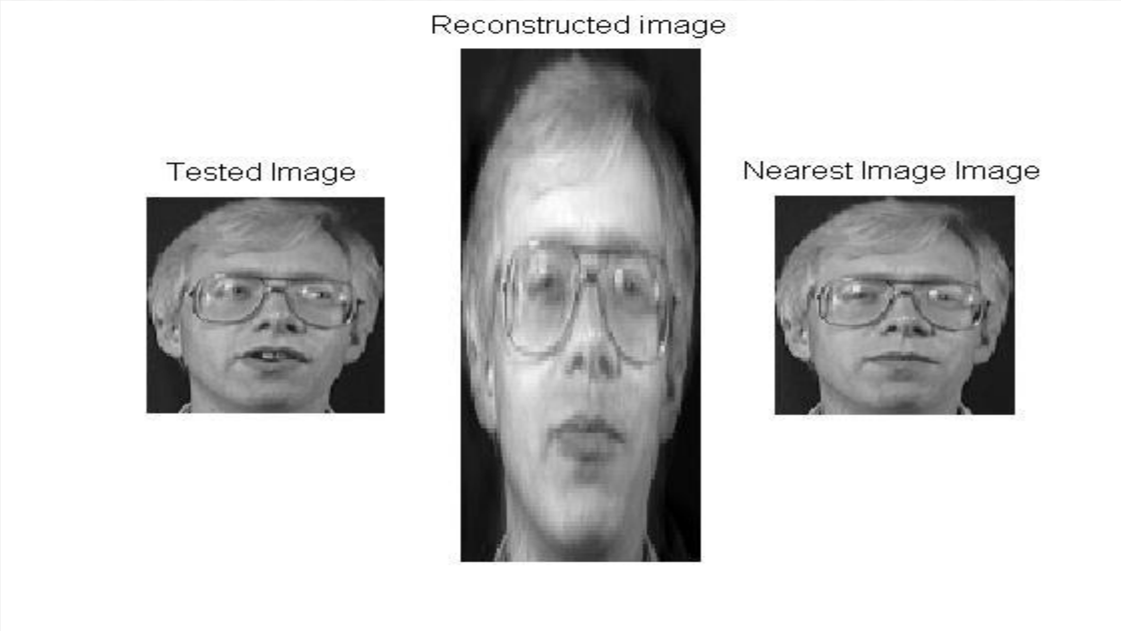
Report the image with minimum Euclidean distance between the testing image and training image with transformed features and reduced dimensionality.

- This is just template matching among the new features.
- Can also use another classifier such as KNN, MLP, Logistic Regression.

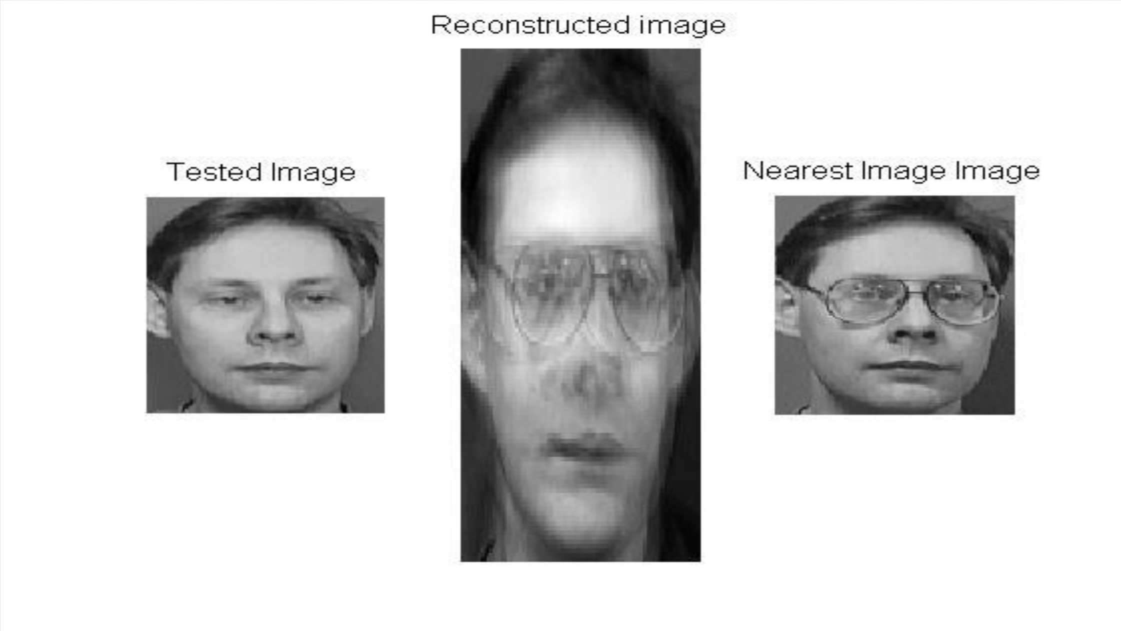
Experimental Results (1)



