

SESIÓN TÉCNICAS DE REDUCCIÓN DIMENSIONAL

CONTENIDOS:

- Pca y t-sne: análisis de componentes principales
 - (pca): en qué consiste el método pca.
 - Aplicaciones de este método.
 - Ventajas y desventajas de este método.
 - Implementación en Python.
 - T-distributed stochastic neighbor embedding (t-sne): en qué consiste el método t-sne.
 - Aplicaciones de este método.
 - Ventajas y desventajas de este método.
 - Implementación en Python.

PCA Y T-SNE: ANÁLISIS DE COMPONENTES PRINCIPALES

En el análisis de datos, es común trabajar con conjuntos de datos de alta dimensión, lo que dificulta su visualización y procesamiento. Para abordar este problema, existen técnicas de reducción de dimensionalidad como PCA y t-SNE. Estas herramientas permiten representar los datos en un espacio de menor dimensión mientras conservan su estructura y relaciones más relevantes.

PCA: ¿EN QUÉ CONSISTE ESTE MÉTODO?

PCA (Análisis de Componentes Principales, por sus siglas en inglés: **Principal Component Analysis**) es un método estadístico utilizado para reducir la dimensionalidad de un conjunto de datos mientras se conserva la mayor cantidad de información posible.

En términos sencillos, si tienes un conjunto de datos con muchas variables (*features*), PCA te ayuda a transformarlo en un conjunto más pequeño de variables que aún capturan la mayor parte de la variabilidad de los datos originales.

¿En qué consiste PCA?

PCA encuentra nuevas variables llamadas **componentes principales**, que son combinaciones lineales de las variables originales. Estas componentes principales tienen las siguientes características:

1. **Son ortogonales entre sí** (no están correlacionadas).
2. **Se ordenan por importancia**, es decir, la primera componente principal captura la mayor parte de la variabilidad de los datos, la segunda un poco menos, y así sucesivamente.
3. **Permiten reducir la cantidad de dimensiones**, eliminando aquellas que aportan poca variabilidad.

El proceso general de PCA es:

1. **Estandarización de los datos** (si es necesario).
2. **Cálculo de la matriz de covarianza** (para entender la relación entre variables).
3. **Cálculo de los valores y vectores propios** (para determinar los componentes principales).
4. **Proyección de los datos** en el nuevo espacio de dimensiones reducidas.

Aplicaciones de PCA

PCA se usa en muchos campos, especialmente cuando se trabaja con datos de alta dimensionalidad. Algunas aplicaciones son:

- **Reconocimiento facial:** Se usa para reducir la cantidad de datos en imágenes y extraer características relevantes.
- **Compresión de imágenes:** PCA permite almacenar imágenes con menos información sin perder demasiada calidad.
- **Análisis de datos financieros:** Para reducir el número de indicadores usados en modelos de inversión o análisis de riesgo.
- **Genómica y bioinformática:** Para reducir el número de variables en el análisis de expresión genética.

- **Procesamiento de texto:** En NLP (*Natural Language Processing*), PCA se usa para reducir la dimensionalidad de datos representados en modelos como TF-IDF.

Ventajas y desventajas de PCA

A continuación, veamos una tabla con las ventajas y desventajas de PCA:

Ventajas	Desventajas
Reduce la dimensionalidad de los datos, lo que facilita el procesamiento y almacenamiento.	Puede ser difícil interpretar los componentes principales, ya que son combinaciones de variables originales.
Elimina la multicolinealidad (correlación entre variables) al generar componentes ortogonales.	Al reducir la dimensionalidad, se pierde algo de información.
Puede mejorar el rendimiento de algoritmos de aprendizaje automático eliminando ruido y redundancia.	Supone que las relaciones entre las variables son lineales, lo que no siempre es cierto.
Facilita la visualización de datos cuando se reduce a 2D o 3D.	

IMPLEMENTACIÓN EN PYTHON: Reducción de Dimensionalidad en Datos de Imágenes de Dígitos

Para aplicar PCA en un caso práctico, usaremos el conjunto de datos *Digits* de Scikit-learn, que contiene imágenes de números escritos a mano (del 0 al 9) con dimensiones de 8x8 píxeles (64 características por imagen).

Pasos que seguiremos en Python

1. Cargar el **dataset Digits** de **Scikit-learn**.
2. Visualizar algunas imágenes originales.

3. Aplicar PCA para reducir la dimensionalidad de 64 a 2 dimensiones.
4. Visualizar los datos reducidos en un gráfico de dispersión.
5. Evaluar la cantidad de varianza retenida.

Ahora, pasemos a la implementación en Python.

