Dimensionalityreduction

March 10, 2025

[7]:
```
# import libraries
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn import datasets

#Load data
digits = datasets.load_digits()

# standarization
features = StandardScaler().fit_transform(digits.data)

# PCA conserving 99% of variance
pca = PCA(n_components=0.99, whiten=True)

# PCA execution
features_pca = pca.fit_transform(features)

# Print results

print("Initial features :", features.shape[1])
print("Reduced quantity" , features_pca.shape[1] )
```

```
Initial features : 64
Reduced quantity 54
```

[17]:
```
# Feature reduction for data linearly inseparable
# import libraries
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA, KernelPCA
from sklearn.datasets import make_circles


#Create data linearly inseparable
digits = datasets.load_digits()

# standarization
features, _ = make_circles(n_samples=1000, random_state=1, noise=0.1, factor=0.1)
```

```
# Apply PCA avec RBF kernel
kpca = KernelPCA(kernel="rbf" , gamma=15, n_components=1)
features_kpca = kpca.fit_transform(features)

# Print results

print("Initial features :", features.shape[1])
print("Reduced quantity" , features_kpca.shape[1] )
```

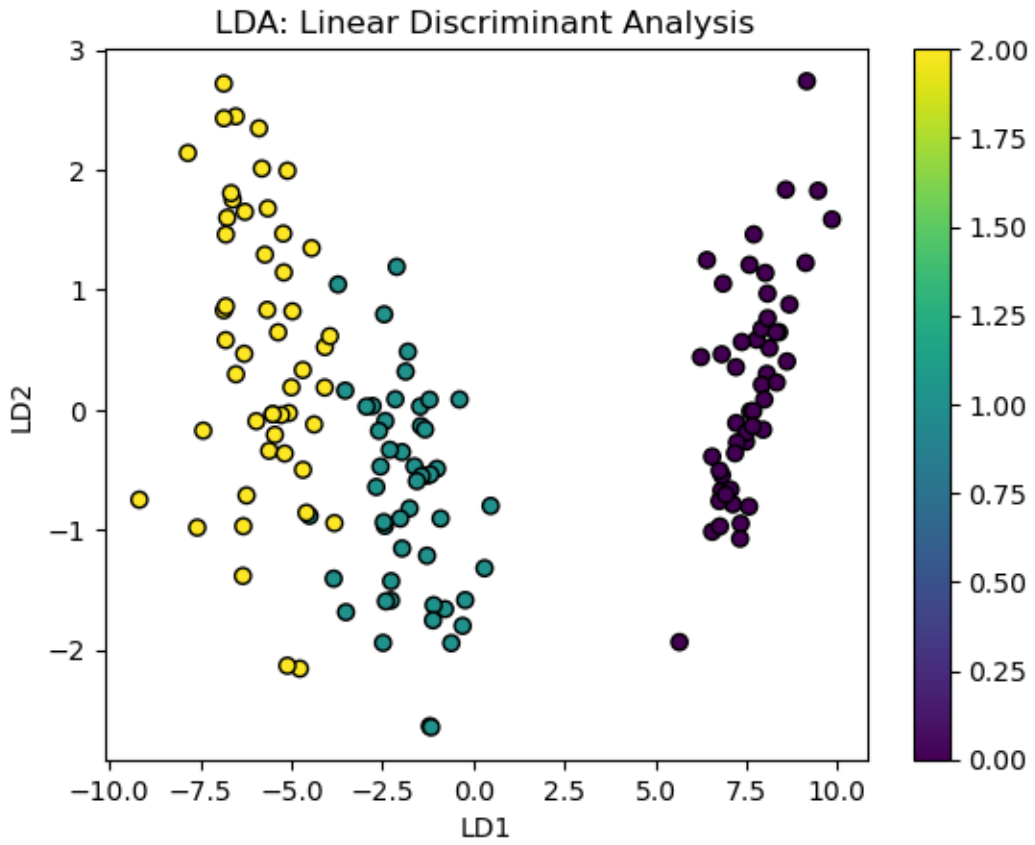
```
Initial features : 2
Reduced quantity 1
```

