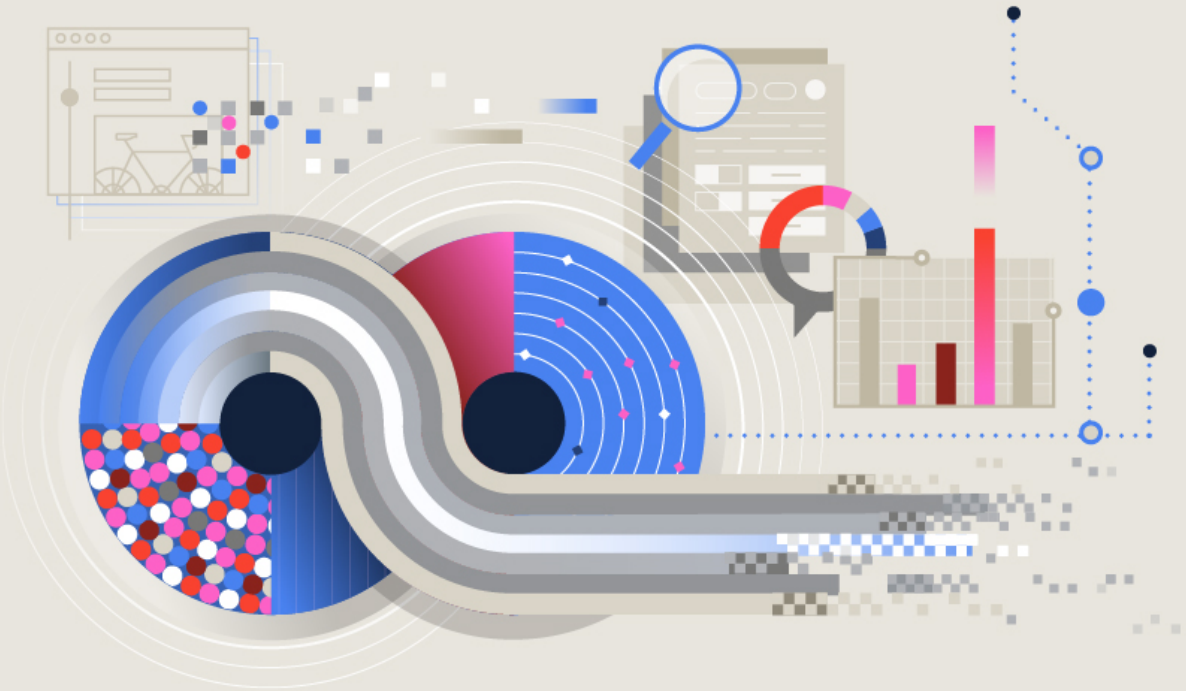
Técnicas de reducción de dimensionalidad: Comparación entre PCA, LDA y KPCA



Por Antonio Richaud

¿Por qué son importantes las técnicas de reducción de dimensionalidad?

En el análisis de datos de alta dimensionalidad, a menudo enfrentamos el desafío de trabajar con grandes volúmenes de características. Aquí es donde las técnicas de reducción de dimensionalidad te pueden ayudar un montón por:

• **Simplificación del conjunto de datos:** Convertir datos de múltiples dimensiones en una representación más compacta facilita su análisis y manipulación.

• **Reducción de la escasez:** Un espacio de características menos disperso tiende a ser más informativo.

• **Prevención del sobreajuste:** Con menos características, los modelos pueden generalizar mejor a datos no vistos.

• **Mejora en la visualización:** Los datos de alta dimensionalidad pueden proyectarse en 2D o 3D, permitiendo una comprensión más intuitiva.

• **Optimización de recursos:** Reducir el número de características disminuye tanto el tiempo de procesamiento como los requisitos de memoria.

• **Mitigación de la 'maldición de la dimensionalidad 👹':** A medida que el número de dimensiones aumenta, los datos se vuelven más dispersos, lo que puede afectar negativamente el rendimiento de los modelos.

• **Descorrelación de características:** Las técnicas de reducción tienden a generar características menos correlacionadas, mejorando la estabilidad del modelo.

