Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Data Mining

Lab - 6

Mohil Parmar | 23010101192 | 11/7/2025

Dimensionality Reduction using NumPy

🔍 What is Data Reduction?

Data reduction refers to the process of reducing the amount of data that needs to be processed and stored, while preserving the essential patterns in the data.

Why do we reduce data?

- To reduce computational cost.
- To remove noise and redundant features.
- To improve model performance and training time.

- To visualize high-dimensional data in 2D or 3D.

Common data reduction techniques include:

- Principal Component Analysis (PCA)
- Feature selection
- Sampling

What is Principal Component Analysis (PCA)?

PCA is a **dimensionality reduction technique** that transforms a dataset into a new coordinate system. It identifies the **directions (principal components)** where the variance of the data is maximized.

Key Concepts:

- **Principal Components**: New features (linear combinations of original features) capturing most variance.
- **Eigenvectors & Eigenvalues**: Used to compute these principal directions.
- **Covariance Matrix**: Measures how features vary with each other.

PCA helps in **visualizing high-dimensional data**, **noise reduction**, and **speeding up algorithms**.

NumPy Functions Summary for PCA

Function	Purpose
`np.mean(X, axis=0)`	Compute mean of each column (feature-wise mean).
`X - np.mean(X, axis=0)`	Centering the data (zero mean).
`np.cov(X, rowvar=False)`	Compute covariance matrix for features.
`np.linalg.eigh(cov_mat)`	Get eigenvalues and eigenvectors (for symmetric matrices).
`np.argsort(values)[::-1]`	Sort values in descending order.

Function	Purpose
`np.dot(X, eigenvectors)`	Project original data onto new axes.

Step 1: Load the Iris Dataset

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
iris = pd.read_csv('../data/iris.csv')
iris
```

Out[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [3]:
```
X = iris.drop(columns = "species")
y = iris['species'].map({
    'setosa': 0,
    'versicolor': 1,
    'virginica': 2
})
```

In [4]:
```
print('Orignal shape:', X.shape)
```

```
Orignal shape: (150, 4)
```

Step 2: Standardize the data (zero mean)

In [5]:
```
mean = np.mean(X, axis=0)
print("Mean of each feature:\n", mean)
```

```
Mean of each feature:
 sepal_length    5.843333
sepal_width     3.057333
petal_length    3.758000
petal_width     1.199333
dtype: float64
```

In [6]:
```
X_meaned = X - np.mean(X, axis = 0)
X_meaned.head()
```

Out[6]:

	sepal_length	sepal_width	petal_length	petal_width
0	-0.743333	0.442667	-2.358	-0.999333
1	-0.943333	-0.057333	-2.358	-0.999333
2	-1.143333	0.142667	-2.458	-0.999333
3	-1.243333	0.042667	-2.258	-0.999333
4	-0.843333	0.542667	-2.358	-0.999333

Step 3: Compute the Covariance Matrix

In [7]:
```
cov_mat = np.cov(X_meaned, rowvar = False)
cov_mat.shape
```

Out[7]:
```
(4, 4)
```

In [8]:
```
print(cov_mat)
```

```
[[ 0.68569351 -0.042434    1.27431544  0.51627069]
 [-0.042434    0.18997942 -0.32965638 -0.12163937]
 [ 1.27431544 -0.32965638  3.11627785  1.2956094 ]
 [ 0.51627069 -0.12163937  1.2956094   0.58100626]]
```

Step 4: Compute eigenvalues and eigenvectors

In [9]:
```
eigen_values, eigen_vectors = np.linalg.eigh(cov_mat)
print('Eigen Values:\n', eigen_values)
print('Eigen Vectors:\n', eigen_vectors)
```

```
Eigen Values:
 [0.02383509 0.0782095  0.24267075 4.22824171]
Eigen Vectors:
 [[ 0.31548719  0.58202985  0.65658877 -0.36138659]
 [-0.3197231  -0.59791083  0.73016143  0.08452251]
 [-0.47983899 -0.07623608 -0.17337266 -0.85667061]
 [ 0.75365743 -0.54583143 -0.07548102 -0.3582892 ]]
```

Step 5: Compute eigenvalues and eigenvectors

In [10]:
```
sorted_index = np.argsort(eigen_values)[::-1]
sorted_eigenvalues = eigen_values[sorted_index]
sorted_eigenvectors = eigen_vectors[:, sorted_index]
```

```python
print('Sorted index:\n', sorted_index)
print('Sorted Eigen Values:\n', sorted_eigenvalues)
print('Sorted Eigen Vectors:\n', sorted_eigenvectors)
```

```
Sorted index:
 [3 2 1 0]
Sorted Eigen Values:
 [4.22824171 0.24267075 0.0782095  0.02383509]
Sorted Eigen Vectors:
 [[-0.36138659  0.65658877  0.58202985  0.31548719]
 [ 0.08452251  0.73016143 -0.59791083 -0.3197231 ]
 [-0.85667061 -0.17337266 -0.07623608 -0.47983899]
 [-0.3582892  -0.07548102 -0.54583143  0.75365743]]
```

Step 6: Select the top k eigenvectors (top 2)

In [11]:

```python
k = 2
eigenvector_subset = sorted_eigenvectors[:, 0:k]
print('Eigen Vector subset:\n', eigenvector_subset)
```

