K-nearest Neighbor

Time Series Team

Ngày 12 tháng 8 năm 2025

Mục lục

Phần I: Lý Thuyết

1 Machine Learning: Review

Machine Learning (ML) là lĩnh vực nghiên cứu giúp máy tính học từ dữ liệu và tự động cải thiện hiệu suất của mình theo thời gian. Các giải thuật ML có thể được phân thành ba nhóm chính:

- **Supervised Learning (Học có giám sát):** Supervised Learning hoạt động trên nguyên tắc: có Input và biết trước Output. Mục tiêu là tìm mối liên hệ giữa Input và Output dựa trên dữ liệu huấn luyện. Khi mô hình đưa ra dự đoán, ta có thể đánh giá độ chính xác và điều chỉnh tham số để tối ưu hóa kết quả.
  Ví dụ:
  - Dự đoán bệnh ung thư từ ảnh chụp X-quang: Dữ liệu huấn luyện bao gồm hàng nghìn ảnh X-quang có nhãn (bệnh hoặc không bệnh). Mô hình học cách nhận diện các dấu hiệu ung thư để dự đoán chính xác trên ảnh mới.

  Giả sử với Supervised Learning, ta đã có ngầm định rằng tồn tại "True underlying relationship"giữa Y (Output) và X (Input). Có một số cách tiếp cận phổ biến:
  - **Parametric Approach:** Mô hình hóa mối quan hệ giữa Input và Output thành các tham số cụ thể, biểu diễn dưới dạng công thức toán học.
  - **Eager Learning:** Quá trình học diễn ra rất lâu nhưng khi test thì rất nhanh. Ví dụ như bài toán dự đoán giá nhà, giá vàng: "It takes long time learning and less time classifying data". Giống như việc dành nhiều thời gian để ôn thi nhưng khi thi thì làm bài nhanh chóng.
  - **Model-based Learning Approach:** Tiếp cận dựa trên mô hình đã được xây dựng sẵn để đưa ra dự đoán.

- **Unsupervised Learning (Học không giám sát):** Ở Unsupervised Learning, ta chỉ có dữ liệu đầu vào mà không có nhãn Output. Mục tiêu chính là phân nhóm dữ liệu dựa trên các đặc điểm chung. **Ví dụ:**
  - Phân nhóm bệnh nhân theo triệu chứng: Bệnh nhân có thể có những đặc điểm tương đồng chưa được nhận biết rõ ràng. Thuật toán như K-Means sẽ giúp nhóm bệnh nhân có triệu chứng tương tự mà không cần biết trước họ mắc bệnh gì.

- **Reinforcement Learning (Học tăng cường):** Reinforcement Learning (RL) được sử dụng khi mô hình phải đưa ra quyết định trong môi trường thay đổi. Hệ thống học bằng cách thử nghiệm (trial & error), nhận thưởng khi hành động đúng và bị phạt khi hành động sai. **Ví dụ:**
  - Điều khiển robot phẫu thuật: RL giúp robot học cách thao tác chính xác dựa trên phản hồi từ môi trường thực tế, tối ưu hóa chuyển động để giảm rủi ro trong quá trình mổ.

- **Semi-supervised Learning (Học bán giám sát):** Semi-supervised Learning là sự kết hợp giữa Supervised và Unsupervised Learning. Khi dữ liệu có nhãn quá ít nhưng dữ liệu chưa nhãn lại nhiều, ta dùng Unsupervised Learning để nhóm dữ liệu trước, sau đó dùng một phần dữ liệu có nhãn để huấn luyện mô hình Supervised Learning. **Ví dụ:**
  - Chẩn đoán bệnh dựa trên ảnh y tế: Chỉ một phần nhỏ ảnh chụp MRI được bác sĩ gán nhãn, phần còn lại được mô hình Unsupervised học để nhóm các đặc điểm giống nhau, sau đó tinh chỉnh bằng dữ liệu có nhãn.

AI đang cách mạng hóa lĩnh vực hình ảnh y tế bằng cách hỗ trợ chẩn đoán nhanh và chính xác hơn. Các thuật toán ML giúp nhận diện tổn thương, dự đoán nguy cơ bệnh và hỗ trợ bác sĩ đưa ra quyết định.

Một trong những thuật toán đơn giản nhưng hiệu quả trong chẩn đoán y tế là **K-Nearest Neighbors (KNN)**. KNN hoạt động dựa trên nguyên tắc so sánh điểm dữ liệu mới với các điểm dữ liệu đã biết để tìm ra nhãn phù hợp nhất. Chẳng hạn, nếu một bệnh nhân có hình ảnh MRI tương đồng với một nhóm bệnh nhân ung thư trước đó, thuật toán sẽ gán nhãn tương tự.

KNN là một dạng **Lazy Learning**, tức là không thực hiện quá trình học trước mà chỉ lưu trữ dữ liệu và đưa ra quyết định khi có yêu cầu. Điều này khiến nó dễ triển khai nhưng có thể chậm khi số lượng dữ liệu lớn.

Ở phần tiếp theo, chúng ta sẽ đi sâu vào **KNN Motivation**, tại sao KNN lại là một phương pháp phù hợp với bài toán phân loại trong y tế và cách thức hoạt động của nó.

2 KNN Motivation

Hiện nay, các nhà nghiên cứu đặt ra một giả thuyết như sau: ***Liệu có giải pháp nào không cần phải train mô hình trước, chỉ cần lưu trữ dữ liệu và khi cần sử dụng, ta chỉ cần truy vấn dữ liệu và đưa ra kết quả ngay lập tức?***

2.1 Lazy Learning vs. Eager Learning

Ban đầu, hầu hết các phương pháp học máy (Machine Learning) đều ngầm định rằng có một mối quan hệ giữa đầu vào X và đầu ra Y, từ đó tiến hành huấn luyện mô hình để tìm ra quy luật này. Tuy nhiên, một hướng tiếp cận khác đặt ra câu hỏi: **Có cách nào mà không cần quan tâm đến bản chất của dữ liệu (linear hay nonlinear)?** Thay vào đó, ta chỉ cần có dữ liệu, rồi khi có input mới, ta thiết kế thuật toán để truy vấn và dự đoán kết quả ngay lập tức mà không mất nhiều thời gian huấn luyện.

Cách tiếp cận trên được gọi là **Lazy Learning**.

Để dễ hình dung, giả sử ta có một kho dữ liệu gồm 10.000 bức ảnh cần phân loại. Cách tiếp cận truyền thống sẽ là:

- Thiết kế thuật toán và huấn luyện mô hình (mất khoảng 10 tiếng).
- Sau khi huấn luyện xong, việc dự đoán cho một bức ảnh mới chỉ mất 30ms.

Cách này được gọi là **Eager Learning**: mô hình sẽ học kỹ từ trước, mất nhiều thời gian train nhưng khi test lại rất nhanh.

Ngược lại, có một cách tiếp cận khác: **Liệu có phương pháp nào giúp ta không cần học trước nhưng vẫn cho ra kết quả chính xác?** Tức là chỉ cần có dữ liệu, khi có input mới, ta sẽ truy vấn trực tiếp vào dữ liệu để đưa ra dự đoán. Đây chính là ý tưởng đằng sau KNN, còn được gọi là **Lazy Motivation**: **không cần học trước nhưng vẫn có thể dự đoán chính xác.**

2.2 Ví dụ Minh Họa: Sinh Viên Thi Cuối Kỳ

Hãy lấy một ví dụ đơn giản về sinh viên thi cuối kỳ môn Toán:

- **Eager Learning (Học nhiều, thi nhanh):**Một nhóm sinh viên chăm chỉ học kỹ toàn bộ nội dung trong sách trước kỳ thi. Khi vào phòng thi, họ không cần tham khảo tài liệu, cứ thế viết bài nhờ vào kiến thức đã ghi nhớ. Đây giống như cách tiếp cận truyền thống trong Machine Learning: tốn thời gian huấn luyện nhưng khi sử dụng thì rất nhanh.

- **Lazy Learning (Không học trước, tìm kiếm nhanh trong lúc thi):**MMột nhóm sinh viên khác lười học nhưng được phép mang sách vào phòng thi. Giả sử thầy giáo sử dụng 10 cuốn sách để ra đề, nhóm này sẽ không học trước mà chỉ tập trung vào kỹ năng tìm kiếm nhanh trong tài liệu. Khi gặp một câu hỏi, họ tra ngay vào cuốn sách nào chứa nội dung liên quan và tìm đáp án.

KNN hoạt động theo cách của nhóm sinh viên lười: **không học trước, nhưng có kỹ năng tìm kiếm và đối chiếu nhanh**. Nếu sách quá nhiều và không biết cách tổ chứctốt, sẽ không thể tìm được đáp án kịp thời. Tương tự, KNN cần một chiến lược hiệu quả để tìm kiếm trong dữ liệu một cách tối ưu.

2.3 KNN - Một Phương Pháp Instance-based Learning

KNN thuộc nhóm **Lazy Learning**, nghĩa là không cần train trước nhưng vẫn có thể đưa ra dự đoán chính xác. Nó còn được gọi là **Instancebased Learning** có thể được ví như một sinh viên mang toàn bộ tài liệu vào phòng thi và tra cứu để trả lời thay vì ghi nhớ kiến thức trước đó.

Cách tiếp cận này rất hữu ích trong nhiều bài toán, đặc biệt là khi mô hình huấn luyện mất quá nhiều thời gian. KNN giúp đánh giá nhanh xem dữ liệu có phù hợp với một thuật toán Machine Learning hoặc Deep Learning khác hay không, trước khi quyết định lựa chọn phương pháp phù hợp.

2.4 KNN: Cách Hoạt Động

Ví dụ 1: Giả sử ta có một tập dữ liệu gồm hình ảnh **chó** và **mèo**. Khi nhận một bức ảnh mới, câu hỏi đặt ra là: **Nó giống chó hay mèo hơn?**

- Nếu bức ảnh mới giống mèo hơn, ta phân loại nó là mèo.
- Để làm điều này, ta cần một phép đo cụ thể, ví dụ **độ tương đồng cosine** (cosine similarity) hoặc **khoảng cách Euclide**.
- Giả sử ảnh mới có độ tương đồng với mèo là 0.6, với chó là 0.2. Vì $0.6 > 0.2$, nên ảnh mới được phân loại là mèo mà không cần train mô hình trước.

Ví dụ 2: Giả sử tập dữ liệu gồm các loài **ong, sâu, chuồn chuồn, gà, mèo**. Khi có một ảnh mới, ta muốn biết nó thuộc loại nào. KNN sẽ **đo khoảng cách** từ ảnh mới đến các ảnh trong tập dữ liệu và tìm nhóm gần nhất.

- Nếu đặt tham số $\mathbf{k = 5}$, ta tìm 5 ảnh gần nhất với ảnh đầu vào.
- Nếu 3 ảnh trong số đó là mèo, 2 ảnh còn lại là chuồn chuồn, ta phân loại ảnh mới là mèo.
- Nếu tăng $\mathbf{k = 7}$, và có 4 ảnh chuồn chuồn, 3 ảnh mèo, ta sẽ phân loại ảnh mới là chuồn chuồn.

Như vậy, **độ chính xác của KNN phụ thuộc rất nhiều vào giá trị k**.

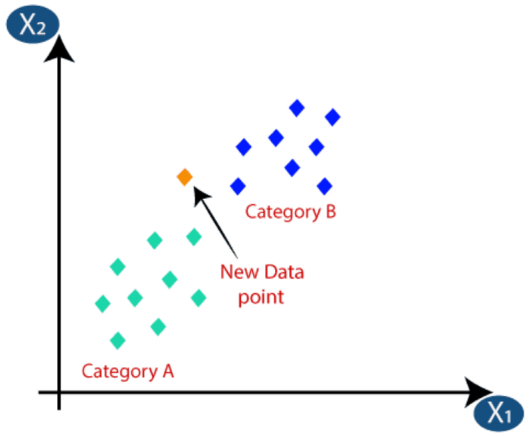
2.5 Kết luận

KNN là một phương pháp **Lazy Learning**, không cần huấn luyện trước mà chỉ dựa vào việc tìm kiếm trong tập dữ liệu. Phương pháp này rất hữu ích trong những bài toán mà việc huấn luyện quá tốn kém hoặc khi muốn đánh giá nhanh dữ liệu trước khi chọn mô hình Machine Learning phù hợp. Tuy nhiên, hiệu suất của KNN phụ thuộc nhiều vào cách đo khoảng cách và giá trị k được chọn

3 KNN for Classification

Giả sử chúng ta có một tập dữ liệu gồm **hai nhóm hình thoi**:

- **Nhóm hình thoi xanh lá**
- **Nhóm hình thoi xanh dương**



Khi xuất hiện **một hình thoi mới**, câu hỏi đặt ra là:

Hình thoi mới này thuộc nhóm nào?

Giả sử dữ liệu được biểu diễn trên hệ tọa độ **Oxy**, mỗi hình thoi trong dataset có **một tọa độ riêng** (có thể hình dung bằng cách trực quan hóa dữ liệu - Visualization). Khi đó, để phân loại hình thoi mới, ta sẽ **so sánh độ tương đồng (similarity)** giữa nó và các hình thoi trong tập dữ liệu bằng cách **tính khoảng cách Euclidean** (cách đơn giản nhất).

Ví dụ, nếu chọn $k = 5$, ta sẽ tìm **5 hình thoi có khoảng cách Euclidean nhỏ nhất** so với hình thoi mới. Dựa trên số lượng các nhóm hình thoi trong k hàng xóm gần nhất, ta quyết định hình thoi mới thuộc về nhóm nào.

Bài toán này có thể được mô hình hóa thành các bước sau:

Bước 1: Chọn số lượng hàng xóm k (Select K-Nearest Neighbors)

- Chọn một số nguyên dương k (ví dụ: $k = 5$).
- k quyết định số lượng điểm gần nhất mà ta sẽ xét để phân loại điểm dữ liệu mới.

$\Rightarrow$ **Mục tiêu**: Xác định giá trị k phù hợp.