[21]:
```
# Feature reduction LDA
# import libraries
from sklearn import datasets
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis


#Load data
iris = datasets.load_iris()
features = iris.data
target = iris.target

# Create and apply LDA
lda = LinearDiscriminantAnalysis(n_components=1)
features_lda = lda.fit(features, target).transform(features)

# Print results

print("Initial features :", features.shape[1])
print("Reduced quantity" , features_lda.shape[1] )
```

```
Initial features : 4
Reduced quantity 1
```

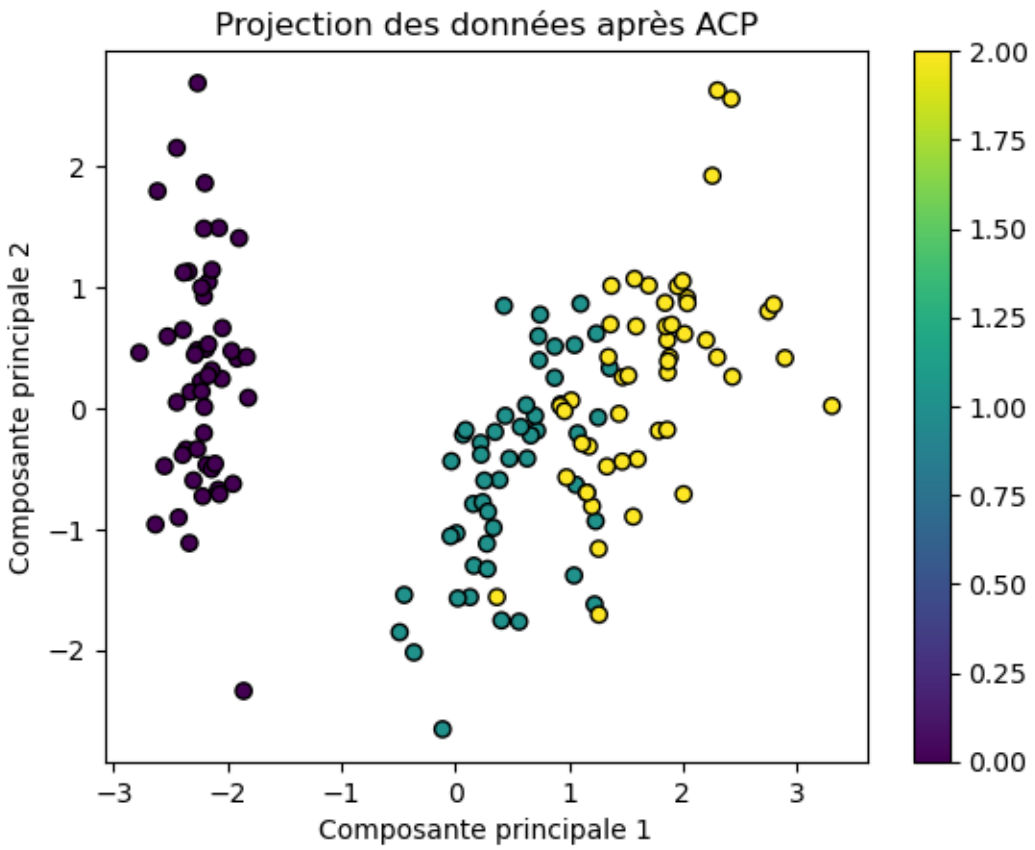
[23]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Standardize the data
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)

# Apply LDA (reduce to 2 components)
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X_scaled, y)

# Plot the results
plt.scatter(X_lda[:, 0], X_lda[:, 1], c=y, cmap='viridis', edgecolor='k')
plt.xlabel('LD1')
plt.ylabel('LD2')
plt.title('LDA: Linear Discriminant Analysis')
plt.colorbar()
plt.show()
```



[25]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
```