Experimental Results (2)

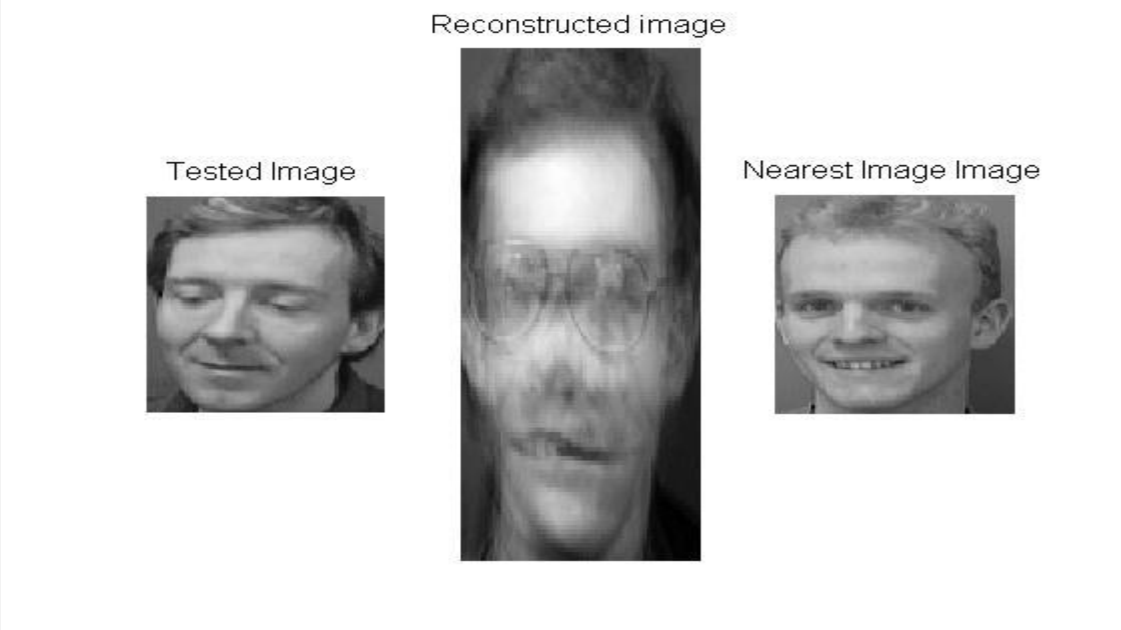


Experimental Results (3)



Experimental Results (4)

see for python implementation example:
https://machinelearningmastery.com/face-recognition-using-principal-component-analysis/



Classification Metrics

Program Goal: To recognize Dogs.

- Actual: 10 cats and 12 dogs.
- Identified: 8 dogs, only 5 correct (tp) and 3 are wrong (fp).
- Missed: 7 dogs missed (fn), 7 cats correctly excluded (tn)
- Accuracy = (5+7) / (5+7+3+7) = 12/22
- Precision = 5/8 (TP / selected elements). How valid are the results?
- Recall or Sensitivity = 5/12 (TP / relevant elements). How complete are the results?

Search Engine

- Should return: 60 pages
- Returns 30 pages, 20 correct (tp), 10 wrong (fp)
- Missed: 40 pages (fn),
- Precision = 20/30. Recall = 20/60.