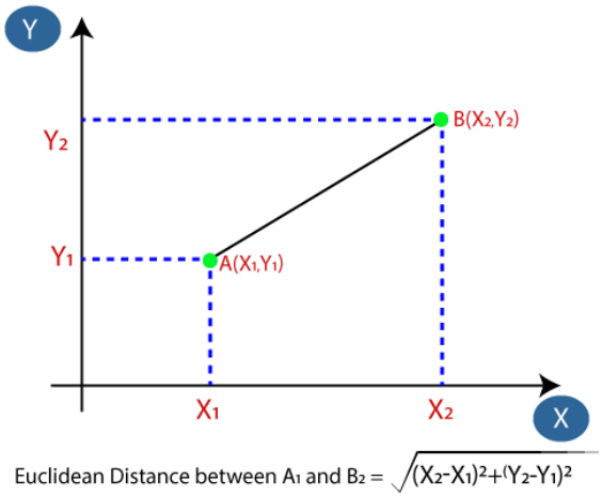
Bước 2: Định nghĩa phương pháp đo độ tương đồng (similarity measure)

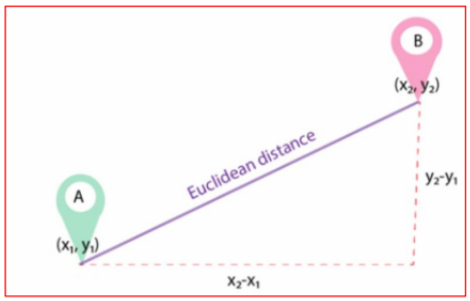
Để so sánh độ tương đồng giữa điểm mới và các điểm trong dataset, ta cần một **hàm đo khoảng cách**. Câu hỏi đặt ra là:

Làm thế nào để đo độ tương đồng giữa hai điểm A và B?

Một trong những cách phổ biến nhất là sử dụng **khoảng cách Euclidean**:

$$d(A, B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$$





Ý nghĩa:

- **Khoảng cách Euclidean nhỏ** ⇒ Hai điểm dữ liệu tương đồng cao.
- **Khoảng cách Euclidean lớn** ⇒ Hai điểm dữ liệu khác biệt nhiều.

Ngoài Euclidean distance, có các phương pháp khác để đo khoảng cách:

- **Manhattan Distance**: Tính khoảng cách theo từng trục tọa độ.
- **Chebyshev Distance**: Chỉ quan tâm đến khoảng cách lớn nhất trên một chiều tọa độ.
- **Hamming Distance**: Thường dùng cho dữ liệu rời rạc, như chuỗi nhị phân.
- **Cosine Similarity**: Đo góc giữa hai vector để đánh giá sự tương đồng.
- **Minkowski Distance** (tổng quát nhất):

$$d(A, B) = \left(\sum_{i=1}^{n}|A_i - B_i|^p\right)^{\frac{1}{p}}$$

  - Khi $p = 2$ ⇒ Euclidean distance.
  - Khi $p = 1$ ⇒ Manhattan distance.

⇒ **Mục tiêu**: Xác định phương pháp đo độ tương đồng phù hợp.

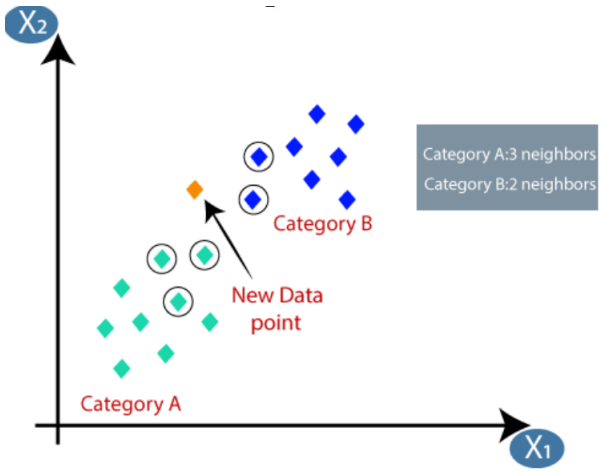
Bước 3: Tìm k hàng xóm gần nhất

Sau khi tính toán khoảng cách giữa điểm mới và toàn bộ tập dữ liệu, ta chọn **k điểm có khoảng cách nhỏ nhất** với điểm mới.

Ví dụ, nếu $k = 5$, ta chọn **5 hình thoi gần nhất** với hình thoi mới dựa trên khoảng cách Euclidean.
⇒ **Mục tiêu**: Xác định nhóm hàng xóm gần nhất.

Bước 4: Voting để dự đoán nhãn của điểm mới

- Kiểm tra nhãn của **k điểm gần nhất**.
- Nhóm nào xuất hiện nhiều hơn sẽ quyết định nhãn của điểm mới.



Ví dụ:

- Trong 5 hàng xóm gần nhất, có **3 hình thoi xanh lá** và **2 hình thoi xanh dương**.
- Hình thoi mới sẽ được phân vào nhóm **hình thoi xanh lá** vì nhóm này chiếm đa số.

⇒ **Mục tiêu**: Dự đoán nhãn của điểm mới dựa trên tần suất xuất hiện.

KNN là một phương pháp đơn giản nhưng hiệu quả trong phân loại. Với bài toán này, chúng ta có thể dễ dàng dự đoán nhóm của hình thoi mới bằng cách đo khoảng cách và sử dụng voting để xác định nhãn phù hợp.

Ví dụ: Bài toán phân loại hoa Iris

Giả sử chúng ta có một **dataset** chứa thông tin về loài hoa **Iris**, bao gồm:

- **4 thuộc tính đặc trưng**:
  1. **Sepal Length (Cm)**
  2. **Sepal Width (Cm)**
  3. **Petal Length (Cm)**
  4. **Petal Width (Cm)**
- **1 nhãn (label)**: **Species**, chỉ ra loài hoa Iris tương ứng.

Mục tiêu:

> Khi có một **bông hoa mới** với 4 thông tin trên, ta cần **dự đoán** xem nó thuộc **loài nào** trong ba loài **Iris Setosa, Iris Versicolor hoặc Iris Virginica**.

Phương pháp giải quyết bài toán

Thông thường, các thuật toán học máy như:

- **Support Vector Machine (SVM)**
- **Logistic Regression**
- **Decision Trees**

sẽ được sử dụng để **huấn luyện mô hình**. Quá trình này đòi hỏi việc **train dữ liệu**, tìm mối liên hệ giữa **Input (4 thuộc tính)** và **Output (loài hoa Iris tương ứng)**.

Tuy nhiên, với **K-Nearest Neighbors (KNN)**, ta **không cần bước huấn luyện mô hình**. KNN hoạt động hoàn toàn theo nguyên tắc **tìm hàng xóm gần nhất**.

Cách sử dụng KNN để phân loại hoa Iris

KNN hoạt động tương tự như bài toán phân loại hình thoi trước đó. Các bước thực hiện như sau:

1. **Bước 1: Chọn số lượng hàng xóm k**
   - Quyết định số lượng hàng xóm k cần xét, ví dụ $k = 5$.
2. **Bước 2: Tính khoảng cách giữa bông hoa mới và các mẫu trong dataset**
   - Sử dụng **Euclidean Distance** hoặc một phương pháp đo độ tương đồng khác.
3. **Bước 3: Chọn k điểm gần nhất**
   - Lấy **k điểm** có khoảng cách nhỏ nhất với bông hoa mới.
4. **Bước 4: Voting để xác định nhãn của hoa mới**
   - Xác định **loài hoa phổ biến nhất** trong k hàng xóm gần nhất → gán nhãn cho bông hoa mới.

Code: Iris Classification
Kết quả: Bông hoa mới sẽ được phân loại thành **Setosa, Versicolor hoặc Virginica** dựa trên dữ liệu gần nhất.

4 How to Select K in KNN

Tầm quan trọng của k trong k-NN

- k là một **hyperparameter** (siêu tham số), chỉ cần thay đổi giá trị của nó có thể ảnh hưởng rất lớn đến hiệu suất (**performance**) của mô hình.
- Có hai phương pháp phổ biến để chọn k:
  1. **Thử nghiệm trên nhiều giá trị của** k, sau đó chọn giá trị tốt nhất.
  2. **Sử dụng Cross-Validation** để tìm k tối ưu. Ngoài ra, chúng ta cũng có thể dùng **Grid Search Cross Validation** để tự động hóa quá trình tìm kiếm này.

Đánh giá độ chính xác của mô hình trước khi chọn k

Trước khi xác định giá trị k, ta cần có phương pháp để đánh giá xem thuật toán có hoạt động chính xác hay không. Giả sử đã có nhiều giá trị k, làm sao biết giá trị nào là tốt nhất?
Cần có thước đo độ chính xác!

Cách đo lường độ chính xác

- Ta có thể sử dụng các hàm có sẵn trong thư viện `sklearn`:
  - **Accuracy Score**
  - **Classification Report**
- **Ví dụ sử dụng Accuracy Score:**

```
    from sklearn.metrics import accuracy_score
    print(round(accuracy_score(y_test, y_pred), 2))

```

- **Ví dụ sử dụng Classification Report:**

```
    from sklearn.metrics import classification_report
    print(classification_report(y_test, y_pred))

```

Hạn chế của Accuracy - Ví dụ bài toán chẩn đoán ung thư

Xét một bài toán **Cancer Classification**, dự đoán bệnh nhân có bị ung thư hay không.
Trường hợp 1: Đánh giá dựa trên Accuracy Giả sử ta huấn luyện mô hình **Logistic Regression** và dự đoán trên một tập kiểm tra gồm 1000 bệnh nhân:

- **Dự đoán chính xác 99%**, tức là chỉ có 10 bệnh nhân bị chẩn đoán sai.
- Ta có thể kết luận độ chính xác là **99%**? Có đáng tin cậy không?

Trường hợp 2: Dataset mất cân bằng Giả sử trong 1000 bệnh nhân, chỉ có **5 người thực sự bị ung thư**. Bây giờ xét hai thuật toán:

Giải thuật 1 (Sai nhưng có Accuracy cao)

```
def predictCancer(x):
    return 0
```

Giải thuật 2 (Dùng Logistic Regression)

- Giả sử mô hình học máy huấn luyện trên dữ liệu thật, nhưng vẫn có **accuracy = 99%**.
- Nếu chỉ đánh giá bằng accuracy, ta có thể kết luận **giải thuật 1 tốt hơn giải thuật 2**, trong khi thực tế không phải vậy.

Precision, Recall và F1-score - Giải pháp thay thế Accuracy

- **Precision**
  Trong số tất cả bệnh nhân mà mô hình dự đoán bị ung thư, có bao nhiêu bệnh nhân thực sự bị ung thư?

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Recall**
  Trong số tất cả bệnh nhân thực sự bị ung thư, mô hình dự đoán đúng bao nhiêu trường hợp?

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **Ví dụ so sánh Precision và Recall**

k	Algorithm	Precision (P)	Recall (R)
1	Algorithm 1	0.5	0.4
2	Algorithm 2	0.7	0.1
3	Algorithm 3	0.02	1.0

Bảng 1: So sánh Precision và Recall giữa các thuật toán

F1-score - Giải pháp cân bằng Precision và Recall

F1-score là trung bình điều hòa giữa Precision và Recall, giúp đánh giá mô hình một cách công bằng hơn:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Tại sao dùng F1-score?

- Nếu chỉ dựa vào Accuracy, ta có thể mắc sai lầm nghiêm trọng.
- F1-score giúp cân bằng giữa Precision và Recall, đặc biệt quan trọng trong các bài toán mất cân bằng dữ liệu (**imbalanced dataset**).
- Khi sử dụng F1-score, ta có thể lựa chọn **Algorithm 1 ($k = 1$)** vì nó đạt cân bằng tốt nhất giữa Precision và Recall.

Do đó, khi đánh giá mô hình k-NN, thay vì chỉ nhìn vào Accuracy, ta nên sử dụng Precision, Recall và F1-score để có cái nhìn chính xác hơn về hiệu suất của mô hình!

4.1 Lựa chọn k trong k-NN

Một trong những phương pháp phổ biến để chọn k trong k-NN là thử nghiệm trên nhiều giá trị khác nhau của k, sau đó chọn giá trị tối ưu. Chẳng hạn, nếu ta quan tâm đến độ chính xác (accuracy), ta có thể sử dụng nó làm thước đo để đánh giá và lựa chọn k. Việc trực quan hóa qua biểu đồ giúp ta xác định k phù hợp hơn. Ngoài ra, kỹ thuật **Cross Validation** (như k-fold validation hoặc grid cross validation) cũng là một cách hữu ích để tìm k tối ưu. Đặc biệt, grid cross validation thường phù hợp với dữ liệu time series.

Ví dụ về lựa chọn k bằng thử nghiệm:

```
error = []
# Tinh accuracy cho cac gia tri k tu 1 den 30
for i in range(1, 30):
      knn = KNeighborsClassifier(n_neighbors=i)
      knn.fit(X_train, y_train)
      pred_i = knn.predict(X_test)
```

```
        error.append(accuracy_score(y_test, pred_i))

plt.figure(figsize=(12,5))
plt.plot(range(1, 30), error, color='blue', marker='o', markerfacecolor='yellow',
    markersize=10)
plt.title('Accuracy vs K Value')
plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.show()
```

Dựa vào biểu đồ trên, ta có thể chọn k tối ưu bằng cách tìm giá trị k cho độ chính xác cao nhất. Nếu dùng **F1-score** thay vì accuracy, ta sẽ chọn k có **F1-score** cao nhất.