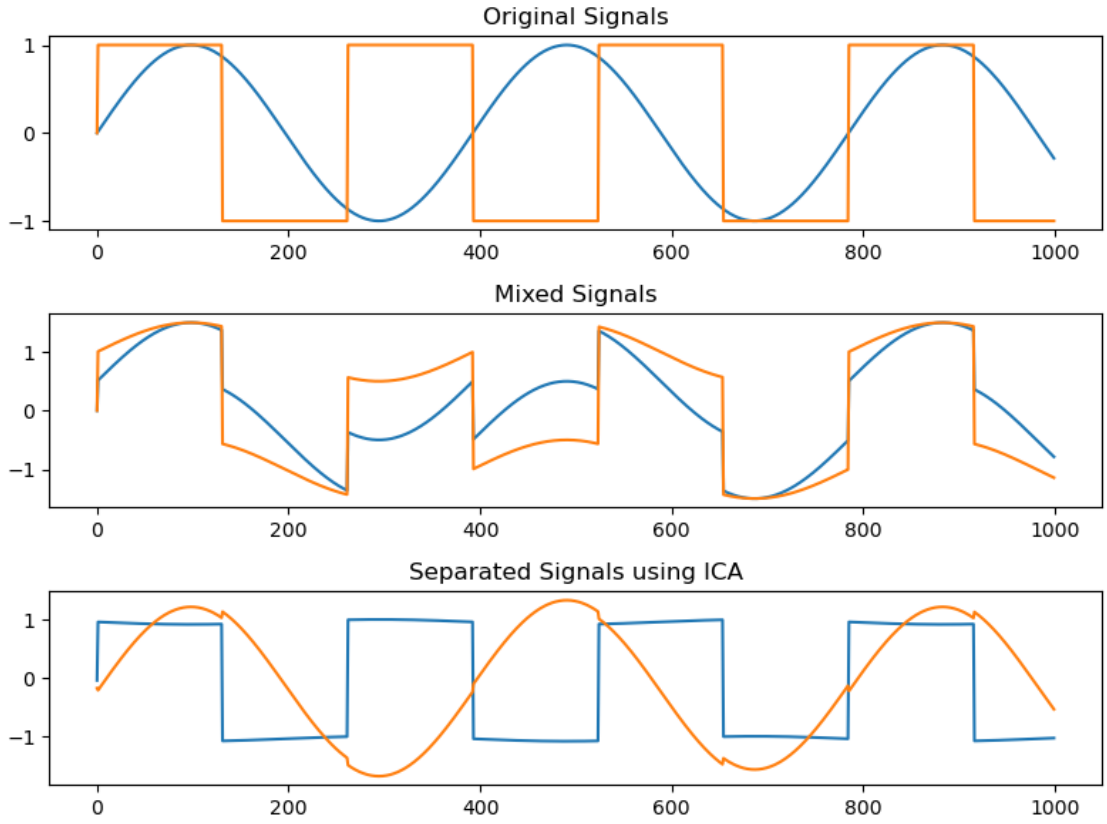
```
# Chargement des données
iris = load_iris()
X = iris.data
y = iris.target

# Standardisation des données
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Application de l'ACP (réduction à 2 composantes principales)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Visualisation des données projetées
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k')
plt.xlabel("Composante principale 1")
plt.ylabel("Composante principale 2")
plt.title("Projection des données après ACP")
plt.colorbar()
plt.show()

# Variance expliquée
print("Variance expliquée par chaque composante :", pca.
 ↪explained_variance_ratio_)
```




```
Variance expliquée par chaque composante : [0.72962445 0.22850762]
```

[27]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import FastICA

# Simulate two independent signals
np.random.seed(0)
time = np.linspace(0, 8, 1000)
s1 = np.sin(2 * time)  # Sinusoidal signal
s2 = np.sign(np.sin(3 * time))  # Square wave