• **Filtrado de ruido:** Se eliminan las variaciones menos significativas, mejorando la calidad de los datos.

• **Facilitación de técnicas avanzadas:** Datos reducidos en dimensionalidad son más aptos para algoritmos como los métodos de núcleo (kernel).

Desventajas a considerar:

• **Perdida de interpretabilidad:** Las nuevas características generadas pueden carecer de una interpretación clara en el mundo real.

• **Costo computacional:** Algunos métodos requieren un alto poder de cómputo, especialmente en grandes volúmenes de datos.

• **Pérdida de información:** Durante el proceso, algunos aspectos importantes de los datos pueden perderse.

Comparación de técnicas

Ahora sí a lo que venimos, compararemos PCA, LDA y KPCA, tres de las técnicas más utilizadas para la reducción de dimensionalidad.

Característica	PCA	LDA	KPCA
Tipo	No supervisado	Supervisado	No supervisado
Linealidad	Lineal	Lineal	No lineal
Objetivo principal	Maximizar la varianza	Maximizar la separabilidad de clases	Maximizar la varianza en un espacio de alta dimensionalidad
Uso de la información de clase	No	Sí	No
Escalabilidad	Moderada	Baja para datos de alta dimensionalidad	Baja para grandes conjuntos de datos
Interpretabilidad	Alta	Alta	Baja
Manejo de multicolinealidad	Sí	Sí	Sí
Optimización para clasificación	No	Sí	No
Captura de relaciones no lineales	No	No	Sí
Requiere ajuste de parámetros	No	No	Sí (selección de kernel)
Sensibilidad al escalado de características	Sí	Menos sensible	Depende del kernel

Análisis de Componentes Principales (PCA)

PCA es una técnica no supervisada diseñada para reducir la dimensionalidad de un conjunto de datos, conservando la mayor cantidad posible de variabilidad. Esto se logra transformando los datos en un nuevo espacio donde las dimensiones (componentes principales) son ortogonales entre sí y están ordenadas de acuerdo con la cantidad de varianza que capturan.

Pasos de PCA:

1. Estandarización: Es necesario que las características estén en la misma escala.

2. Cálculo de la matriz de covarianza: La matriz de covarianza resume las relaciones entre las características.

3. Cálculo de los autovalores y autovectores: Los autovalores indican la magnitud de la varianza que cada componente principal captura, y los autovectores definen la dirección de esos componentes.

4. Ordenar los autovectores: Se priorizan aquellos con los autovalores más altos, ya que capturan más varianza.

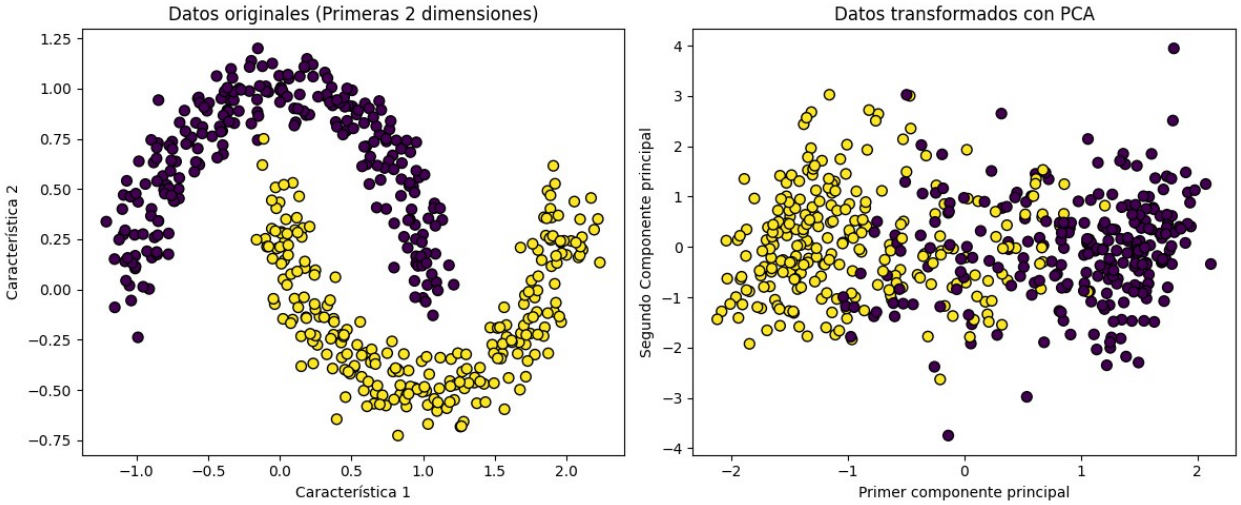
5. Selección de los componentes principales: Elegir los k componentes que retienen la mayor parte de la varianza.