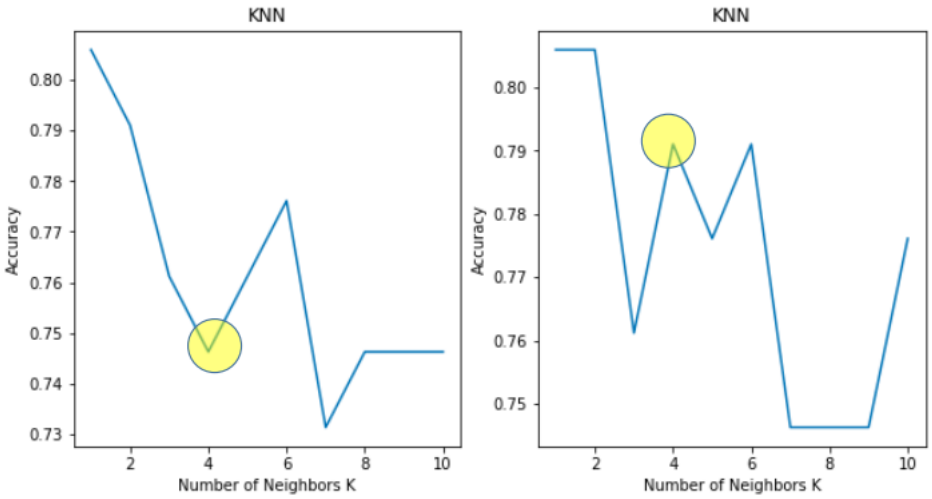
Câu hỏi: KNN có "train"không?
Trong đoạn code trên, có hai dòng quan trọng:

```
knn.fit(X_train, y_train)
pred_i = knn.predict(X_test)
```

Dù KNN là một thuật toán không cần huấn luyện mô hình (non-parametric), nhưng tại sao vẫn có hàm `fit()`? **Thực chất, `fit()` không huấn luyện mô hình như các thuật toán khác mà chỉ lưu trữ dữ liệu huấn luyện để sử dụng trong quá trình dự đoán.** KNN chỉ thực sự tính toán khi có dữ liệu mới cần dự đoán (`predict()`).

4.2 Ảnh hưởng của k trong KNN (Brute-force)

Trong phương pháp **Brute-force KNN**, khi thử nghiệm với các giá trị k khác nhau, ta có thể nhận thấy **sự dao động trong độ chính xác giữa các lần chạy chương trình**. Điều này đặc biệt rõ ràng với **các giá trị k chẵn**.



Hình 1: Biểu đồ accuracy khi chạy KNN 2 lần khác nhau

- **Ví dụ về sự khác biệt khi k là số chẵn:**
  - Giả sử với **k = 4**, lần chạy thứ nhất có **accuracy = 0.745**, nhưng lần chạy thứ hai lại có **accuracy = 0.79**.
  - Trong khi đó, với **k lẻ**, độ chính xác ổn định hơn.

- **Tại sao k chẵn ảnh hưởng đến độ chính xác?**
  Để hiểu điều này, ta xem lại các bước của KNN:

  1. Chọn số lượng hàng xóm k.
  2. Định nghĩa phương pháp đo độ tương đồng (thường là Euclidean distance).
  3. Tìm **k hàng xóm gần nhất**.
  4. **Voting để dự đoán nhãn của điểm mới**.

Ở **bước 4 (Voting)**, nếu k là số chẵn, có thể xảy ra trường hợp **số lượng nhãn bị chia đều** giữa hai lớp. Ví dụ: - Với $\mathbf{k = 4}$, nếu 2 điểm thuộc loại A và 2 điểm thuộc loại B, thuật toán sẽ không biết chọn loại nào, dẫn đến sự bất ổn định trong dự đoán. - Tương tự, trong bài toán phân loại nhiều lớp như **Iris dataset**, giá trị k không phù hợp có thể làm giảm hiệu suất mô hình.

4.3 Cách giải quyết vấn đề khi k là số chẵn?

Nếu chỉ sử dụng voting, việc chọn loại A hay B đều hợp lý, nhưng điều này không tối ưu. Thay vào đó, ta có thể **đánh trọng số (weighting)** để ưu tiên các điểm gần hơn trong không gian đặc trưng.
Ba phương pháp đánh trọng số trong KNN:

1. **Uniform weights** (Mặc định trong KNN)
   - Mọi điểm trong k hàng xóm gần nhất đều có trọng số bằng nhau.
   - Không quan tâm đến khoảng cách.
   - Không thể giải quyết vấn đề khi k là số chẵn.
2. **Distance weights** (Trọng số theo khoảng cách)
   - Trọng số được tính theo 1/khoảng cách, nghĩa là **các điểm gần hơn sẽ có trọng số lớn hơn**.
   - Phân loại điểm mới bằng cách cộng trọng số của các điểm thuộc từng lớp.
3. **User-defined weights** (Trọng số do người dùng xác định)
   - Có thể tự đặt trọng số theo quy tắc riêng dựa trên kiến thức chuyên môn hoặc đặc điểm dữ liệu.
   - Dùng khi **cả hai cách trên không phù hợp với bài toán**.

4.4 Nếu k quá lớn hoặc k quá bé thì xảy ra hiện tượng gì?

Việc chọn k không phù hợp có thể gây ra hai vấn đề chính:

- **Nếu k quá bé (ví dụ $k = 1$ hoặc $k = 3$)**
  - **Dễ bị overfitting**: Mô hình có thể nhạy cảm với nhiễu trong dữ liệu. Nếu một điểm dữ liệu bị gán nhãn sai, nó có thể ảnh hưởng lớn đến kết quả dự đoán.
  - **Quyết định bị chi phối bởi điểm lân cận gần nhất**, không phản ánh tổng thể xu hướng dữ liệu.
  - **Dự đoán có thể không ổn định**: Nếu có một chút thay đổi trong dữ liệu, nhãn của điểm cần dự đoán có thể thay đổi mạnh.

Ví dụ: Giả sử ta có dữ liệu phân loại mèo và chó. Nếu $k = 1$, chỉ một điểm gần nhất quyết định nhãn. Nếu điểm gần nhất đó là nhiễu (ví dụ một con mèo bị gán nhầm là chó), thì mô hình có thể dự đoán sai.

- **Nếu k quá lớn (ví dụ $k = N$, với N là toàn bộ dữ liệu)**
  - **Dễ bị underfitting**: Nếu chọn k quá lớn, mô hình sẽ chỉ đơn giản chọn nhóm có số lượng lớn hơn, mất đi khả năng nhận diện mẫu cục bộ.
  - **Giảm độ chính xác**: Các điểm xa hơn, không liên quan cũng được tính vào quyết định phân loại, khiến mô hình kém hiệu quả.
  - **Thời gian tính toán tăng lên**: Khi k lớn, cần tính khoảng cách và sắp xếp nhiều điểm hơn, làm giảm tốc độ dự đoán.

  Ví dụ: Nếu dùng $k = 1000$ trên dữ liệu có 1100 mèo và 900 chó, thì tất cả dự đoán sẽ nghiêng về mèo, bất kể điểm gần nhất có là chó hay không.

4.5 Kết luận

- **Việc chọn k trong KNN rất quan trọng**, có thể thực hiện bằng thử nghiệm, Cross Validation, hoặc các phương pháp tối ưu khác.
- **KNN không thực sự "huấn luyện"mô hình**, mà chỉ lưu trữ dữ liệu và sử dụng trong quá trình dự đoán.
- **k chẵn có thể gây ra vấn đề khi voting**, có thể khắc phục bằng cách **đánh trọng số** theo khoảng cách thay vì sử dụng voting thông thường.
- **Chọn phương pháp đánh trọng số phù hợp** sẽ giúp KNN hoạt động hiệu quả hơn, đặc biệt là trong các bài toán phân loại phức tạp.
- **k quá bé có thể gây overfitting cục bộ, trong khi k quá lớn có thể gây underfitting toàn cục**.

5 KNN for Regression

Nếu bài toán của chúng ta là **KNN cho Regression** thì sao? Trước hết, cần hiểu rằng **bài toán Regression khác với Classification** ở chỗ **đầu ra (output) của mô hình là một giá trị số liên tục**, thay vì một lớp (category) cụ thể.

Ví dụ, trong bài toán **phân loại hoa Iris**, đầu ra là một trong ba loại hoa: **Iris Setosa, Iris Versicolor, Iris Virginica** (các giá trị rời rạc). Nhưng nếu bài toán này chuyển sang dạng **hồi quy**, thì **đầu ra có thể là giá của một bông hoa**, một giá trị số **liên tục**, không giới hạn trong một tập hợp hữu hạn.

Cách áp dụng KNN vào bài toán Regression

KNN cho bài toán hồi quy vẫn tuân theo **quy trình cơ bản giống với Classification**, với một điểm khác biệt quan trọng:

- Trong **Classification**, ta tìm **k điểm gần nhất** và chọn **lớp có tần suất cao nhất** để gán nhãn.
- Trong **Regression**, ta cũng tìm **k điểm gần nhất**, nhưng **thay vì voting**, ta **tính toán giá trị trung bình (hoặc trung vị,...) của các giá trị đầu ra** của k điểm này và sử dụng nó làm dự đoán.

Tóm tắt quy trình KNN Regression qua 4 bước

1. **Chọn số lượng hàng xóm k**
2. **Định nghĩa phương pháp đo độ tương đồng** (ví dụ: khoảng cách Euclid)
3. **Tìm k hàng xóm gần nhất**
4. **Tính toán giá trị trung bình (hoặc trung vị,...) của k điểm gần nhất và dự đoán giá trị đầu ra**

So sánh KNN Regression với các phương pháp khác

- KNN **không yêu cầu bất kỳ giả định nào về dữ liệu**. Điều này khác với Linear Regression, SVM, hay các mô hình khác, vì chúng thường **giả định một số tính chất nhất định của dữ liệu** (ví dụ: Linear Regression giả định dữ liệu có mối quan hệ tuyến tính).
- KNN **không quan tâm đến việc dữ liệu có tuyến tính (linear) hay phi tuyến tính (non-linear)**, mà chỉ đơn giản là tìm những điểm gần nhất để đưa ra dự đoán.

Ứng dụng và đánh giá KNN Regression

- KNN thường được sử dụng khi **chưa có thông tin rõ ràng về bản chất của dữ liệu**, vì nó không yêu cầu giả định về phân phối dữ liệu.
- Độ chính xác của KNN có thể **không cao**, nhưng nó là một phương pháp **đơn giản và hiệu quả** để **đánh giá sơ bộ tập dữ liệu**.
- Nếu KNN cho kết quả quá thấp, ta có thể **phân tích nguyên nhân** (ví dụ: dữ liệu có nhiễu, phân bố không đồng đều, giá trị ngoại lai,...) để hiểu rõ hơn về dataset. Sau đó, ta có thể chọn một phương pháp khác phù hợp hơn.

Tóm lại: KNN Regression là một cách tiếp cận đơn giản, không đòi hỏi giả định về dữ liệu, và giúp đánh giá tổng quan về dataset. Tuy nhiên, nếu KNN không hoạt động tốt, ta cần kiểm tra lại dữ liệu và cân nhắc phương pháp khác.

6 Tăng tốc KNN bằng KD-Tree

Giải thích về hàm `fit()` trong KNN

Khi sử dụng KNN, ta có đoạn code:

```
knn.fit(X_train, y_train)
pred_i = knn.predict(X_test)
```

Ta biết rằng KNN không thực sự có giai đoạn "train"giống như các mô hình học máy khác (như Linear Regression hay Neural Networks). Vậy tại sao vẫn có hàm `fit()`, và nó có thực sự "train"mô hình không?

Thực tế, hàm `fit()` trong KNN **không** thực hiện quá trình học (training) mà chỉ lưu trữ dữ liệu huấn luyện và **xây dựng một cấu trúc dữ liệu hỗ trợ** để tăng tốc quá trình tìm kiếm hàng xóm gần nhất.

Nhược điểm chính của KNN: Tốc độ xử lý

Nhược điểm lớn nhất của KNN là **tốc độ xử lý chậm** khi dataset có kích thước lớn.

- Giả sử ta có **1 triệu sample**, khi một **sample mới** xuất hiện, thuật toán phải tính khoảng cách từ sample đó đến **toàn bộ 1 triệu điểm**, sau đó **sắp xếp và chọn k điểm gần nhất**.
- Nếu thực hiện theo cách brute-force (vét cạn), mỗi lần dự đoán sẽ **tốn rất nhiều thời gian** do phải thực hiện phép tính khoảng cách cho tất cả điểm dữ liệu.

Vậy hàm `fit()` có vai trò gì?

- Khi gọi `fit()`, KNN có thể **xây dựng một cấu trúc dữ liệu hỗ trợ (như KD-Tree hoặc Ball-Tree)** giúp **tăng tốc độ tìm kiếm hàng xóm gần nhất**.
- Nhờ đó, thay vì phải tính khoảng cách đến **toàn bộ dataset**, thuật toán có thể **loại bớt rất nhiều điểm không cần thiết**, **giảm đáng kể số phép tính toán** khi dự đoán.

Vấn đề của KNN khi số lượng features lớn

Ngoài vấn đề về tốc độ, KNN còn gặp khó khăn khi dữ liệu có số lượng features lớn:

- Nếu số lượng features rất lớn (**high-dimensional data**), chỉ một sai lệch nhỏ trong cách tính khoảng cách cũng có thể **gây ảnh hưởng lớn** đến độ chính xác của mô hình.
- Ví dụ: Khi xử lý dữ liệu ảnh có hàng ngàn pixel làm feature, KNN có thể gặp vấn đề do khoảng cách Euclidean không còn chính xác.
- Giải pháp: **Giảm số lượng features (dimensionality reduction)** bằng các phương pháp như **PCA (Principal Component Analysis)** hoặc **feature selection** để chỉ giữ lại những feature quan trọng nhất.

6.1 KNN với Brute-Force

- Thông thường, KNN sử dụng **Brute-Force Search**: Khi có một sample mới, thuật toán phải **tính khoảng cách đến tất cả các điểm** trong dataset, sau đó **sắp xếp và chọn k điểm gần nhất**.
- Với dataset nhỏ, cách này **không phải vấn đề lớn**. Nhưng với **dataset lớn**, cách này sẽ **rất chậm**.
- **Giải pháp:** Thay vì kiểm tra tất cả các điểm, ta có thể sử dụng **KD-Tree** hoặc **Ball-Tree** để **tăng tốc tìm kiếm**.

6.2 KD-Tree: Cấu trúc dữ liệu giúp tăng tốc KNN

Giả sử ta có dataset với 2 features (X, Y), cách xây dựng KD-Tree như sau:

1. **Chọn một feature làm trục phân tách** (ví dụ: chọn feature X).
2. **Tìm trung vị (median) của feature đó** để chia dataset thành hai nửa gần bằng nhau.
3. **Chia dataset thành 2 nhóm:**
    - **Nhóm bên trái** có giá trị nhỏ hơn trung vị.

- **Nhóm bên phải** có giá trị lớn hơn trung vị.