# Mix the signals
S = np.c_[s1, s2]
A = np.array([[1, 0.5], [0.5, 1]])  # Mixing matrix
X = S @ A.T  # Mixed signals

# Apply ICA
ica = FastICA(n_components=2)
S_ica = ica.fit_transform(X)  # Recovered signals
```

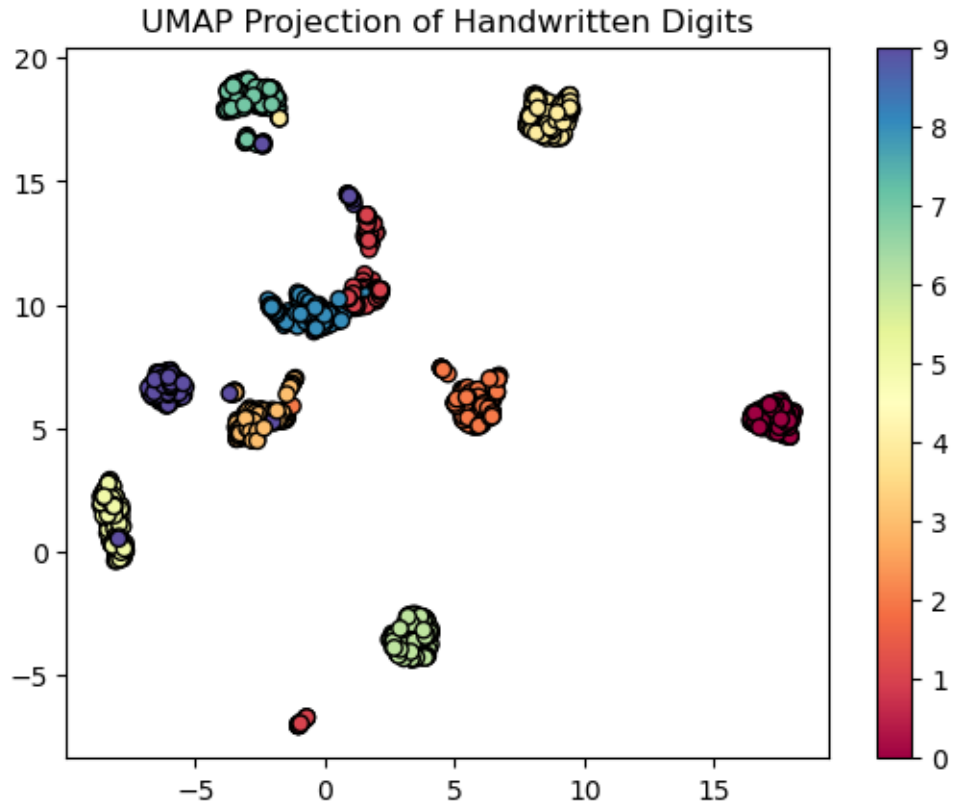
```
# Plot results
fig, axes = plt.subplots(3, 1, figsize=(8, 6))
axes[0].plot(S)
axes[0].set_title("Original Signals")
axes[1].plot(X)
axes[1].set_title("Mixed Signals")
axes[2].plot(S_ica)
axes[2].set_title("Separated Signals using ICA")

plt.tight_layout()
plt.show()
```



[4]:
```
import umap.umap_ as umap
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits

# Load dataset (handwritten digits)
digits = load_digits()
```

```
X, y = digits.data, digits.target

# Apply UMAP for dimensionality reduction
reducer = umap.UMAP(n_components=2, random_state=42)
X_umap = reducer.fit_transform(X)

# Visualization
plt.scatter(X_umap[:, 0], X_umap[:, 1], c=y, cmap='Spectral', edgecolor='k')
plt.colorbar()
plt.title("UMAP Projection of Handwritten Digits")
plt.show()
```

```
C:\Users\Thinkpad\anaconda3\Lib\site-packages\umap\umap_.py:1952: UserWarning:
n_jobs value 1 overridden to 1 by setting random_state. Use no seed for
parallelism.
  warn(
```




```
[ ]:
```