

60004210210

Amartya Mishra

COMPS – C31

ML Experiment 4

## ML Experiment 4

Aim: To implement Principle Component Analysis (PCA)

Theory:

PCA works on the condition that while the data in a higher dimensional space is mapped to data in lower dimensional space the variance of the data in the lower dimensional space should be maximum

PCA is a statistical procedure that uses an orthogonal transformation that converts a set of co-related variables to a set of unco-related variables

It is an unsupervised learning algorithm used to examine the iterations among a set of variables

steps

I) Standardization

$$x = \frac{x - \bar{x}}{s}$$

II) Covariance Matrix Computation

$$\text{cov}(x_1, x_2) = \frac{1}{n-1} \sum_{i=1}^n (x_{1i} - \bar{x}_1)(x_{2i} - \bar{x}_2)$$

iii) Compute Eigen values & Eigen Vectors of covariance matrix to identify principal components.

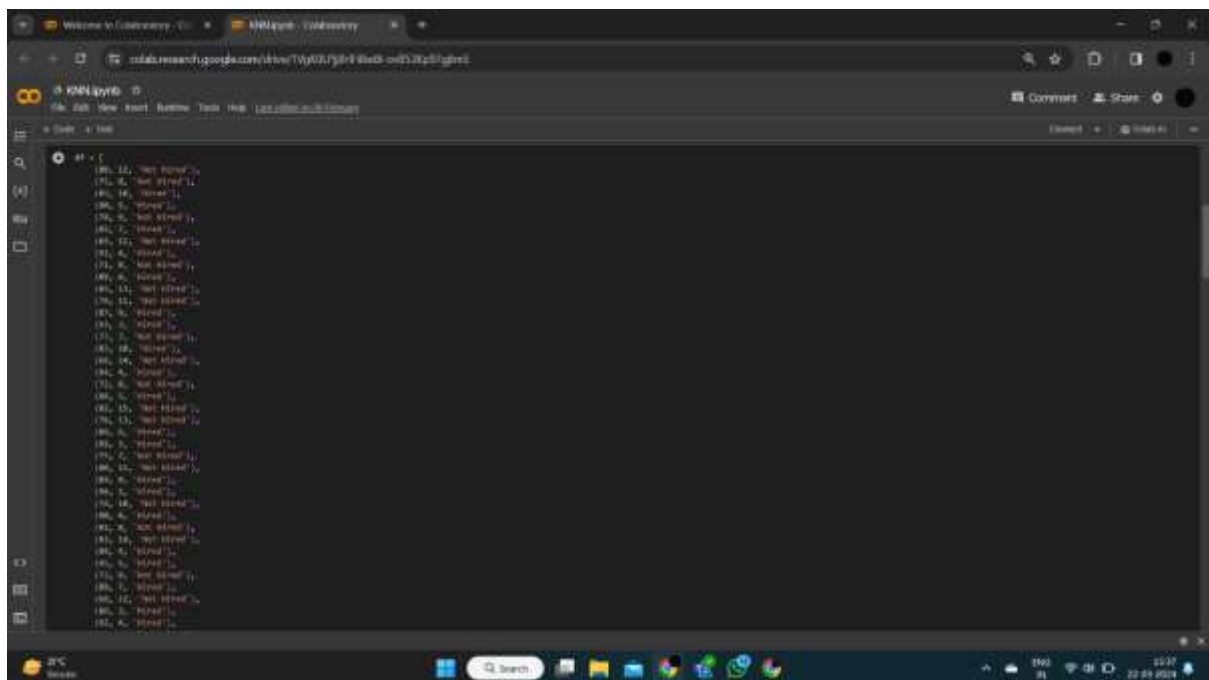
$$Ax = \lambda x$$

$$Ax - \lambda x = 0$$

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

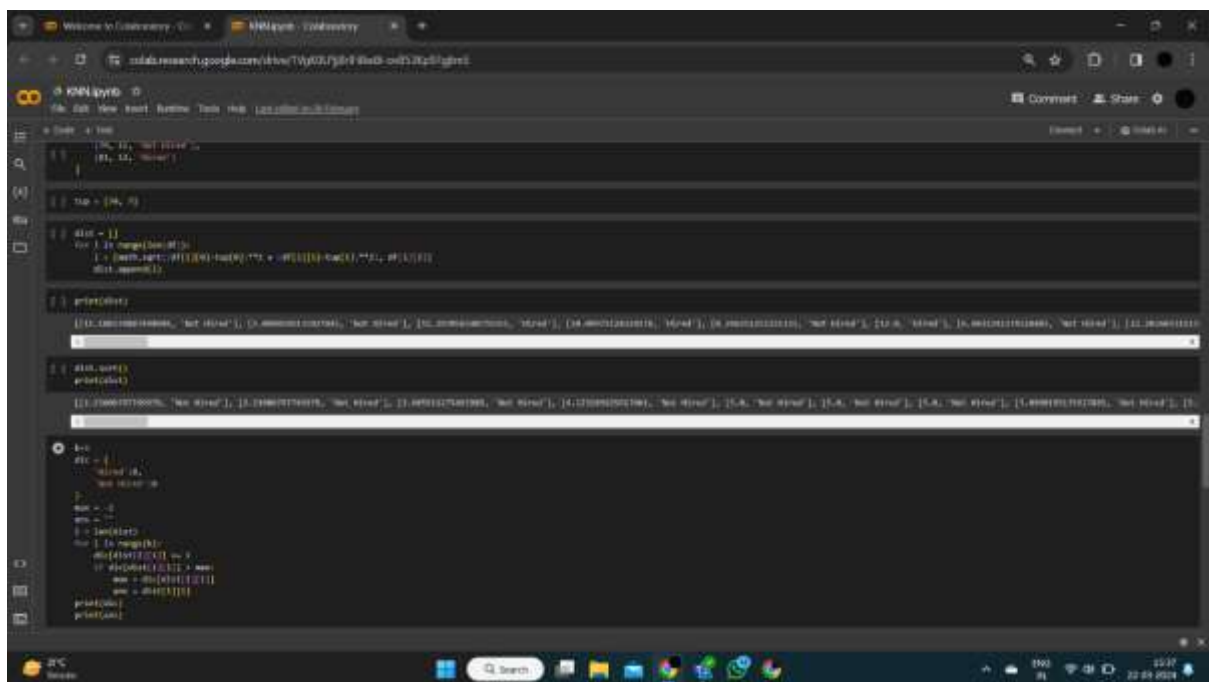
Conclusion: Thus we implement PCA

## Implementation:



A screenshot of a Jupyter Notebook interface. The browser address bar shows a Google Colab link. The notebook has a single code cell containing a list of 20 coordinate pairs, each with a string and a list of two integers. The pairs are: (10, 12, "Not found"), (11, 8, "Not found"), (10, 10, "found"), (10, 6, "found"), (10, 8, "Not found"), (10, 2, "found"), (10, 12, "Not found"), (11, 6, "found"), (11, 8, "Not found"), (11, 10, "Not found"), (11, 12, "Not found"), (12, 6, "found"), (12, 8, "Not found"), (12, 10, "Not found"), (12, 12, "Not found"), (13, 6, "found"), (13, 8, "Not found"), (13, 10, "Not found"), (13, 12, "Not found"), (14, 6, "found"), (14, 8, "Not found"), (14, 10, "Not found"), (14, 12, "Not found"). The bottom status bar shows 8°C and a Windows taskbar.

```
# = []
(10, 12, "Not found"),
(11, 8, "Not found"),
(10, 10, "found"),
(10, 6, "found"),
(10, 8, "Not found"),
(10, 2, "found"),