4. **Chuyển sang feature tiếp theo (ví dụ: Y) và lặp lại bước 2, 3.**
5. **Tiếp tục quá trình này cho đến khi tất cả các điểm dữ liệu được phân vùng.**

Ví dụ: Với 7 điểm dữ liệu trong không gian 2D:

$$A(2,3), B(5,4), C(9,6), D(4,7), E(8,1), F(7,2), G(1,5)$$

- Bước 1 – Cấp 0 (chia theo x):

  Danh sách theo x tăng dần:

```
    P7 (1,5), P1 (2,3), P4 (4,7), **P2 (5,4)**, P6 (7,2), P5 (8,1), P3 (9,6)

```

  ⇒ Median = **P2 (5,4)** → đây là **root**

- Bước 2 – Cấp 1:
  - Trái của P2: [P7, P1, P4]
    - Chia theo **y**
    - Sắp theo y:

```
        P1 (2,3), P7 (1,5), P4 (4,7)

```

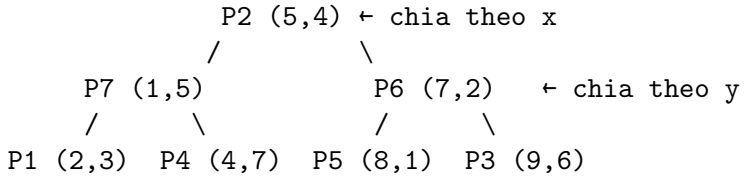
    - Median = **P7 (1,5)** → node con trái của P2
  - Phải của P2: [P6, P5, P3]
    - Chia theo **y**
    - Sắp theo y:

```
            P5 (8,1), P6 (7,2), P3 (9,6)

```

    - Median = **P6 (7,2)** → node con phải của P2

- Bước 3 – Cấp 2:
  - Trái của P7: [P1]
    - Chia theo **x**
    - 1 điểm duy nhất → làm node lá
  - Phải của P7: [P4]
    - Cũng chỉ có 1 điểm → node lá
  - Trái của P6: [P5]
    - Node lá
  - Phải của P6: [P3]
    - Node lá

- Kết quả KD-Tree:



6.3 Tìm kiếm KNN bằng KD-Tree

Sau khi KD-Tree được xây dựng, thuật toán tìm KNN hoạt động như sau:

1. **Bắt đầu từ root**
2. **Đi xuống (đệ quy)** đến **lá** theo nhánh phù hợp nhất với vị trí của A
3. **Cập nhật "best-so-far"list** (heap gồm k điểm gần nhất hiện tại)
4. **Quay lui (backtrack)** và kiểm tra **nhánh bên kia nếu cần**
5. Kết thúc khi duyệt xong các nhánh có thể chứa điểm gần hơn

Giải thích chi tiết từng bước:

- 1. **Xuống cây theo vị trí của A**
  - Tại mỗi node, kiểm tra trục chia (x hoặc y).
  - Nếu A nhỏ hơn node → đi về bên trái; lớn hơn → đi bên phải.
  - **Tiếp tục đi xuống cho đến node lá.**
- 2. **Gặp node lá → thêm điểm vào danh sách "k gần nhất"**
  - Bắt đầu cập nhật **heap/priority queue** lưu các điểm gần nhất (max-heap nếu dùng Python).
  - Nếu chưa đủ k điểm → thêm thoải mái.
  - Nếu đã đủ → chỉ thay thế nếu điểm mới gần hơn.
- 3. **Backtrack (quay lui)** và kiểm tra nhánh còn lại
  Tại mỗi node khi quay ngược trở lên:
  So sánh:
  - So khoảng cách từ A đến **mặt phẳng chia** tại node hiện tại.
  - Nếu khoảng cách này $<$ **khoảng cách xa nhất trong heap** → **có thể có điểm gần hơn bên nhánh kia** → phải kiểm tra nhánh kia.

  Ngược lại:
  - Nếu mặt phẳng chia **quá xa** → bỏ qua luôn nhánh đó → tiết kiệm thời gian.
- 4. **Tiếp tục quá trình duyệt như trên**, cập nhật heap mỗi khi gặp điểm gần hơn.
- 5. Kết thúc
  - Khi đã duyệt xong tất cả các nhánh có khả năng chứa điểm gần hơn → heap chứa đúng **k điểm gần nhất**.

Ví dụ: Tìm điểm gần nhất với của $A(6, 5)$:

- Bước 1: Tại **root = P2 (5,4)**, chia theo **x**
  - A.x = 6 $>$ P2.x = 5 → đi nhánh phải → **P6 (7,2)**
  - Giữ `best = P2`, khoảng cách = $\sqrt{(6-5)^2 + (5-4)^2} = \sqrt{2} \approx$ **1.41**

- Bước 2: Tại **P6 (7,2)**, chia theo **y**
  - A.y = 5 > P6.y = 2 → đi nhánh phải → **P3 (9,6)**
  - So khoảng cách:
    - A → P6 = $\sqrt{(6-7)^2+(5-2)^2} = \sqrt{10} \approx 3.16$
    - `best` vẫn là P2 (1.41 < 3.16)
- Bước 3: Tại **P3 (9,6)** (lá)
  - A → P3 = $\sqrt{(6-9)^2+(5-6)^2} = \sqrt{10} \approx 3.16$
  - Không tốt hơn `best`, bỏ qua
- Backtrack lên **P6**
  - Khoảng cách từ A đến mặt phẳng chia y=2 = $|5-2| = 3 \rightarrow 3 > 1.41$ → **có thể có điểm gần hơn ở nhánh còn lại**
  - ⇒ **Duyệt nhánh trái của P6 → P5 (8,1)**
    - A → P5 = $\sqrt{(6-8)^2+(5-1)^2} = \sqrt{20} \approx 4.47$
    - Không gần hơn → bỏ
- Backtrack lên **P2 (root)**
  - Khoảng cách từ A đến mặt phẳng chia x=5 = $|6-5| = 1$
  - $1 < 1.41$ → **phải duyệt nhánh trái**!
  - ⇒ Sang trái → **P7 (1,5)**
- Tại **P7 (1,5)**, chia theo **y**
  - A.y = 5 = P7.y → đi phải → **P4 (4,7)**
  - A → P4 = $\sqrt{(6-4)^2+(5-7)^2} = \sqrt{8} \approx 2.83$ → vẫn thua P2
  - ⇒ Duyệt nốt **P1 (2,3)** → khoảng cách $\sqrt{20} \approx 4.47$ → thua

Điểm gần nhất với A(6,5) là P2(5,4), với khoảng cách $\sqrt{2} \approx 1.41$

6.4 Ưu điểm của KD-Tree trong KNN

- **Giảm số phép tính khoảng cách**
  - **Brute-force KNN:** Để tìm k điểm gần nhất của điểm Q, ta phải tính khoảng cách từ Q đến **tất cả** điểm trong tập dữ liệu (tốn $O(n)$ phép toán).
  - **KD-Tree KNN:** Thay vì xét toàn bộ tập dữ liệu, KD-Tree giúp **loại bỏ các vùng không liên quan**, giảm số phép tính khoảng cách xuống còn khoảng $O(\log n)$ trong trường hợp lý tưởng.

Ví dụ minh họa:

- Giả sử ta có 10.000 điểm dữ liệu trong không gian 2D.
- Với Brute-force, ta phải tính 10.000 khoảng cách.
- Với KD-Tree, chỉ cần tính khoảng cách đến một phần nhỏ trong số đó, ví dụ **300-500 điểm** (tùy cấu trúc cây).

Kết quả: Thời gian chạy nhanh hơn đáng kể khi dữ liệu lớn.

- **Phù hợp khi số chiều nhỏ (low-dimensional space)**
  - KD-Tree hoạt động **hiệu quả nhất khi số chiều d nhỏ** (thường $d \leq 10$).
  - Trong không gian thấp, việc chia vùng (hyperplane) giúp loại bỏ nhiều điểm không cần thiết.

  Ví dụ:
  - Nếu ta có dữ liệu 2D (x, y), mỗi lần chia không gian sẽ loại bỏ khoảng **50% dữ liệu** khỏi việc xét KNN.
  - Khi d tăng, hiệu suất giảm do các vùng phân tách trở nên kém hiệu quả hơn.
- **Tối ưu cho dữ liệu phân bố đều**
  - Nếu dữ liệu **phân bố đồng đều**, các vùng được chia sẽ cân bằng và thuật toán có thể truy vấn nhanh.
  - **Ngược lại**, nếu dữ liệu có sự tập trung cao tại một vùng nhỏ, cây có thể bị mất cân bằng, làm giảm hiệu suất.
- **Có thể sử dụng trong bài toán tìm kiếm gần đúng (approximate nearest neighbor)**
  - Nếu không cần kết quả **chính xác tuyệt đối**, ta có thể dừng sớm quá trình truy vấn và chấp nhận một số lỗi nhỏ để tăng tốc độ.

6.5 Hạn chế của KD-Tree trong KNN

- **Không hiệu quả khi số chiều cao (curse of dimensionality)**
  - Khi số chiều d tăng, hiệu quả của KD-Tree giảm mạnh.
  - Trong không gian cao, hầu như mọi điểm đều **gần như cách đều nhau**, khiến việc chia vùng không còn giúp ích nhiều.

  Giải pháp: Sử dụng **Ball Tree** hoặc **Annoy (Approximate Nearest Neighbor)** cho dữ liệu có nhiều chiều.
- **Cây có thể mất cân bằng, làm giảm hiệu suất**
  - Nếu dữ liệu không được phân bố đều, KD-Tree có thể trở nên **mất cân bằng** và có chiều cao lớn hơn mức lý tưởng.
  - Khi đó, việc tìm kiếm sẽ không còn nhanh hơn so với brute-force nhiều.

  Giải pháp:
  - **Chọn median làm nút gốc** thay vì chọn điểm bất kỳ để đảm bảo cây cân bằng.
  - **Dùng random projection** để cải thiện phân bố dữ liệu.
- **Một số trường hợp vẫn phải xét toàn bộ dataset**
  - Nếu KD-Tree chia vùng không hiệu quả, hoặc nếu k quá lớn, có thể xảy ra trường hợp **mọi điểm đều cần được xét**, giống như Brute-force.

  Giải pháp:

  - **Giảm số k cần tìm** nếu có thể.
  - **Sử dụng heuristic để giới hạn việc xét nhánh đối diện**.

- **Không phù hợp khi dữ liệu thay đổi liên tục**
  - Nếu dataset thường xuyên thay đổi (thêm/xóa điểm), KD-Tree phải được **xây dựng lại từ đầu** để đảm bảo hiệu suất.

  Giải pháp: Nếu dữ liệu động, thay vì dùng KD-Tree, có thể xem xét **LSH (Locality-Sensitive Hashing)** hoặc **Annoy (Approximate Nearest Neighbor)**.

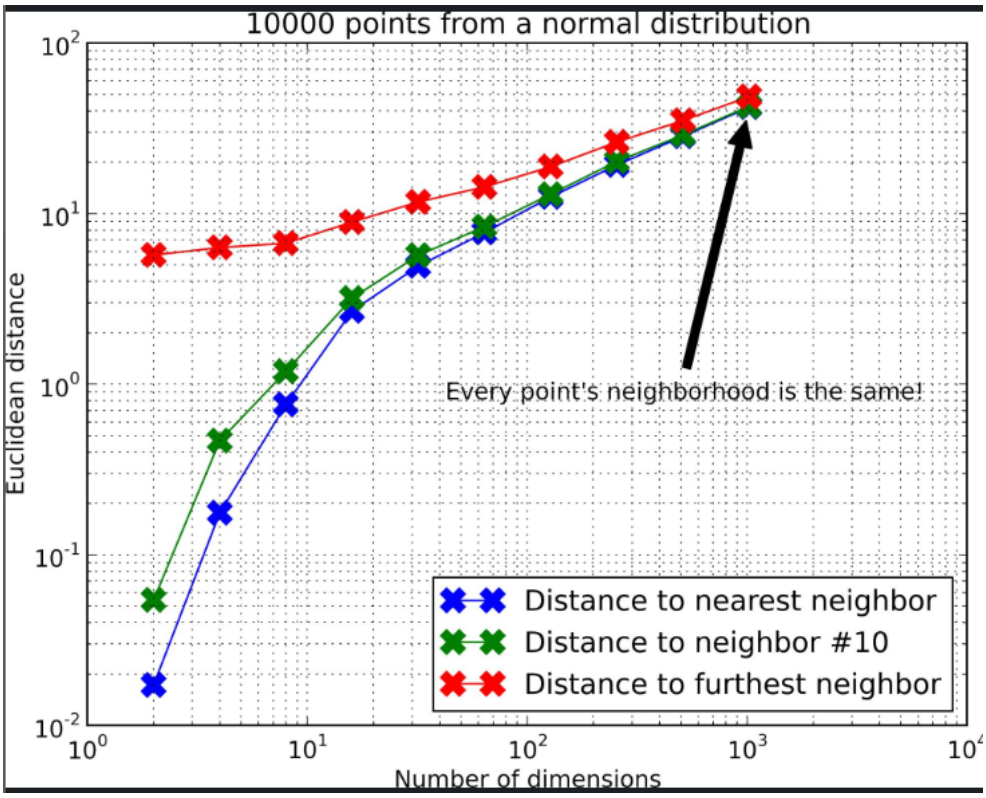
6.6 Kết luận

- KD-Tree là một **công cụ mạnh** giúp tăng tốc tìm kiếm KNN, đặc biệt hiệu quả khi **số chiều nhỏ**.
- Tuy nhiên, **với dữ liệu có nhiều chiều (high-dimensional data)**, hiệu quả của KD-Tree giảm đáng kể, và lúc này ta cần xem xét các phương pháp khác như **Ball Tree** hoặc **Annoy**.
- Việc lựa chọn phương pháp phù hợp phụ thuộc vào **kích thước và đặc điểm của dữ liệu đầu vào**.