6. Proyección de los datos originales: Finalmente, los datos se proyectan en el nuevo espacio de componentes principales.

Veamos un ejemplo, donde usamos un conjunto de datos simulado de “dos lunas” entrelazadas, al que agregamos ruido. Luego, aplicamos PCA para transformar estos datos a un nuevo espacio de características, donde se reducen a solo dos dimensiones.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler

# Función para generar un conjunto de datos complejo con ruido
def generate_data(n_samples=500):

    # Generar datos con forma de dos lunas entrelazadas (n_samples es el número de puntos)
    X1, y1 = make_moons(n_samples=n_samples, noise=0.1)

    # Agregar ruido aleatorio en 3 dimensiones adicionales para simular datos más complejos
    noise_dims = np.random.randn(n_samples, 3) * 0.1

    # Combinar las lunas con el ruido adicional
    X = np.hstack((X1, noise_dims))

    return X, y1  # Devolver los datos y las etiquetas (aunque las etiquetas no se usan en PCA)

# Generar los datos
X, y = generate_data(n_samples=500)

# Estandarizar las características para que todas tengan la misma escala
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Aplicar PCA para reducir las dimensiones a solo dos componentes principales
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Visualización de los datos originales en las primeras dos dimensiones
plt.figure(figsize=(12, 5))

# Subplot 1: Visualización de los datos originales
plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', s=50, edgecolor='k')
plt.title("Datos originales (Primeras 2 dimensiones)")
plt.xlabel("Característica 1")
plt.ylabel("Característica 2")
```

```
# Subplot 2: Visualización de los datos transformados con PCA
plt.subplot(1, 2, 2)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', s=50,
edgecolor='k')
plt.title("Datos transformados con PCA")
plt.xlabel("Primer componente principal")
plt.ylabel("Segundo Componente principal")

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



Análisis Discriminante Lineal (LDA)

LDA, a diferencia de PCA, es un método supervisado que utiliza la información de las clases para encontrar las combinaciones lineales de características que mejor separen las clases. Esto lo hace ideal para tareas de clasificación, donde la prioridad es maximizar la separabilidad entre clases.

Pasos de LDA:

1. Calcular los vectores de media para cada clase.

2. Calcular las matrices de dispersión intra e inter-clase.

3. Obtener los autovectores y autovalores del producto entre la matriz inversa de dispersión intra-clase y la de inter-clase.

