Principal Component Analysis

Giới thiệu

Khi dữ liệu có nhiều đặc trưng (10, 50, 1000...), chúng ta không thể trực quan hóa trực tiếp. PCA giúp giảm số chiều của dữ liệu mà vẫn giữ các thông tin quan trọng.

Khái niệm chính:

- PCA tìm các **trục mới** (principal components) sao cho dữ liệu trên các trục này vẫn giữ được thông tin quan trọng.
- Nó cho phép **giảm từ nhiều tính năng xuống 2–3 chiều**, thuận tiện để vẽ đồ thị và trực quan hóa.

Algorithm

- PCA tạo các **trục mới (principal components)** để chiếu dữ liệu, tối đa hóa **phương sai** dữ liệu trên các trục này.
- Khác với hồi quy tuyến tính, PCA là **học không giám sát**, không dùng nhãn y, chỉ làm việc với các đặc trưng X.

Tiền xử lý dữ liệu

- Chuẩn hóa các tính năng để có **giá trị trung bình bằng 0**.
- Nếu các tính năng có **tỷ lệ khác nhau**, thực hiện **scaling** để cân bằng phạm vi giá trị.

Ý tưởng trực quan với dữ liệu 2D

- Giả sử dữ liệu có hai đặc trưng x_1, x_2 và 5 ví dụ đào tạo.
- PCA tìm một **trục mới z** sao cho khi chiếu dữ liệu lên z:
  - **Giữ được nhiều phương sai nhất** (dữ liệu trải rộng tốt nhất trên trục này).
  - Các điểm dữ liệu chiếu lên z tách biệt nhau càng nhiều càng tốt.
- **Chiếu dữ liệu lên trục z**:
  - Vẽ đoạn thẳng vuông góc từ mỗi điểm đến trục z.

- Giá trị trên trục z chính là tọa độ mới của điểm (số duy nhất nếu giảm xuống 1 chiều).

Các trục tiếp theo

- Trục thứ hai z2z_2z2 vuông góc 90° với trục đầu tiên z1z_1z1.
- Nếu có nhiều tính năng (ví dụ 50), các thành phần chính tiếp theo cũng vuông góc với tất cả trục trước đó.
- Giảm dữ liệu xuống 2–3 thành phần chính thường đủ để **trực quan hóa** dữ liệu.

Cài đặt với Sklearn

Chuẩn bị dữ liệu

- **Kiểm tra phạm vi giá trị của các tính năng**. Nếu các feature rất khác nhau, nên **scale** hoặc **chuẩn hóa** trước khi áp dụng PCA để thuật toán chọn trục hợp lý.
- PCA trong scikit-learn sẽ **tự động trừ đi giá trị trung bình** của từng tính năng, nên bạn không cần mean-normalization riêng.

Chạy PCA bằng scikit-learn

- Tạo đối tượng `PCA(n_components=k)` với `k` = số thành phần chính muốn giữ (thường là 2 hoặc 3 để trực quan hóa).
- Gọi `fit(X)` để "học" các trục chính (Z_1, Z_2, Z_3...).
- Kiểm tra `explained_variance_ratio_` để biết mỗi thành phần giữ bao nhiêu phần trăm phương sai của dữ liệu.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=1)   # giảm từ nD xuống 1D
pca.fit(X)
print(pca.explained_variance_ratio_)   # [0.992]
Z = pca.transform(X)   # chiếu dữ liệu xuống trục Z
```

- `Z` là dữ liệu sau khi chiếu lên các trục mới → mỗi mẫu chỉ còn k số (k = số thành phần).

- Có thể **vẽ Z** (2D/3D) để nhìn cấu trúc, nhóm, bất thường.
- Nếu k = số chiều gốc → PCA trả lại dữ liệu gần như y hệt (không giảm).

Một số ứng dụng và lưu ý

- **Trực quan hóa dữ liệu** (mục tiêu chính ngày nay).
- **Nén dữ liệu** hoặc **tăng tốc huấn luyện** (phổ biến 10–20 năm trước, giờ ít dùng vì phần cứng và thuật toán mới đủ nhanh).
- PCA cũng tiêu tốn chi phí tính toán, nên nếu dùng mạng nơ-ron hiện đại, thường không cần PCA trước.