7 Feature Engineering and Feature Selection trong KNN

7.1 Dữ liệu nhiều chiều

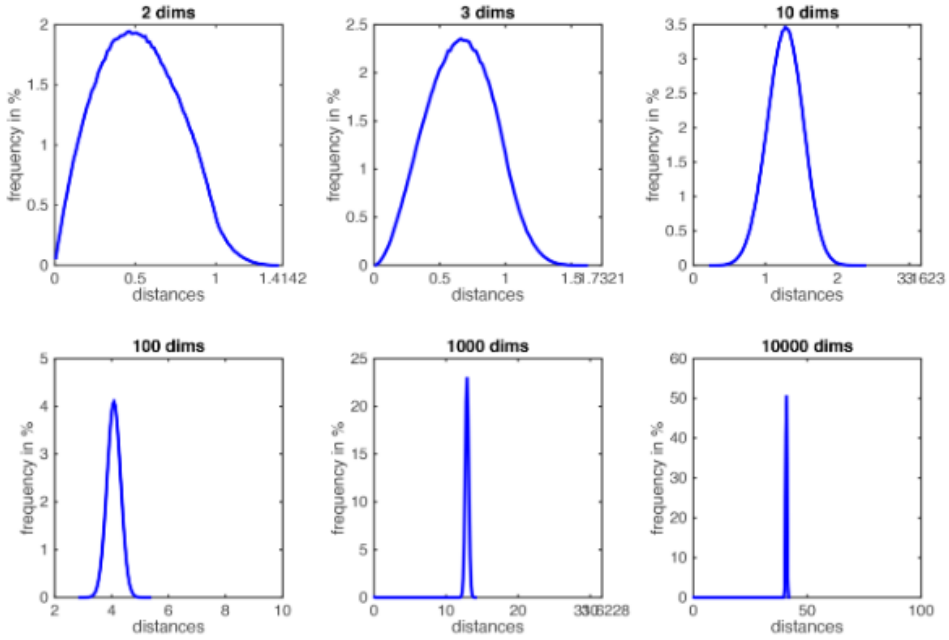
7.1.1 Ảnh hưởng dữ liệu nhiều chiều trong mô hình KNN



Hình 2: Khoảng cách khi dữ liệu nhiều chiều

Lời nguyền của chiều dữ liệu xảy ra khi số chiều (dimensions) quá lớn. Khi số chiều tăng lên, khoảng cách giữa bất kỳ hai điểm nào cũng có xu hướng trở nên gần như bằng nhau (vì khi cộng dồn khoảng cách ở tất cả các chiều, giá trị thường tăng đến một mức rất lớn và tương tự nhau). Điều này khiến khái niệm "gần nhau" gần như mất ý nghĩa, và mọi khoảng cách đều xấp xỉ bằng nhau. Vì KNN là một phương pháp dựa trên khoảng cách, nên nó chịu ảnh hưởng trực tiếp từ dữ liệu có chiều cao. Khi khoảng cách riêng lẻ giữa các điểm trở nên kém quan trọng hơn trong không gian có chiều cao, thì không gian chiều cao đó trở nên đồng nhất hơn. Một số ảnh hưởng cụ thể như sau:

- **Chi phí tính toán tăng:** Số lượng phép tính khoảng cách tăng lên theo số chiều của dữ liệu.
- **Khoảng cách đồng nhất:** Trong không gian có chiều cao, khoảng cách giữa các điểm có xu hướng trở nên tương tự nhau. Sự đồng nhất này làm cho việc phân biệt điểm gần và điểm xa trở nên khó khăn.
- **Quá khớp (Overfitting):** Không gian chiều cao có thể dẫn đến hiện tượng quá khớp, tức là mô hình học cả nhiễu trong dữ liệu huấn luyện thay vì các mẫu thực sự.



Hình 3: Lời nguyền của chiều dữ liệu

Thuật toán phân loại kNN giả định rằng các điểm tương tự nhau sẽ có nhãn giống nhau. Tuy nhiên, trong không gian nhiều chiều, các điểm được lấy ngẫu nhiên từ một phân phối xác suất thường **không bao giờ thật sự gần nhau**. Chúng ta có thể minh họa điều này bằng một ví dụ đơn giản: vẽ ngẫu nhiên các điểm trong hình lập phương đơn vị (unit cube) và xem xét lượng không gian mà **k hàng xóm gần nhất** của một điểm kiểm tra sẽ chiếm.

Cụ thể, xét hình lập phương đơn vị $[0,1]^d$. Tất cả dữ liệu huấn luyện được lấy phân phối đều trong hình lập phương này, tức là $\forall i, x_i \in [0,1]^d$, và ta xét $k = 10$ hàng xóm gần nhất của điểm kiểm tra. Gọi ℓ là độ dài cạnh của **siêu lập phương nhỏ nhất** chứa tất cả k hàng xóm gần nhất. Khi đó:

$$\ell^d \approx \frac{k}{n} \quad \Rightarrow \quad \ell \approx \left(\frac{k}{n}\right)^{1/d}$$

Nếu $n = 1000$, ta có:

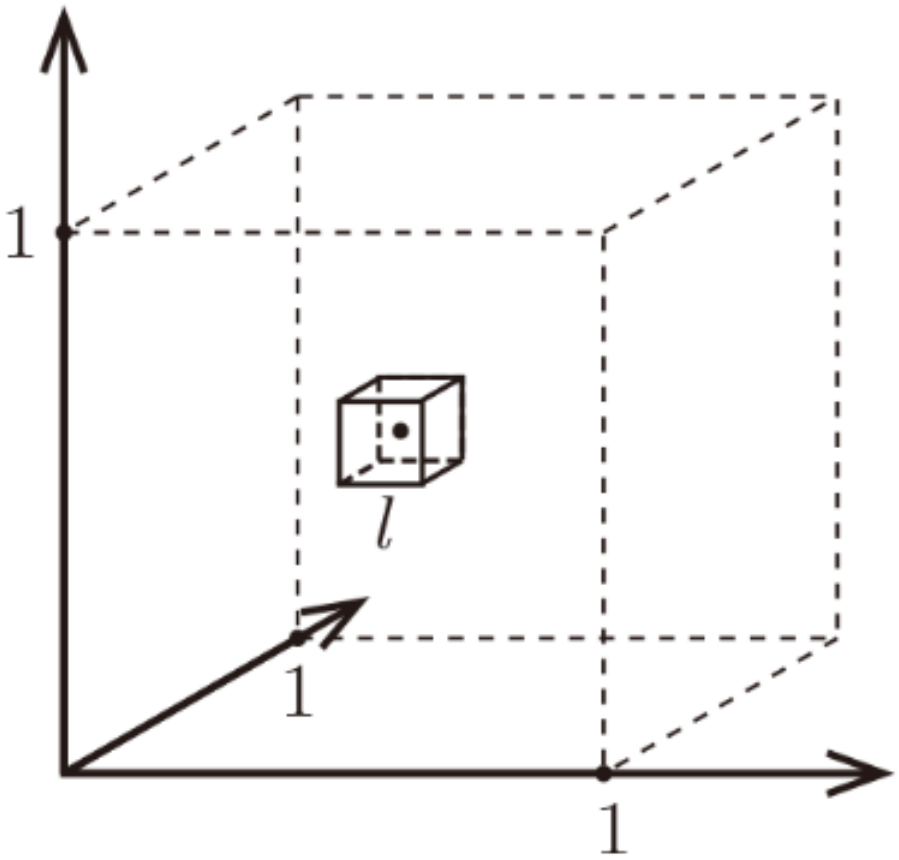
d	ℓ
2	0.1
10	0.63
100	0.955
1000	0.9954

Khi $d \gg 0$, gần như **toàn bộ không gian** là cần thiết để tìm 10-NN. Điều này phá vỡ giả định của kNN vì các điểm hàng xóm gần nhất **không thực sự gần** (và không thực sự giống) điểm kiểm tra.

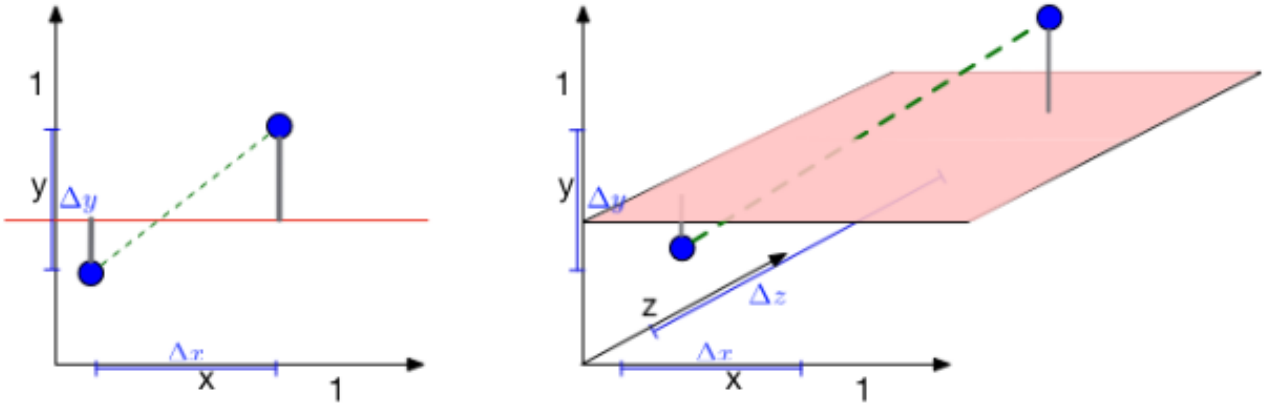
Một ý tưởng khắc phục có thể là **tăng số lượng mẫu huấn luyện** n cho đến khi hàng xóm gần nhất thực sự gần. Giả sử muốn $\ell = \frac{1}{10} = 0.1$, ta có:

$$n = k \cdot 10^d$$

Điều này tăng **theo hàm mũ**! Với $d > 100$, số điểm cần thiết còn nhiều hơn số electron trong vũ trụ.



7.1.2 Khoảng cách tới siêu phẳng (Hyperplane)



Khoảng cách giữa hai điểm ngẫu nhiên tăng mạnh khi số chiều tăng. Nhưng khoảng cách tới siêu phẳng thì sao?

Xét hình: có hai điểm màu xanh và một siêu phẳng màu đỏ.

- Với $d = 2$, khoảng cách giữa hai điểm là $\sqrt{\Delta x^2 + \Delta y^2}$.
- Khi thêm chiều thứ 3 ($d = 3$), khoảng cách trở thành $\sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$ — chắc chắn không nhỏ hơn và thường lớn hơn.

Ngược lại, **khoảng cách tới siêu phẳng không đổi** khi thêm một chiều mới nếu chiều mới này vuông góc với vector pháp tuyến của siêu phẳng. Trong không gian d chiều, có $d - 1$ chiều vuông góc với pháp tuyến này, nên di chuyển trong các chiều đó **không thay đổi** khoảng cách tới siêu phẳng.

Kết quả là: khi khoảng cách giữa các điểm trở nên rất lớn trong không gian nhiều chiều, **khoảng cách tới siêu phẳng lại tương đối rất nhỏ**.

Điều này quan trọng cho các thuật toán học máy, vì nhiều mô hình (Perceptron, SVM) đặt siêu phẳng giữa các cụm dữ liệu. Một hệ quả của lời nguyền chiều dữ liệu là **hầu hết các điểm dữ liệu nằm rất gần các siêu phẳng phân loại** → chỉ cần nhiễu nhỏ (thường khó nhận ra) là có thể đổi nhãn phân loại. Hiện tượng này chính là *mẫu đối kháng* (*adversarial samples*).

7.1.3 Xử lý Dữ liệu nhiều chiều

Xử lý dữ liệu có kích thước cao không phải là một nhiệm vụ dễ dàng. Kích thước cao có nghĩa là chúng ta cần lưu trữ các vectơ hoặc mảng với số lượng phần tử/số lượng lớn, điều này tiêu tốn một lượng lớn bộ nhớ lưu trữ. Không chỉ bộ nhớ lưu trữ, mà các thao tác tìm kiếm trong không gian kích thước cao cũng là một nút thắt cổ chai, vì thời gian và tính toán tăng theo cấp số nhân. Do đó, việc giảm kích thước dữ liệu là rất quan trọng, nhưng khi giảm kích thước, chúng ta cũng cần giữ lại được phân bố hoặc các mẫu biểu hiện trong không gian kích thước cao.

7.1.4 Giảm Kích Thước: PCA, LDA, t-SNE

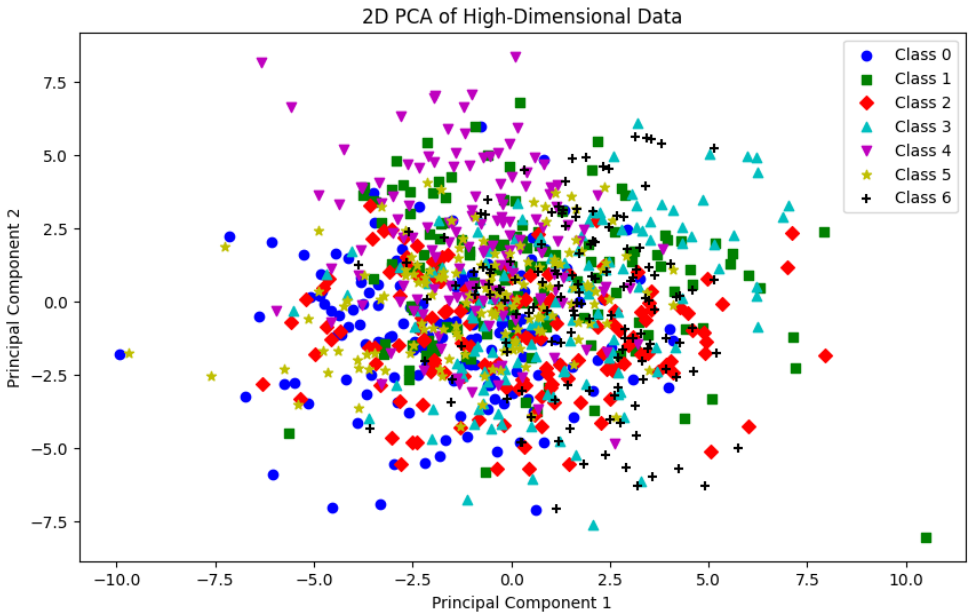
Các kỹ thuật giảm kích thước biến đổi dữ liệu có chiều cao thành không gian có chiều thấp hơn, đồng thời bảo tồn các cấu trúc quan trọng.