F-1 score measures harmonic mean between precision & recall

		Predicted Class	
		Positive	**Negative**
Actual Class	**Positive**	True Positive (TP)	False Negative (FN) **Type II Error**
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)

$$\text{Precision} = \frac{tp}{tp + fp}$$

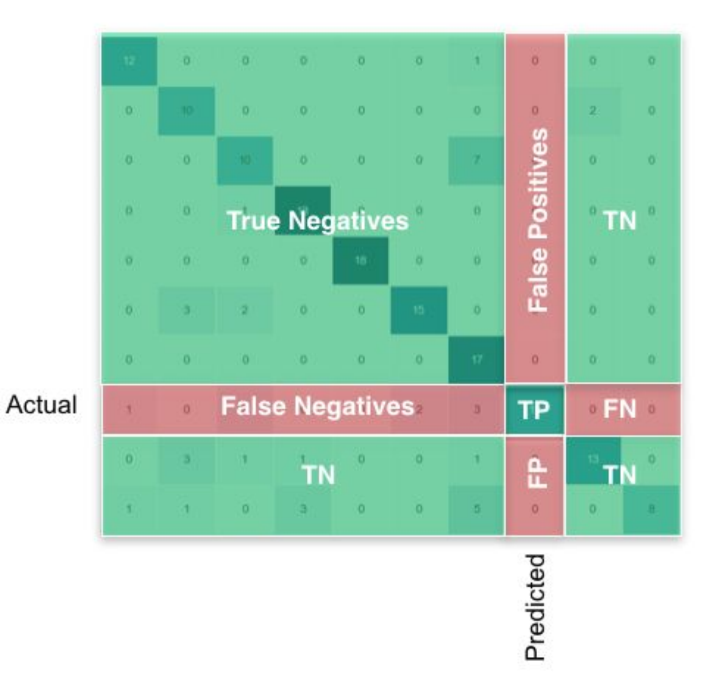
$$\text{Recall} = \frac{tp}{tp + fn}$$

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

“True” - Success.
“False” - Failed.
“Positive” - should be Yes class.
“Negative” - should be No class.