4. Ordenar los autovectores en función de los autovalores decrecientes.

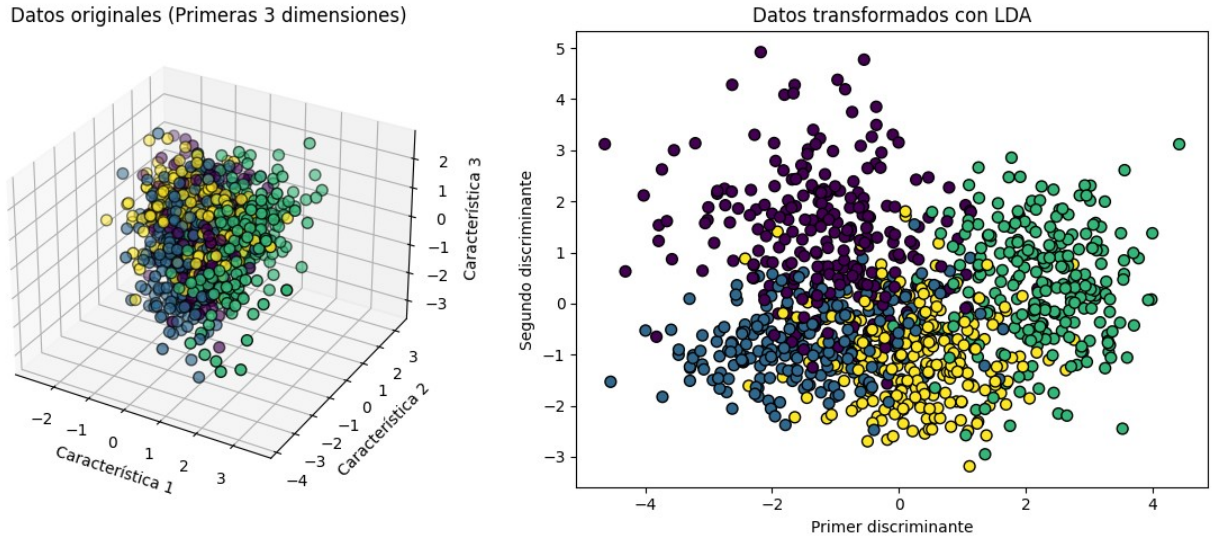
5. Seleccionar los k autovectores principales como nuevo espacio de características.

6. Proyección en el nuevo espacio de características para mejorar la separabilidad entre clases.

Veamos un ejemplo, donde se genera un conjunto de datos artificial con varias características y se utiliza LDA para reducir la dimensionalidad a dos discriminantes principales, optimizando la separación entre las clases. Este código y visualización son una excelente manera de entender cómo el LDA puede ser utilizado para simplificar y mejorar las tareas de clasificación, especialmente cuando tienes muchos atributos en tus datos.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler

# Función para generar un conjunto de datos complejo con ruido
def generate_data(n_samples=500):

    # Generar datos con forma de dos lunas entrelazadas (n_samples es
el número de puntos)
    X1, y1 = make_moons(n_samples=n_samples, noise=0.1)

    # Agregar ruido aleatorio en 3 dimensiones adicionales para
simular datos más complejos
    noise_dims = np.random.randn(n_samples, 3) * 0.1

    # Combinar las lunas con el ruido adicional
    X = np.hstack((X1, noise_dims))

    return X, y1  # Devolver los datos y las etiquetas (aunque las
etiquetas no se usan en PCA)

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import make_classification
from sklearn.preprocessing import StandardScaler

# Función para generar un conjunto de datos complejo para LDA
def generate_complex_data_lda(n_samples=1000, n_features=20,
n_classes=4):
    # Genera un conjunto de datos de clasificación con múltiples
características
    X, y = make_classification(n_samples=n_samples,
n_features=n_features, n_classes=n_classes,
                               n_informative=10, n_redundant=5,
n_repeated=3,
                               n_clusters_per_class=2, class_sep=1.5,
random_state=42)

    return X, y  # Retorna las características X y las etiquetas y

# Generar los datos complejos
X, y = generate_complex_data_lda(n_samples=1000)

# Estandarizar las características
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Aplicar LDA para reducir las dimensiones a 2 discriminantes
principales
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X_scaled, y)

# Visualización de los datos originales en las primeras 3 dimensiones
fig = plt.figure(figsize=(12, 5))

# Subplot 1: Datos originales en 3D (primeras 3 características)
ax = fig.add_subplot(121, projection='3d')
ax.scatter(X_scaled[:, 0], X_scaled[:, 1], X_scaled[:, 2], c=y,
cmap='viridis', s=50, edgecolor='k')
ax.set_title("Datos originales (Primeras 3 dimensiones)")
ax.set_xlabel("Característica 1")
ax.set_ylabel("Característica 2")
ax.set_zlabel("Característica 3")