PCA (Phân tích Thành phần Chính)

PCA giảm chiều bằng cách chiếu dữ liệu lên các thành phần chính, những thành phần này nắm bắt được phương sai lớn nhất. Về mặt toán học, PCA tìm các vector riêng (thành phần chính) của ma trận hiệp phương sai của dữ liệu và chiếu dữ liệu lên các thành phần này.

Ví dụ mã Python:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.decomposition import PCA

# Bước 1: Tạo bộ dữ liệu tổng hợp
X_train, y_train = make_classification(n_samples=1000, n_features=768, n_informative=10,
                                       n_redundant=0, n_classes=7, random_state=42)

# Bước 2: Áp dụng PCA giảm chiều xuống 2
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)

# Bước 3: Hiển thị dữ liệu sau khi giảm chiều
plt.figure(figsize=(10, 6))
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
markers = ['o', 's', 'D', '^', 'v', '*', '+']

for i, color, marker in zip(range(7), colors, markers):
    plt.scatter(X_train_pca[y_train == i, 0], X_train_pca[y_train == i, 1],
                c=color, marker=marker, label=f'Lớp {i}')

plt.title('PCA 2 chiều của Dữ liệu Kích thước Cao')
plt.xlabel('Thành phần Chính 1')
plt.ylabel('Thành phần Chính 2')
plt.legend()
plt.show()
```



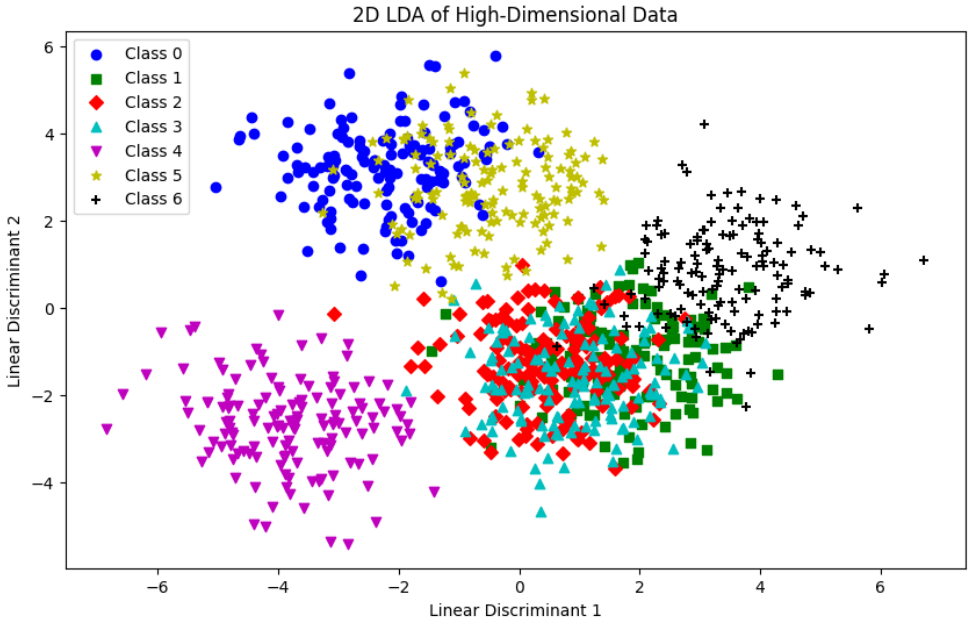
LDA (Phân tích Phân biệt Tuyến tính)

LDA giảm chiều bằng cách tối đa hóa sự phân tách giữa các lớp khác nhau. LDA tìm các tổ hợp tuyến tính của các đặc trưng giúp phân tách tốt nhất các lớp.

Ví dụ mã Python:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# Tạo bộ dữ liệu tổng hợp
X_train, y_train = make_classification(n_samples=1000, n_features=768, n_informative=10,
                                       n_redundant=0, n_classes=7, random_state=42)

# Áp dụng LDA giảm chiều xuống 2
lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train, y_train)

# Hiển thị dữ liệu sau khi giảm chiều
plt.figure(figsize=(10, 6))
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
markers = ['o', 's', 'D', '^', 'v', '*', '+']

for i, color, marker in zip(range(7), colors, markers):
    plt.scatter(X_train_lda[y_train == i, 0], X_train_lda[y_train == i, 1],
                c=color, marker=marker, label=f'Lớp {i}')

plt.title('LDA 2 chiều của Dữ liệu Kích thước Cao')
plt.xlabel('Biến Phân biệt Tuyến tính 1')
plt.ylabel('Biến Phân biệt Tuyến tính 2')
```

```
plt.legend()
plt.show()
```



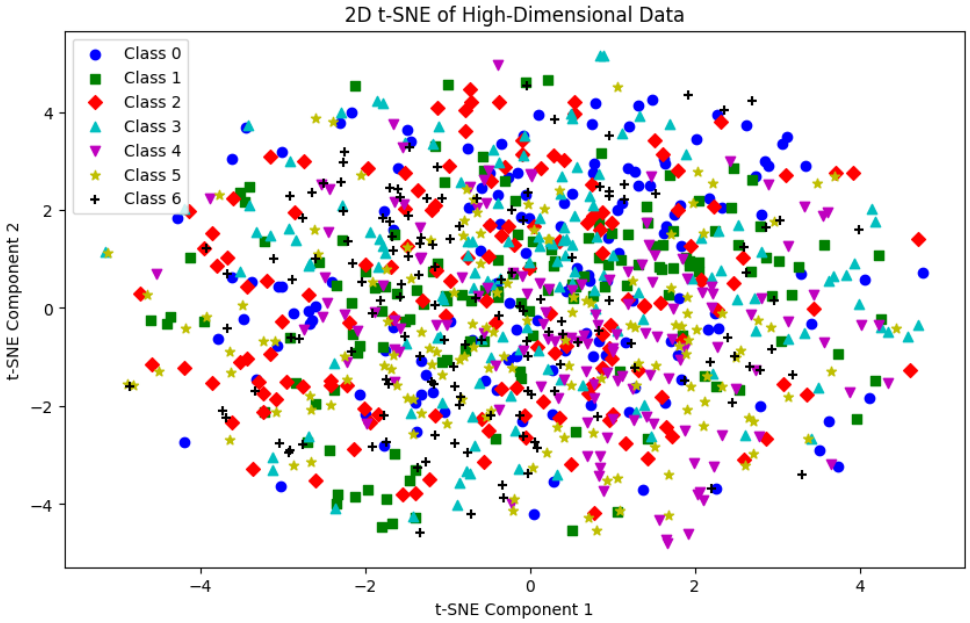
t-SNE (t-Phân phối Hàng xóm Ngẫu nhiên)

t-SNE được sử dụng để trực quan hóa dữ liệu có chiều cao bằng cách giảm chiều sao cho giữ lại thông tin về vùng lân cận. t-SNE tối thiểu hóa độ lệch Kullback-Leibler giữa xác suất kết hợp của biểu diễn thấp chiều và dữ liệu gốc, nhằm giữ cho phân bố trong không gian thấp chiều và cao chiều tương tự nhau.

Ví dụ mã Python:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.manifold import TSNE

# Tạo bộ dữ liệu tổng hợp
X_train, y_train = make_classification(n_samples=1000, n_features=768, n_informative=10,
                                       n_redundant=0, n_classes=7, random_state=42)

# Áp dụng t-SNE giảm chiều xuống 2
tsne = TSNE(n_components=2, random_state=42)
X_train_tsne = tsne.fit_transform(X_train)

# Hiển thị dữ liệu sau khi giảm chiều
plt.figure(figsize=(10, 6))
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
markers = ['o', 's', 'D', '^', 'v', '*', '+']

for i, color, marker in zip(range(7), colors, markers):
```

```
    plt.scatter(X_train_tsne[y_train == i, 0], X_train_tsne[y_train == i, 1],
                c=color, marker=marker, label=f'Lớp {i}')

plt.title('t-SNE 2 chiều của Dữ liệu Kích thước Cao')
plt.xlabel('Thành phần t-SNE 1')
plt.ylabel('Thành phần t-SNE 2')
plt.legend()
plt.show()
```



Hình 4: Enter Caption

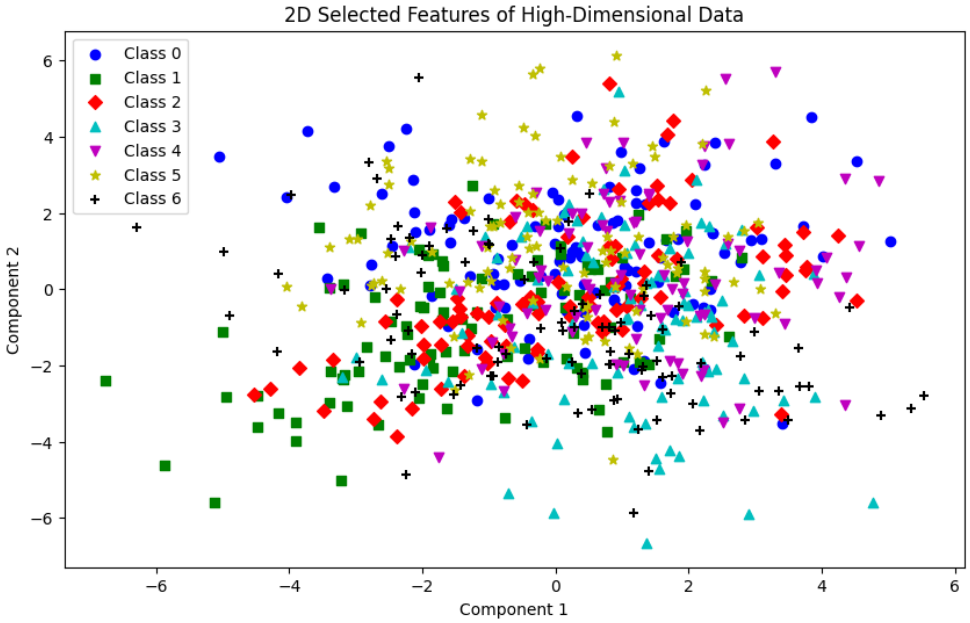
7.1.4 Phương pháp Lựa chọn Đặc trưng

Lựa chọn đặc trưng liên quan đến việc chọn ra các đặc trưng quan trọng nhất, giảm chiều bằng cách loại bỏ các đặc trưng không liên quan hoặc dư thừa. Ví dụ dưới đây tạo một bộ dữ liệu tổng hợp với 1000 mẫu, 768 đặc trưng, và 7 lớp, sau đó chia thành tập huấn luyện và tập kiểm tra. Phương pháp 'SelectKBest' với chỉ số ANOVA F-value được sử dụng để chọn ra 2 đặc trưng quan trọng nhất. Sau đó, có thể dùng dữ liệu này để huấn luyện bộ phân lớp K-Nearest Neighbors (KNN).

Ví dụ mã Python:

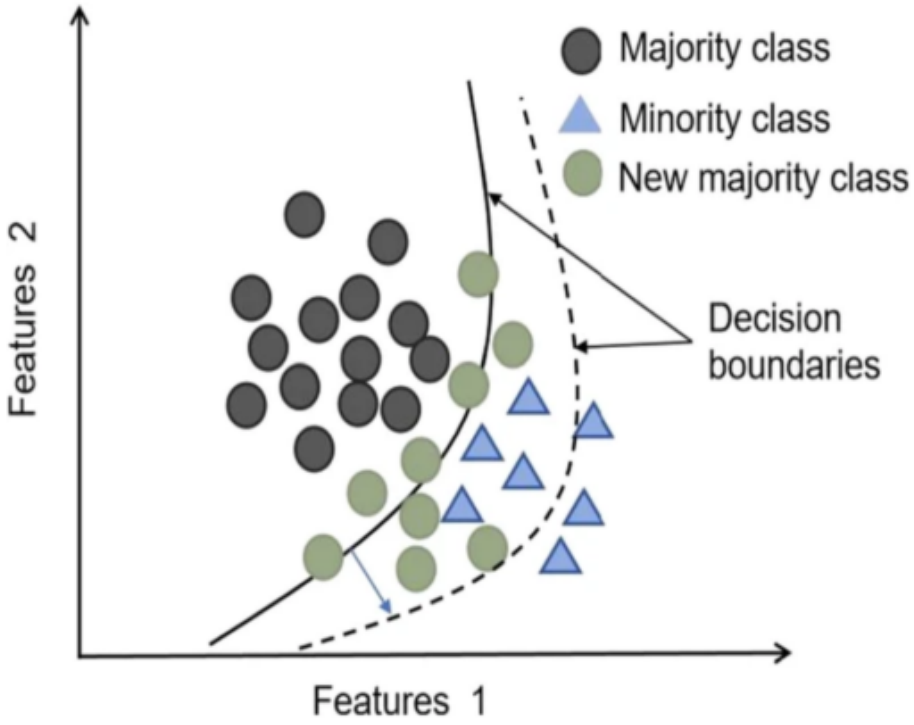
```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Tạo bộ dữ liệu tổng hợp
X, y = make_classification(n_samples=1000, n_features=768, n_informative=10,
                           n_redundant=0, n_classes=7, random_state=42)
```

```
# Chia bộ dữ liệu thành tập huấn luyện và tập kiểm tra
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Lựa chọn đặc trưng
selector = SelectKBest(score_func=f_classif, k=2)
X_train_selected = selector.fit_transform(X_train, y_train)

# Hiển thị dữ liệu sau khi lựa chọn đặc trưng
plt.figure(figsize=(10, 6))
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
markers = ['o', 's', 'D', '^', 'v', '*', '+']

for i, color, marker in zip(range(7), colors, markers):
    plt.scatter(X_train_selected[y_train == i, 0], X_train_selected[y_train == i, 1],
                c=color, marker=marker, label=f'Lớp {i}')

plt.title('2D Các Đặc trưng Được Chọn của Dữ liệu Kích thước Cao')
plt.xlabel('Thành phần 1')
plt.ylabel('Thành phần 2')
plt.legend()
plt.show()
```



7.2 Dữ liệu mất cân bằng về lớp class

Bộ phân loại k-Nearest Neighbors (kNN) dự đoán nhãn của một điểm truy vấn bằng cách bỏ phiếu đa số giữa k hàng xóm gần nhất của nó. Trong hình minh họa, một điểm mới (ở trung tâm) nhìn vào các hàng xóm có màu để bỏ phiếu cho lớp. Tuy nhiên, khi một lớp hiếm, hầu hết các hàng xóm đó có thể đến từ lớp chiếm ưu thế - tạo ra sự thiên vị trong kết quả.



Trên thực tế, kNN cổ điển là "lười biếng"và tốn kém tại thời điểm dự đoán, và hiệu suất của nó suy giảm trên dữ liệu mất cân bằng/nhiều chiều. Nó có xu hướng để lớp đa số chiếm ưu thế trong tất cả các vùng lân cận, dẫn đến dự đoán thiên vị về lớp thường xuyên.

Nhiều nghiên cứu xác nhận rằng trong khi kNN hoạt động tốt trên các tập cân bằng, độ chính xác của nó giảm mạnh khi các lớp bị lệch. Trong thực tế, điều này có nghĩa là kNN nguyên bản sẽ thường bỏ lỡ hoàn toàn lớp thiểu số, mang lại độ chính xác tổng thể cao nhưng khả năng thu hồi kém trên lớp hiếm. Kết quả có thể gây ảnh hưởng nghiêm trọng trong các bối cảnh như phát hiện gian lận hoặc chẩn đoán y tế, nơi lớp thiểu số là quan trọng.

Các Chiến lược Giải quyết

7.2.1 Bỏ phiếu kNN có trọng số Khoảng cách

Một cách đơn giản để tăng cường ảnh hưởng của thiểu số là cân bằng các hàng xóm theo khoảng cách. Theo mặc định, kNN bỏ phiếu "đồng nhất mỗi hàng xóm trong k hàng xóm có một phiếu bầu. Thay vào đó, đặt `weights='distance'` trong scikit-learn: điều này làm cho phiếu bầu của mỗi hàng xóm tỷ lệ thuận với $\frac{1}{\text{khoảng cách}}$.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5, weights='distance')
knn.fit(X_train, y_train)
```

Với `weights='distance'`, "các hàng xóm gần hơn... có ảnh hưởng lớn hơn". Trong nội bộ, bộ phân loại tính toán một mode có trọng số của các nhãn hàng xóm (tổng trọng số cho mỗi lớp) thay vì một đếm đơn giản.