```
Eigen Vector subset:
 [[-0.36138659  0.65658877]
 [ 0.08452251  0.73016143]
 [-0.85667061 -0.17337266]
 [-0.3582892  -0.07548102]]
```

Step 7: Project the data onto the top k eigenvectors

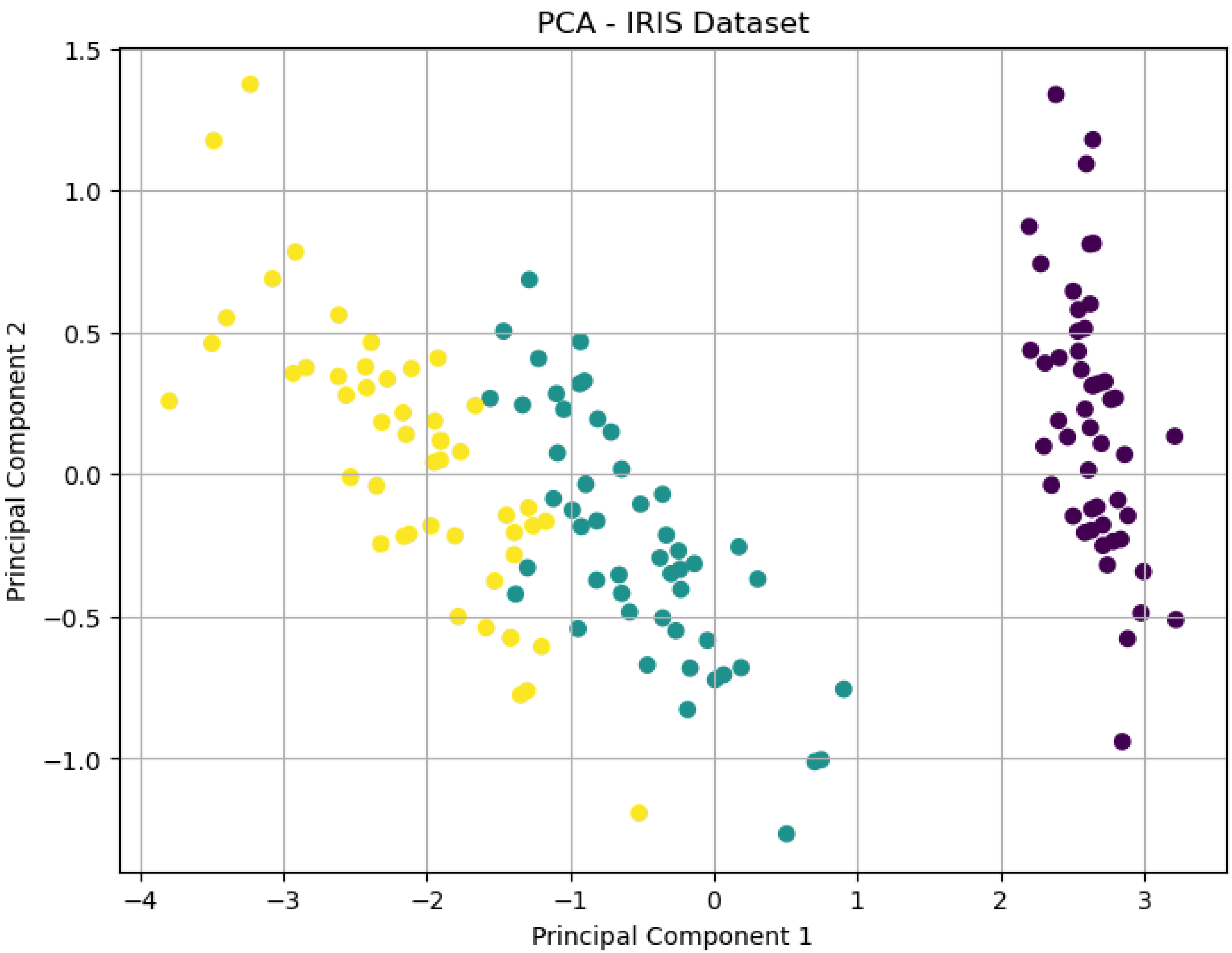
In [12]:

```python
X_reduced = np.dot(X_meaned, eigenvector_subset)
print('Reduced data shape:', X_reduced.shape)
```

```
Reduced data shape: (150, 2)
```

Step 8: Plot the PCA-Reduced Data

In [13]:
```
plt.figure(figsize = (8, 6))
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c = y)
plt.title('PCA - IRIS Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()
```



Extra - Bining Method

5,10,11,13,15,35,50,55,72,92,204,215.

Partition them into three bins by each of the following methods: (a) equal-frequency (equal-depth) partitioning (b) equal-width partitioning

In [14]:

```python
data = [5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215]
data.sort()

print('Sorted Data:', data)

# (a) equal-frequency (equal-depth) partitioning
n = len(data)
k = 3
size = n // k

print('\n(a) Equal-Frequency Bins:')
for i in range(0, n, size):
    bin_data = data[i : i + size]
    print(f'Bin {i // size + 1}:', bin_data)

# (b) equal-width partitioning
min_val = min(data)
max_val = max(data)
range_val = max_val - min_val
width = range_val / k

bins = [[] for _ in range(k)]

for val in data:
    index = int((val - min_val) / width)
    if index == k:  # edge case for the max value
        index -= 1
    bins[index].append(val)
```

```python
print('\n(b) Equal-Width Bins:')
for i, b in enumerate(bins):
    bin_range_start = min_val + i * width
    bin_range_end = bin_range_start + width
    print(f'Bin {i + 1} ({bin_range_start:.2f} to {bin_range_end:.2f}): {b}')
```

```
Sorted Data: [5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215]

(a) Equal-Frequency Bins:
Bin 1: [5, 10, 11, 13]
Bin 2: [15, 35, 50, 55]
Bin 3: [72, 92, 204, 215]

(b) Equal-Width Bins:
Bin 1 (5.00 to 75.00): [5, 10, 11, 13, 15, 35, 50, 55, 72]
Bin 2 (75.00 to 145.00): [92]
Bin 3 (145.00 to 215.00): [204, 215]
```