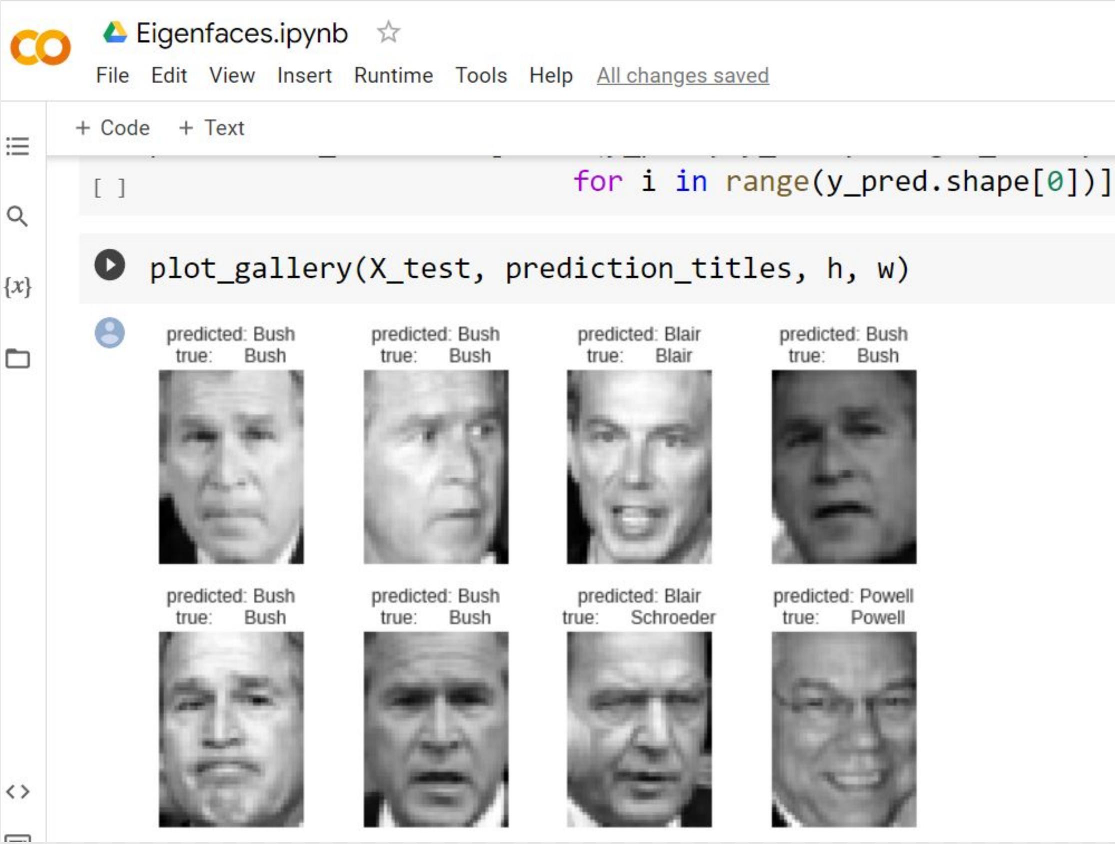
Classification Metrics



		Predicted Class: Positive	Predicted Class: Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	**Sensitivity (Recall)** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

“True” - Successful Classification. “False” - Failed Classification.
“Positive” - should be Yes class. “Negative” - should be No class.