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

# 1. Cargar el dataset Digits
digits = load_digits()
X = digits.data # Matriz de características (imágenes en forma de vectores)
y = digits.target # Etiquetas (números reales de las imágenes)

# 2. Visualizar algunas imágenes originales
fig, axes = plt.subplots(1, 5, figsize=(10, 3))
for i, ax in enumerate(axes):
    ax.imshow(digits.images[i], cmap='gray')
    ax.set_title(f'Dígito: {digits.target[i]}')
    ax.axis('off')
plt.show()

# 3. Normalizar los datos antes de PCA
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```


Ilustración 1 Parte 1 de la implementación

```
# 4. Aplicar PCA para reducir la dimensionalidad a 2 componentes
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# 5. Visualizar los datos en el nuevo espacio de 2D
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', alpha=0.7)
plt.colorbar(scatter, label='Dígito real')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('Visualización de PCA en el conjunto de datos Digits')
plt.show()

# 6. Evaluar la varianza explicada por cada componente
explained_variance = pca.explained_variance_ratio_
print(f'Varianza explicada por la primera componente: {explained_variance[0]:.2%}')
print(f'Varianza explicada por la segunda componente: {explained_variance[1]:.2%}')
```

Ilustración 2 Parte 2 de la implementación



Este código toma el conjunto de datos de dígitos escritos a mano, lo normaliza, aplica PCA para reducirlo a 2 dimensiones y luego visualiza los resultados en un gráfico de dispersión. También imprime la cantidad de varianza explicada por cada componente principal.

T-DISTRIBUTED STOCHASTIC NEIGHBOR EMBEDDING (T-SNE)

t-SNE es un método de reducción de dimensionalidad no lineal utilizado principalmente para la visualización de datos de alta dimensión en espacios de 2D o 3D. Fue desarrollado por **Geoffrey Hinton y Laurens van der Maaten** en 2008 y es especialmente útil cuando los datos tienen estructuras complejas que los métodos lineales (como PCA) no pueden captar.

¿En qué consiste t-SNE?

A diferencia de PCA, que busca preservar la varianza de los datos al transformarlos en combinaciones lineales, t-SNE está diseñado para preservar la relación de proximidad entre puntos similares. Esto significa que los puntos cercanos en el espacio original también deberían estar cercanos en el espacio reducido.

El algoritmo funciona en dos fases principales:

1. Cálculo de probabilidades de similitud:

- Se calcula la probabilidad de que dos puntos sean vecinos en el espacio original mediante distribuciones de probabilidad de Gauss.
- Luego, se hace lo mismo en el espacio reducido, pero usando una distribución *t-Student* para modelar las distancias.

2. Optimización para minimizar la divergencia entre ambas distribuciones:

- Se usa el método de *gradient descent* para minimizar la diferencia entre ambas distribuciones, reorganizando los puntos en el espacio reducido.

El resultado es que los puntos que estaban cerca en el espacio original se agruparán en clusters en el espacio reducido, facilitando la visualización.

Aplicaciones de t-SNE

t-SNE es ampliamente usado en escenarios donde los datos tienen estructuras no lineales y queremos entender mejor su agrupación. Algunos casos de uso incluyen:

- **Análisis de datos genómicos:** Para identificar relaciones entre distintos tipos de células o genes.
- **Procesamiento de imágenes:** Ayuda a visualizar la organización de imágenes en función de características aprendidas.
- **Procesamiento de texto (NLP):** Se usa para visualizar embeddings de palabras como *word2vec* o *BERT*.
- **Aprendizaje automático:** Permite entender cómo se agrupan los datos antes de aplicar modelos de clasificación.
- **Detección de anomalías:** Puede revelar patrones en datos de seguridad o fraudes financieros.

Ventajas y desventajas de t-SNE

A continuación, veamos una tabla con las ventajas y desventajas de t-SNE:

Ventajas	Desventajas
Es muy eficaz para encontrar patrones y estructuras ocultas en datos de alta dimensionalidad.	Es computacionalmente costoso en comparación con otros métodos como PCA.
Mantiene relaciones locales entre los puntos, lo que facilita la agrupación de datos similares.	Puede ser difícil de interpretar, ya que la escala de los ejes en t-SNE no tiene un significado claro.
Es muy útil para visualizar datos en 2D o 3D sin perder demasiada información.	Los resultados pueden variar en cada ejecución debido a su inicialización aleatoria.
	No es adecuado para reducción de dimensionalidad antes de aplicar modelos de

	aprendizaje automático, ya que se enfoca en visualización más que en preservar información para predicción.
--	---

IMPLEMENTACIÓN EN PYTHON: Visualización de datos de dígitos con t-SNE

Vamos a aplicar t-SNE al mismo conjunto de datos **Digits** de **Scikit-learn** para visualizar cómo se agrupan los números escritos a mano.

Pasos que seguiremos en Python

1. Cargar el dataset Digits.
2. Estandarizar los datos (para mejorar el rendimiento de t-SNE).
3. Aplicar t-SNE para reducir la dimensionalidad a 2D.
4. Visualizar los datos reducidos en un gráfico de dispersión.

Ahora, pasemos al código.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler


# 1. Cargar el dataset Digits
digits = load_digits()
X = digits.data # Matriz de características
y = digits.target # Etiquetas (dígitos reales)

# 2. Normalizar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. Aplicar t-SNE para reducir la dimensionalidad a 2D
tsne = TSNE(n_components=2, perplexity=30, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)

# 4. Visualizar los datos en el nuevo espacio 2D
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='jet', alpha=0.7)
plt.colorbar(scatter, label='Dígito real')
plt.xlabel('Dimensión 1')
plt.ylabel('Dimensión 2')
plt.title('Visualización con t-SNE del conjunto de datos Digits')
plt.show()
```

Ilustración 3 Implementación en Python



Este código utiliza t-SNE para reducir la dimensionalidad del conjunto de datos de dígitos escritos a mano y visualizar la agrupación de los números en un gráfico de dispersión.