Bạn cũng có thể định nghĩa một hàm cân bằng tùy chỉnh. Ví dụ, để cân bằng theo $\frac{1}{d^2}$ hoặc một hàm Gaussian:

```
import numpy as np
```

```
def gaussian_weights(distances):
    sigma = 1.0
    return np.exp(- (distances**2) / (2*sigma**2))

knn_custom = KNeighborsClassifier(n_neighbors=5, weights=gaussian_weights)
```

7.2.2 Điều chỉnh Số lượng Hàng xóm (k)

Việc lựa chọn k là quan trọng, đặc biệt với sự mất cân bằng. Một k nhỏ (như 1 hoặc 3) có nghĩa là mỗi truy vấn nhạy cảm chỉ với các hàng xóm rất gần nhất - có khả năng nắm bắt các hàng xóm thiểu số nếu có bất kỳ ai rất gần. Một k lớn hơn (như 10+) tính trung bình trên nhiều điểm, có thể làm mượt nhiễu nhưng cũng có xu hướng bao gồm nhiều điểm đa số hơn, làm loãng tín hiệu thiểu số.

Trong các cài đặt mất cân bằng, một phương pháp thường dùng là thử nghiệm với các giá trị k nhỏ hơn, cho các trường hợp thiểu số ít ỏi cơ hội tốt hơn để ảnh hưởng đến phiếu bầu.

Khi nào sử dụng: k nhỏ hơn có xu hướng làm nổi bật cấu trúc thiểu số cục bộ, có thể tăng khả năng thu hồi trên lớp hiếm, với rủi ro phương sai cao hơn. k lớn hơn có thể ổn định chống nhiễu nhưng có thể ưu ái đa số.

7.2.3 Tạo mẫu tổng hợp quá mức (SMOTE)

Một kỹ thuật mạnh mẽ ở cấp độ dữ liệu là tạo ra nhiều ví dụ thiểu số hơn. SMOTE (Synthetic Minority Over-sampling Technique) tạo ra các điểm thiểu số tổng hợp mới bằng cách nội suy giữa các điểm hiện có.

Ý tưởng cơ bản: đối với mỗi mẫu thiểu số, chọn một trong k hàng xóm thiểu số gần nhất của nó và tạo một điểm ở giữa.

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X_train, y_train)
print("Before: ", sorted(Counter(y_train).items()))
print("After: ", sorted(Counter(y_res).items()))
```

Khi nào sử dụng: SMOTE phù hợp khi bạn có đủ mẫu thiểu số gốc để nội suy có ý nghĩa là có thể. Nó giả định lớp thiểu số có phần nào liên tục trong không gian đặc trưng.

Mẹo triển khai: Sử dụng imblearn trong một pipeline để tránh rò rỉ dữ liệu:

```
from imblearn.pipeline import Pipeline
pipe = Pipeline([
    ('smote', SMOTE(random_state=42)),
    ('knn', KNeighborsClassifier(n_neighbors=5))
])
pipe.fit(X_train, y_train)
```

7.2.4 Giảm mẫu Lớp đa số

Đôi khi việc loại bỏ các ví dụ đa số có thể giúp cân bằng dữ liệu mà không cần thêm các điểm tổng hợp. Các phương pháp giảm mẫu (ngẫu nhiên hoặc có thông tin) bỏ một số mẫu đa số, làm cho các lớp cân bằng hơn.

```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(sampling_strategy='auto', random_state=42)
X_under, y_under = rus.fit_resample(X_train, y_train)
knn = KNeighborsClassifier(n_neighbors=5).fit(X_under, y_under)
```

Đánh đổi: Giảm mẫu tránh dữ liệu tổng hợp, nhưng với chi phí loại bỏ dữ liệu có thể hữu ích. Nếu lớp đa số của bạn cực kỳ lớn, giảm mẫu ngẫu nhiên có thể hiệu quả và nhanh.

7.2.5 Điều chỉnh Ngưỡng Quyết định

kNN có thể tạo ra đầu ra xác suất (ví dụ: sử dụng `predict_proba`, cho tỷ lệ hàng xóm bỏ phiếu cho mỗi lớp). Thay vì sử dụng ngưỡng mặc định 0.5 để gán nhãn dương (thiểu số), bạn có thể dịch chuyển ngưỡng để ưu ái thiểu số.

```
probs = knn.predict_proba(X_val)[:, 1]
for thr in np.linspace(0,1,101):
    preds = (probs > thr).astype(int)
    f1 = f1_score(y_val, preds)
```

Khi nào sử dụng: Điều chỉnh ngưỡng là một bước hậu xử lý trên kNN đã được huấn luyện. Nó đặc biệt hữu ích trong các vấn đề nhị phân nơi false negative tốn kém.

7.2.6 Phương pháp Ensemble với kNN

Ensembling có thể giảm thiểu mất cân bằng lớp bằng cách kết hợp nhiều mô hình. Hai cách tiếp cận phổ biến: bagging và boosting.

Bagging KNN: Huấn luyện nhiều kNN trên các tập con khác nhau của dữ liệu (với thay thế). Trong các biến thể bagging cân bằng, mỗi mẫu bootstrap được cân bằng bằng cách sử dụng over/undersampling.

```
#BalancedBaggingClassifier
from imblearn.ensemble import BalancedBaggingClassifier
balanced_bagging = BalancedBaggingClassifier(
    base_estimator=KNeighborsClassifier(),
    n_estimators=10,
    random_state=42
)
```

7.3 Dữ liệu có sự khác biệt về thang đo

KNN dùng **khoảng cách** giữa điểm cần dự đoán và các điểm trong tập huấn luyện. Nếu một đặc trưng có **thang đo lớn** (ví dụ: thu nhập 0–100,000) và một đặc trưng khác có **thang đo nhỏ** (ví dụ: tuổi 0–100), thì thang đo lớn sẽ **chi phối** các trị tuyệt đối của hiệu số và do đó **chi phối toàn bộ khoảng cách**. Kết quả: kNN "quan tâm"nhiều đến đặc trưng lớn và hầu như bỏ qua đặc trưng nhỏ — điều này thường là không mong muốn.

7.3.1 Vì sao lại chi phối? (công thức + minh họa số)

Công thức khoảng cách Euclidean

Khoảng cách Euclidean giữa hai vectơ $\mathbf{x}$ và $\mathbf{y}$ là:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_i (x_i - y_i)^2} \tag{1}$$

Nếu với một chỉ số j ta có x_j ở thang đo lớn, thì $(x_j - y_j)^2$ có thể lớn gấp hàng triệu lần các $(x_i - y_i)^2$ khác — nên tổng và căn bậc hai bị chi phối bởi thành phần đó.

Ví dụ số (tính rõ từng bước)

Giả sử chỉ có 2 đặc trưng: Income (đơn vị VND), Age (năm).

- Điểm cần dự đoán: $\mathbf{x} = [50000, 30]$ (income = 50,000; age = 30)

- Neighbor A: $\mathbf{a} = [60000, 31]$
- Neighbor B: $\mathbf{b} = [40000, 35]$

Tính khoảng cách đến Neighbor A

$$\begin{aligned}
\text{Hiệu income:} &\quad 60000 - 50000 = 10000 &(2)\\
\text{Bình phương:} &\quad 10000^2 = 100,000,000 &(3)\\
\text{Hiệu age:} &\quad 31 - 30 = 1 &(4)\\
\text{Bình phương:} &\quad 1^2 = 1 &(5)\\
\text{Tổng bình phương:} &\quad 100,000,000 + 1 = 100,000,001 &(6)\\
\text{Khoảng cách:} &\quad \sqrt{100,000,001} \approx 10,000.00005 &(7)
\end{aligned}$$

Tính khoảng cách đến Neighbor B

$$\begin{aligned}
\text{Hiệu income:} &\quad 40000 - 50000 = -10000 &(8)\\
\text{Bình phương:} &\quad 10000^2 = 100,000,000 &(9)\\
\text{Hiệu age:} &\quad 35 - 30 = 5 &(10)\\
\text{Bình phương:} &\quad 5^2 = 25 &(11)\\
\text{Tổng bình phương:} &\quad 100,000,000 + 25 = 100,000,025 &(12)\\
\text{Khoảng cách:} &\quad \sqrt{100,000,025} \approx 10,000.00125 &(13)
\end{aligned}$$

Kết quả: Khoảng cách chủ yếu do **income** quyết định; hiệu số tuổi (1 vs 5) chỉ thay đổi khoảng cách ở phần thập phân rất nhỏ — tức tuổi gần như **không ảnh hưởng** tới quyết định chọn neighbor.

Sau khi chuẩn hóa về [0,1]

Bây giờ chuẩn hóa về [0,1] (ví dụ Min–Max: income / 100,000; age / 100):

- $\mathbf{x}' = [0.5, 0.30]$
- $\mathbf{a}' = [0.6, 0.31]$
- $\mathbf{b}' = [0.4, 0.35]$

Khoảng cách đến A sau chuẩn hóa

$$\begin{aligned}
\text{Diff income:} &\quad 0.6 - 0.5 = 0.1 \to 0.1^2 = 0.01 &(14)\\
\text{Diff age:} &\quad 0.31 - 0.30 = 0.01 \to 0.01^2 = 0.0001 &(15)\\
\text{Sum:} &\quad 0.01 + 0.0001 = 0.0101 &(16)\\
d &\quad \approx \sqrt{0.0101} \approx 0.1005 &(17)
\end{aligned}$$

Khoảng cách đến B sau chuẩn hóa

$$\text{Diff income:} \quad 0.4 - 0.5 = -0.1 \to 0.01 \tag{18}$$

$$\text{Diff age:} \quad 0.35 - 0.30 = 0.05 \to 0.0025 \tag{19}$$

$$\text{Sum:} \quad 0.01 + 0.0025 = 0.0125 \tag{20}$$

$$d \quad \approx \sqrt{0.0125} \approx 0.1118 \tag{21}$$

Sau khi chuẩn hóa, **thành phần tuổi đóng góp rõ rệt** vào khoảng cách — tức kNN giờ cân bằng hơn giữa hai đặc trưng.

7.3.2 Hậu quả thực tế trên kNN

Lựa chọn hàng xóm sai

Những điểm giống về đặc tính quan trọng nhưng khác về thang đo lớn sẽ bị bỏ qua.

Quyết định phân lớp bị lệch

Ranh phân lớp phản ánh đặc trưng lớn, dẫn tới giảm hiệu quả (đặc biệt nếu đặc trưng nhỏ là quan trọng cho phân biệt lớp).

Độ nhạy với nhiễu và outlier

Nếu thang đo lớn có outlier, các khoảng cách sẽ bị méo mạnh.

Ảnh hưởng đến các metric xác suất

`predict_proba` (tỉ lệ phiếu láng giềng) bị chi phối bởi neighbors theo thang đo lớn.

7.3.3 Chuẩn hóa và Scaling

Chuẩn hóa là bước quan trọng nhất trong Feature Engineering cho KNN vì thuật toán dựa vào khoảng cách Euclidean.

7.3.3.1 Min-Max Scaling

Chuyển đổi dữ liệu về khoảng [0, 1]:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{22}$$

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np

X = np.array([[1, 2000], [2, 3000], [3, 1000]])

# Min-Max Scaling
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
print(" D    l i u   g c :\n", X)
print("Sau Min-Max Scaling:\n", X_scaled)
```

7.3.3.2 Standard Scaling (Z-score Normalization)

Chuyển đổi dữ liệu về phân phối chuẩn với mean=0 và std=1:

$$X_{scaled} = \frac{X - \mu}{\sigma} \tag{23}$$

```
from sklearn.preprocessing import StandardScaler

# Standard Scaling
standard_scaler = StandardScaler()
X_standard = standard_scaler.fit_transform(X)
print("Sau Standard Scaling:\n", X_standard)
```

7.3.3.3 Robust Scaling

Sử dụng median và IQR, ít bị ảnh hưởng bởi ngoại lai:

$$X_{scaled} = \frac{X - median(X)}{IQR(X)} \tag{24}$$

```
from sklearn.preprocessing import RobustScaler

# Robust Scaling
robust_scaler = RobustScaler()
X_robust = robust_scaler.fit_transform(X)
print("Sau Robust Scaling:\n", X_robust)
```

7.3.3.4 So sánh các phương pháp scaling

Trường hợp dữ liệu	**Phương pháp scaling nên dùng**	**Giải thích**
Có nhiều outlier	`RobustScaler`	Dùng median và IQR thay vì mean và std, nên ít bị ảnh hưởng bởi giá trị cực đoan.
Muốn đưa giá trị về một khoảng cụ thể (ví dụ [0,1])	`MinMaxScaler`	Bảo toàn hình dạng phân phối nhưng thay đổi biên, hữu ích cho các thuật toán yêu cầu phạm vi cố định (như mạng nơ-ron, KNN).
Phân phối gần chuẩn (Gaussian)	`StandardScaler`	Đưa dữ liệu về mean = 0, std = 1, phù hợp cho mô hình giả định dữ liệu chuẩn hóa (như Logistic Regression, SVM).
Phân phối không chuẩn, muốn giảm tác động của outlier nhẹ	`MaxAbsScaler`	Chia cho giá trị tuyệt đối lớn nhất, phù hợp dữ liệu thưa (sparse).
Không chắc phân phối thế nào	Thử cả `StandardScaler` và `RobustScaler` rồi đánh giá trên cross-validation	Vì ảnh hưởng của scaling phụ thuộc mô hình và đặc điểm dữ liệu.