Example Eigenfaces



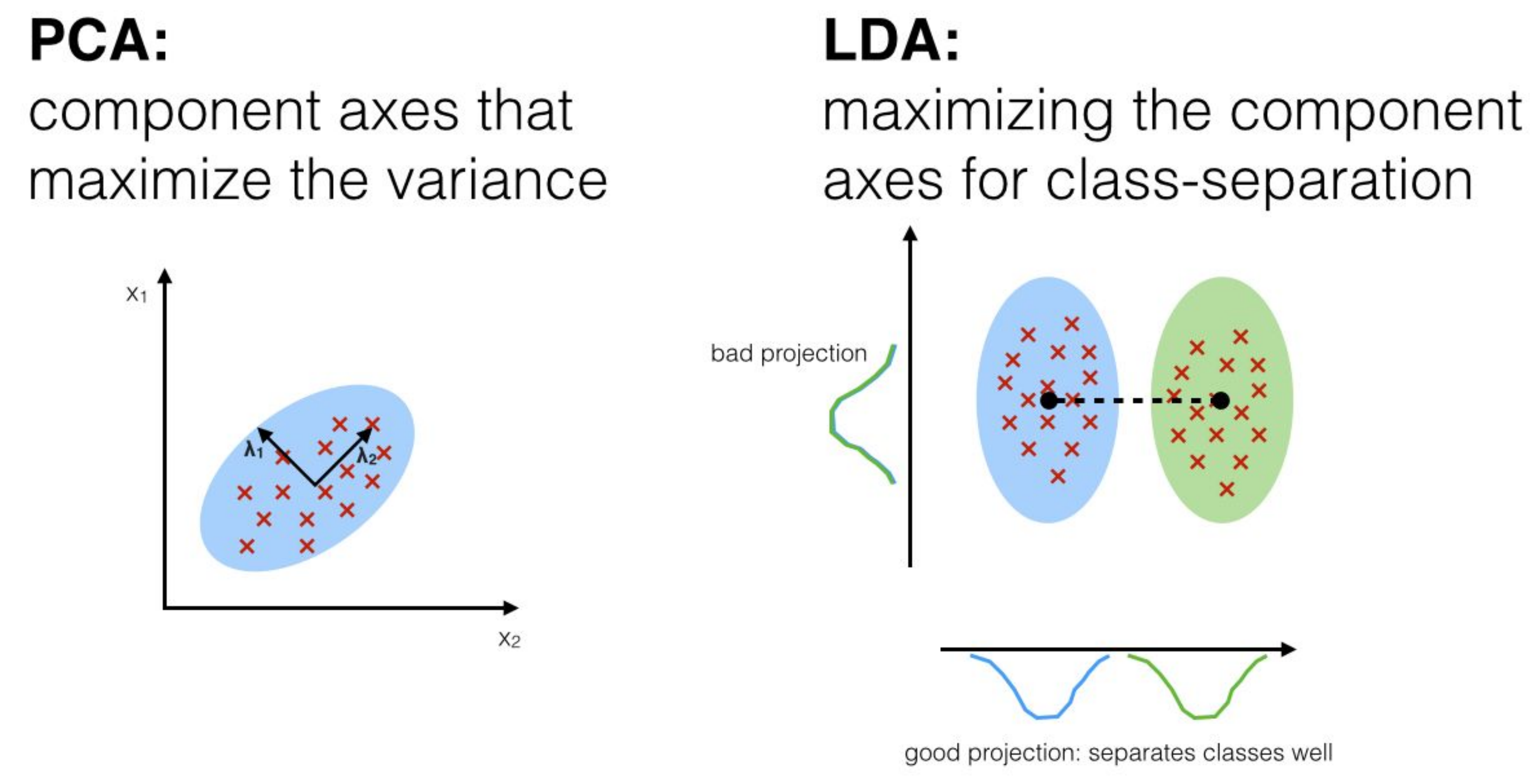
PCA Conclusion

- Advantages
  - PCA can reduce the number of dimensions without much loss of information
  - Performance of recognition is good even when noise is present
  - PCA can be used for face recognition and face reconstruction
  - Best low-dimensional space can be determined by the Best Eigenvectors of the covariance matrix
- Limitations
  - PCA is appearance-based face recognition algorithm so PCA performs well when face images are scaled to same size even with rotation, but no occlusion.

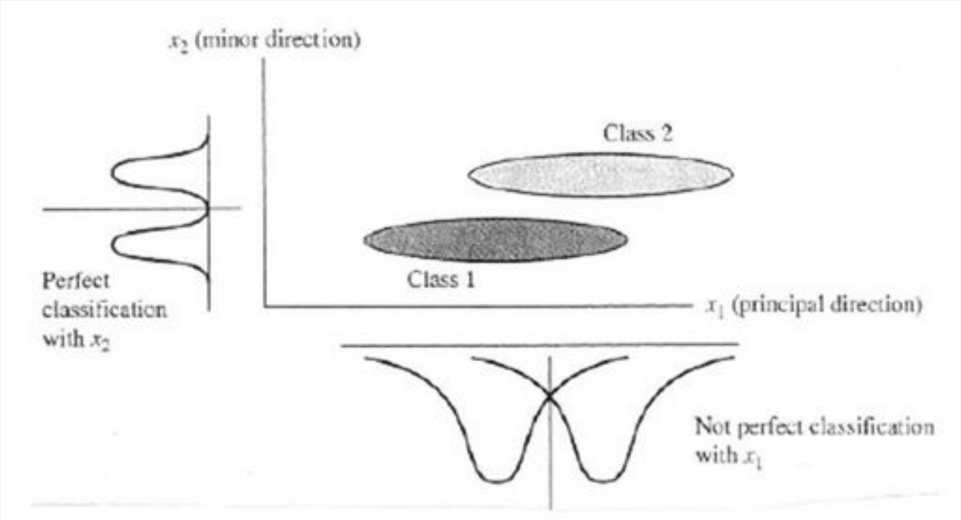
Linear Discriminant Analysis (LDA) – A Variation of PCA

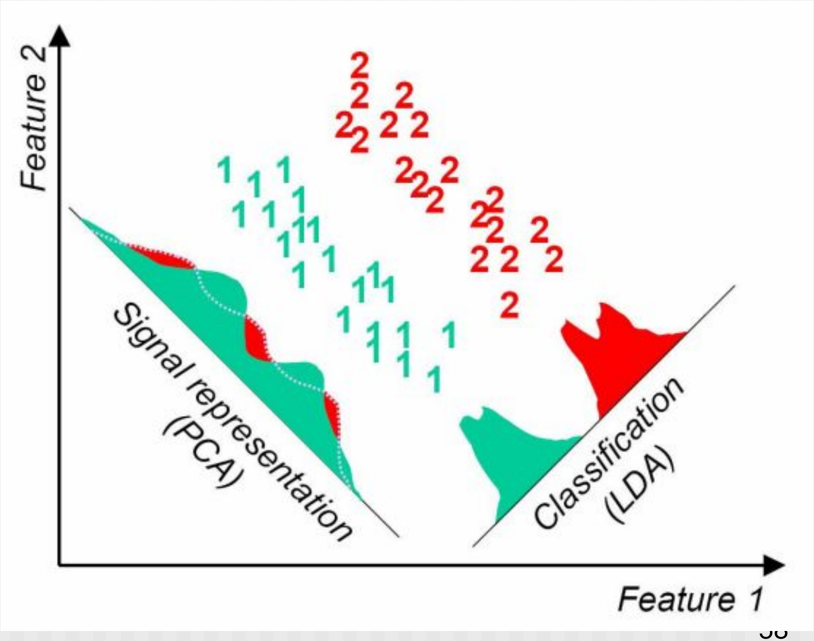
- Procedure to find model the difference between the classes of data to separate the classes well
  - Consider for the scatter data ***within-classes*** and also the scatter data ***between-classes***.
  - Works better than PCA when you have n samples of one person's face compared to only 1 sample of one person's face.