# Subplot 2: Visualización de los datos transformados con LDA
plt.subplot(122)
plt.scatter(X_lda[:, 0], X_lda[:, 1], c=y, cmap='viridis', s=50,
edgecolor='k')
plt.title("Datos transformados con LDA")
plt.xlabel("Primer discriminante")
plt.ylabel("Segundo discriminante")

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



Análisis de Componentes Principales con Kernel (KPCA)

KPCA es una extensión no lineal del PCA que utiliza métodos de kernel para realizar la reducción de dimensionalidad en espacios de características de alta dimensionalidad. Este enfoque es particularmente útil para captar relaciones no lineales en los datos, algo que PCA tradicional no puede hacer.

¿Por qué utilizar KPCA?

KPCA es especialmente adecuado cuando se trabaja con datos que presentan estructuras no lineales. Al proyectar los datos en un espacio de mayor dimensionalidad mediante un kernel, se pueden capturar patrones complejos que de otra manera serían invisibles para métodos lineales como PCA.

Pasos de KPCA:

1. Seleccionar una función kernel: Esto es crucial para KPCA. Entre las más comunes están:

- **Gaussian (RBF):** Ideal para capturar relaciones locales entre los puntos de datos.
- **Polinomial:** Captura relaciones más globales, dependiendo del grado del polinomio.

2. Calcular la matriz kernel: La matriz kernel mide las similitudes entre cada par de puntos en el espacio original, pero proyectados en un espacio de alta dimensionalidad a través del truco del kernel.

3. Centrar la matriz kernel: Para asegurar que los datos estén correctamente centrados, se resta la media de la matriz kernel. Este paso es similar a centrar los datos en PCA tradicional.

4. Calcular los autovectores y autovalores: A partir de la matriz kernel centrada, se calculan los autovectores (direcciones principales) y los autovalores (magnitudes de la varianza capturada).

5. Ordenar los autovectores por autovalores decrecientes: Los autovectores asociados con los autovalores más grandes capturan la mayor cantidad de varianza. Estos autovectores se seleccionan como las nuevas dimensiones.

6. Elegir los primeros k autovectores como el nuevo espacio de características: Los componentes principales seleccionados se convierten en el nuevo espacio de características.

7. Proyectar los datos originales en el nuevo espacio usando el truco del kernel: Los datos originales se proyectan en el espacio reducido mediante los autovectores calculados, capturando la mayor parte de la variabilidad, incluidas las relaciones no lineales.

Ejemplo de KPCA con RBF Kernel

Vamos a ver un ejemplo muestra de cómo KPCA puede desenrollar estructuras no lineales, como un conjunto de datos en forma de “rollo suizo” (Swiss roll), que PCA tradicional no podría manejar correctamente. (Esta en la siguiente pagina :b).

```python
import numpy as np
from sklearn.decomposition import KernelPCA
import matplotlib.pyplot as plt
from sklearn.datasets import make_swiss_roll
from mpl_toolkits.mplot3d import Axes3D

# Generar datos Swiss roll (conjunto de datos en forma de rollo suizo)
n_samples = 1500  # Número de muestras
noise = 0.05  # Añadir algo de ruido para hacerlo más realista
X, color = make_swiss_roll(n_samples=n_samples, noise=noise)

# Aplicar KPCA con kernel RBF (Radial Basis Function)
kpca = KernelPCA(n_components=2, kernel='rbf', gamma=0.002)
X_kpca = kpca.fit_transform(X)

# Visualización de los datos originales Swiss roll en 3D
fig = plt.figure(figsize=(12, 5))

# Subplot 1: Visualización de los datos originales en 3D
ax = fig.add_subplot(121, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=color, cmap='viridis', s=50)
ax.set_title("Datos originales swiss roll")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")

# Subplot 2: Visualización de los datos transformados con KPCA
plt.subplot(122)
plt.scatter(X_kpca[:, 0], X_kpca[:, 1], c=color, cmap='viridis', s=50,
edgecolor='k')
plt.title("Datos transformados con KPCA")
plt.xlabel("Primer componente principal")
plt.ylabel("Segundo componente principal")

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