(10, 12, "Not found"),
(11, 6, "found"),
(11, 8, "Not found"),
(11, 10, "Not found"),
(11, 12, "Not found"),
(12, 6, "found"),
(12, 8, "Not found"),
(12, 10, "Not found"),
(12, 12, "Not found"),
(13, 6, "found"),
(13, 8, "Not found"),
(13, 10, "Not found"),
(13, 12, "Not found"),
(14, 6, "found"),
(14, 8, "Not found"),
(14, 10, "Not found"),
(14, 12, "Not found")
```



A screenshot of a Jupyter Notebook interface showing a search algorithm implementation. The code cell contains a function `find` that takes a list of coordinates and a target coordinate. It uses a recursive approach to find the target. The function prints the current state of the list and the target. The output shows the function being called with the list of coordinates and the target (10, 12). The function returns the target coordinate. The bottom status bar shows 8°C and a Windows taskbar.

```
def find(coordinates, target):
    if len(coordinates) == 0:
        return None
    if target in coordinates:
        return target
    else:
        return find(coordinates[1:], target)

coordinates = [(10, 12, "Not found"), (11, 8, "Not found"), (10, 10, "found"), (10, 6, "found"), (10, 8, "Not found"), (10, 2, "found"), (10, 12, "Not found"), (11, 6, "found"), (11, 8, "Not found"), (11, 10, "Not found"), (11, 12, "Not found"), (12, 6, "found"), (12, 8, "Not found"), (12, 10, "Not found"), (12, 12, "Not found"), (13, 6, "found"), (13, 8, "Not found"), (13, 10, "Not found"), (13, 12, "Not found"), (14, 6, "found"), (14, 8, "Not found"), (14, 10, "Not found"), (14, 12, "Not found")]
target = (10, 12)

print(find(coordinates, target))
```

```
def main():
    n = 10
    arr = [0] * n
    arr[0] = 1
    for i in range(1, n):
        arr[i] = arr[i-1] + arr[i-2]
    print(arr)

def main():
    n = 10
    arr = [0] * n
    arr[0] = 1
    for i in range(1, n):
        arr[i] = arr[i-1] + arr[i-2]
    print(arr)
```

```
def main():
    n = 10
    arr = [0] * n
    arr[0] = 1
    for i in range(1, n):
        arr[i] = arr[i-1] + arr[i-2]
    print(arr)
```