PCA doesn't always transform data for best class discrimination



LDA does better than PCA in dimension reduction for classification





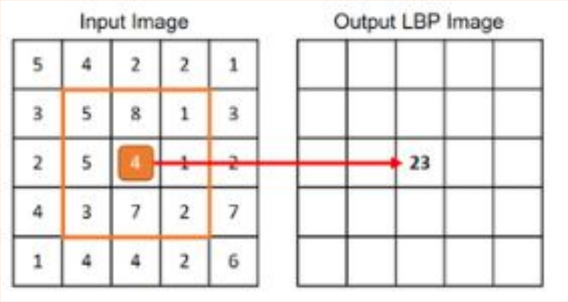
LDA & PCA Conclusion

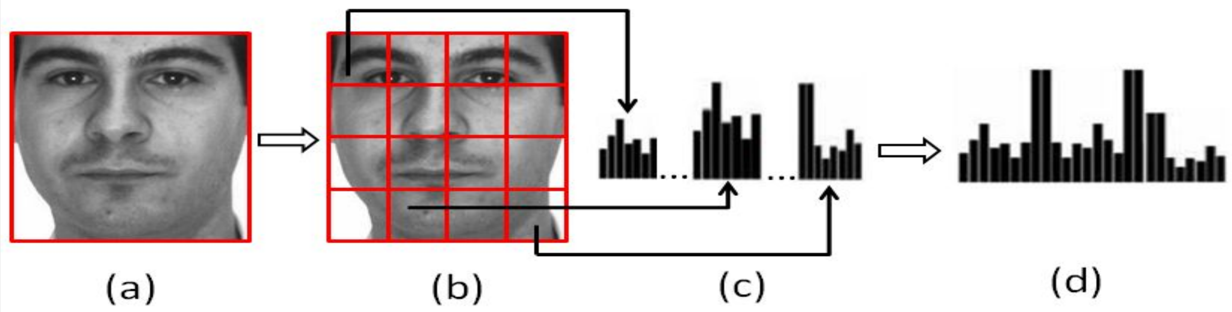
- LDA is suitable for pattern classification when the number of training samples of each class are large
- PCA is suitable to reduce dimension without difference class separation
- PCA can outperform LDA when the training set is small
- LDA can outperform PCA when the number of samples is large for each class

Local Binary Pattern (LBP)

Efficient feature-based Face Recognition that partitions the image into blocks and uses histogram features from 8-nbrs per pixel in each block.

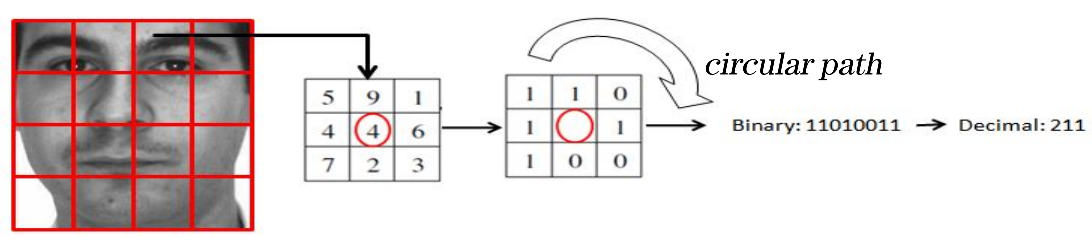
Step 1. Input Image to LBP Image.





Each step of the LBP approach. (a) Face image (b) Face image divided into blocks (c) Each block has a LBP histogram of 256 values (d) Concatenated feature of 16 histograms

How each pixel's LBP is obtained and converted to a decimal representation used for histogram. Histogram shows the frequency of each decimal no. per block. 16 blocks = 16 histograms.



Some references

https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60

https://pyimagesearch.com/2021/05/10/opencv-eigenfaces-for-face-recognition/

https://learnopencv.com/eigenface-using-opencv-c-python/

https://pythonmachinelearning.pro/face-recognition-with-eigenfaces/

https://scipy-lectures.org/packages/scikit-learn/auto_examples/plot_eigenfaces.html

https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python