Bảng 2: Lựa chọn phương pháp scaling phù hợp với dữ liệu

7.4 Outliers và Noise

Outliers (giá trị ngoại lai) và noise (nhiễu, bao gồm lỗi trong features hoặc label) là hai dạng sai lệch dữ liệu thường gặp. Với K-Nearest Neighbors (kNN), vì thuật toán quyết định dựa trên *khoảng cách* tới các láng giềng cục bộ, nên những điểm bất thường hoặc nhãn sai có thể gây ảnh hưởng mạnh đến kết quả phân loại/ước lượng.

kNN thường dùng khoảng cách Euclid (hoặc các metric khác). Với hai vector mẫu $x, y \in \mathbb{R}^p$:

$$d(x,y) \;=\; \sqrt{\sum_{i=1}^{p}(x_i - y_i)^2}.$$

Nếu một chiều j có thang đo lớn (hoặc có outlier cực lớn), thì thành phần $(x_j - y_j)^2$ sẽ chiếm áp đảo tổng trên, làm cho các chiều khác không còn ảnh hưởng đáng kể lên $d(x,y)$. Kết quả là kNN "quan tâm" chủ yếu đến chiều có giá trị lớn hoặc outlier, dẫn tới lựa chọn láng giềng sai.

7.4.1 Ví dụ

Giả sử chỉ có hai đặc trưng: Income (đơn vị VND) và Age (năm). Điểm cần phân loại:

$$x = [50000,\ 30].$$

Hai láng giềng:

$$a = [60000,\ 31], \qquad b = [40000,\ 35].$$

Tính khoảng cách Euclid (không chuẩn hoá):

$$d(x,a)^2 = (60000-50000)^2 + (31-30)^2 = 10000^2 + 1^2 = 100000000 + 1 = 100000001,$$
$$d(x,a) = \sqrt{100000001} \approx 10000.00005.$$

$$d(x,b)^2 = (40000-50000)^2 + (35-30)^2 = (-10000)^2 + 5^2 = 100000000 + 25 = 100000025,$$
$$d(x,b) = \sqrt{100000025} \approx 10000.00125.$$

Như vậy, sự khác biệt về tuổi (1 vs 5) gần như không ảnh hưởng vì Income (thang lớn) chi phối.

Nếu chuẩn hoá Min–Max (Income chia cho 100000, Age chia cho 100):

$$x' = [0.5,\ 0.30], \quad a' = [0.6,\ 0.31], \quad b' = [0.4,\ 0.35].$$

Tính lại:

$$d(x',a')^2 = (0.6-0.5)^2 + (0.31-0.30)^2 = 0.1^2 + 0.01^2 = 0.01 + 0.0001 = 0.0101,$$
$$d(x',a') = \sqrt{0.0101} \approx 0.1005.$$

$$d(x',b')^2 = (0.4-0.5)^2 + (0.35-0.30)^2 = 0.1^2 + 0.05^2 = 0.01 + 0.0025 = 0.0125,$$
$$d(x',b') = \sqrt{0.0125} \approx 0.1118.$$

Sau khi chuẩn hoá, thành phần tuổi có đóng góp rõ rệt hơn và quyết định láng giềng gần nhất trở nên hợp lý hơn.

7.4.2 Nguồn gốc và kiểu noise

- **Noise trong features:** lỗi đo lường, giá trị bị nhập sai, hoặc dữ liệu thiếu chuyển đổi đơn vị.
- **Outliers:** sự khác biệt thực sự (rare event) hoặc dữ liệu sai (erroneous entry).
- **Noise trong labels:** nhãn bị gán sai (label flip), do lỗi con người hoặc nhãn mơ hồ.

7.4.3 Hệ quả cụ thể cho kNN

- **Chọn sai láng giềng:** Một outlier có khoảng cách nhỏ tới mẫu mới có thể chi phối phiếu bầu.
- **Biên phân lớp bị méo:** Ranh giới quyết định trở nên phức tạp/không chính xác do vài điểm nhiễu.
- **Giảm độ tin cậy của xác suất:** `predict_proba` (tỉ lệ phiếu) bị ảnh hưởng mạnh khi trong neighborhood có nhiều nhiễu.
- **Đối với regression:** outliers làm tăng MSE do ảnh hưởng bình phương trên sai số.
- **Trong không gian nhiều chiều:** outliers còn nguy hiểm hơn vì curse of dimensionality khiến mọi điểm đều tương đối xa — outlier càng dễ chi phối.

7.4.4 Các phương pháp xử lý và khuyến nghị

7.4.4.1 Tiền xử lý (preprocessing)

- **Phát hiện và loại outlier:** dùng IQR rule, z-score, hoặc phương pháp dựa trên mật độ (DBSCAN, Isolation Forest). Nếu outlier là lỗi, loại bỏ; nếu là sự kiện hợp lệ, cân nhắc giữ nhưng xử lý đặc biệt.
- **Scaling robust:** dùng `RobustScaler` (median + IQR) nếu có nhiều outlier; hoặc `StandardScaler` / `MinMaxScaler` nếu không có outlier.

7.4.4.2 Làm sạch nhãn (label cleaning)

- **Edited Nearest Neighbours (ENN):** loại bỏ mẫu mà nhãn của nó khác với nhãn đa số của k láng giềng — thường giúp loại label noise.
- **Tomek Links:** cặp hai mẫu thuộc 2 lớp khác nhau mà là nearest neighbor của nhau; xoá mẫu của class đa số để làm biên rõ hơn.

7.4.4.3 Thay đổi cách tính khoảng cách / bỏ ảnh hưởng outlier

- **Mahalanobis distance:**
$$d_M(x, y) = \sqrt{(x - y)^\top S^{-1}(x - y)},$$
trong đó S là ma trận hiệp phương sai. Mahalanobis chuẩn hoá theo phương sai và loại bỏ tương quan giữa các chiều.
- **Hàm trọng số từ khoảng cách:** dùng `weights='distance'` hoặc hàm Gaussian để giảm ảnh hưởng của các láng giềng xa/không liên quan.

7.4.4.4 Thay đổi cấu trúc mẫu / mô hình

- **Tăng k được chọn cẩn trọng:** k lớn làm mượt biên và giảm ảnh hưởng của một vài outlier, nhưng k quá lớn có thể làm mất chi tiết.
- **Ensemble / bagging với resampling:** mỗi bag cân bằng hoặc làm sạch dữ liệu sẽ giảm thiểu tác động của nhiễu.
- **Chuyển qua mô hình ít nhạy đến scaling/outlier:** ví dụ Random Forest, có thể chịu outlier tốt hơn.

7.4.4.5 Ví dụ pipeline (Python, scikit-learn + imbalanced-learn)

```python
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.neighbors import KNeighborsClassifier
from imblearn.under_sampling import EditedNearestNeighbours

pipe = make_pipeline(
    RobustScaler(),                         # reduce impact of outliers on features
    EditedNearestNeighbours(),              # Edit noise labels
    KNeighborsClassifier(
        n_neighbors=5,
        weights='distance'                  # weight voting
    )
)

pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
```

7.4.4.6 Chú ý thực nghiệm và đánh giá

- **Cross-validation stratified:** dùng StratifiedKFold để giữ tỉ lệ lớp khi đánh giá.
- **Metrics phù hợp:** với imbalance hoặc khi minority quan trọng, dùng Precision/Recall/F1, AUC-PR, hoặc Balanced Accuracy thay vì chỉ dùng accuracy.
- **Không rò rỉ dữ liệu:** fit tất cả bộ lọc/Scaler chỉ trên train trong mỗi split CV (dùng pipeline để tránh leakage).

7.5 Feature Correlation

7.5.1 Tác động của Feature Correlation lên KNN

Trong nhiều tập dữ liệu, đặc trưng (features) thường có tương quan cao với nhau. Ví dụ, trong dữ liệu về nhà đất, Diện tích và Số phòng ngủ có thể cùng tăng khi nhà lớn hơn.

Khi tương quan quá cao, ta gặp hiện tượng multicollinearity (đa cộng tuyến), dẫn đến:

- **Méo khoảng cách**: Với các thuật toán dựa trên khoảng cách (KNN, K-Means...), hai đặc trưng giống nhau khiến một hướng trong không gian bị "kéo" mạnh, làm kết quả đo khoảng cách mất cân bằng.
- **Tăng dimensionality không cần thiết**: Các feature gần như trùng lặp không cung cấp thêm thông tin nhưng vẫn tăng số chiều, làm mô hình phức tạp hơn (curse of dimensionality).
- **Gây instability**: Trong các mô hình tuyến tính (Linear Regression, Logistic Regression), hệ số ước lượng (coefficients) có thể thay đổi lớn khi dữ liệu biến động nhẹ.
- **Redundant Information Amplification:**
  - Features tương quan cao cung cấp thông tin trùng lặp
  - Trong ví dụ: area_m2, area_ft, area_double về cơ bản là cùng 1 thông tin
  - Chúng chiếm 40% trọng số trong distance calculation
- **Dilution of Important Signals:**
  - Features thực sự quan trọng (rooms, age, distance_cbd) bị "pha loãng"
  - KNN khó phân biệt giữa những điểm thực sự khác biệt về business logic
- **Computational Inefficiency:**
  - Tính toán khoảng cách trên nhiều features redundant
  - Increased memory usage và training time

7.5.2 Tình huống: Dự đoán Giá Nhà

Giả sử chúng ta có dataset về giá nhà với các features sau:

- **Area (m^2):** Diện tích nhà
- **Area_ft:** Diện tích nhà ($feet^2$) = Area $\times$ 10.764
- **Rooms:** Số phòng
- **Age:** Tuổi nhà
- **Distance_CBD:** Khoảng cách đến trung tâm (km)

Vấn đề: Area và Area_ft có correlation = 1.0 (perfect correlation)

Tạo Dữ liệu Mẫu

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Set random seed cho reproducibility
np.random.seed(42)

n_samples = 1000

# Feature
area_m2 = np.random.normal(150, 50, n_samples)  # 100-200 m
rooms = np.random.randint(1, 6, n_samples)      # 1-5 rooms
age = np.random.randint(0, 50, n_samples)       # 0-50 years old
distance_cbd = np.random.normal(10, 5, n_samples)  # 5-15 km

# Features have high correlation with area_m2
area_ft = area_m2 * 10.764  # Perfect correlation (r = 1.0)
area_double = area_m2 * 2   # Perfect correlation (r = 1.0)

# Logic business
price_score = (
    area_m2 * 0.01 +
    rooms * 10 +
    (50 - age) * 0.5 +
    (20 - distance_cbd) * 2
)

# 3 Class: Low, Medium, High price
price_class = pd.cut(price_score, bins=3, labels=['Low', 'Medium', 'High'])

# Create DataFrame
df = pd.DataFrame({
    'area_m2': area_m2,
    'area_ft': area_ft,        # Highly correlated  v i  area_m2
    'area_double': area_double, # Highly correlated  v i  area_m2
    'rooms': rooms,
    'age': age,
    'distance_cbd': distance_cbd,
    'price_class': price_class
})

print("Dataset shape:", df.shape)
print("\nFirst 5 rows:")
print(df.head())
```

Phân tích Correlation

```
# Correlation matrix
correlation_matrix = df.drop('price_class', axis=1).corr()

print("Correlation Matrix:")
print(correlation_matrix.round(3))
```

```

# Visualize correlation
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='RdBu', center=0,
            square=True, fmt='.3f')
plt.title('Feature Correlation Matrix')
plt.tight_layout()
plt.show()

# High correlation pairs
def find_high_correlation_pairs(corr_matrix, threshold=0.8):
    high_corr_pairs = []
    for i in range(len(corr_matrix.columns)):
        for j in range(i+1, len(corr_matrix.columns)):
            corr_val = abs(corr_matrix.iloc[i, j])
            if corr_val > threshold:
                high_corr_pairs.append(
                    (corr_matrix.columns[i],
                     corr_matrix.columns[j],
                     corr_val)
                )
    return high_corr_pairs

high_corr = find_high_correlation_pairs(correlation_matrix, 0.9)
print("\nHigh correlation pairs (>0.9):")
for pair in high_corr:
    print(f"{pair[0]} - {pair[1]}: {pair[2]:.3f}")
```

Kết quả mong đợi:

```
High correlation pairs (>0.9):
area_m2 - area_ft: 1.000
area_m2 - area_double: 1.000
area_ft - area_double: 1.000
```

Test KNN với Original Data (có correlation)

```
X = df.drop('price_class', axis=1)
y = df['price_class']

# Encode target labels
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)

# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Used features:", X.columns.tolist())
print("Training set shape:", X_train_scaled.shape)

# Test KNN with original data
```

```
knn_original = KNeighborsClassifier(n_neighbors=5, weights='distance')
knn_original.fit(X_train_scaled, y_train)

# Cross-validation score
cv_scores_original = cross_val_score(
    knn_original, X_train_scaled, y_train, cv=5, scoring='accuracy'
)

# Test set performance
y_pred_original = knn_original.predict(X_test_scaled)
test_accuracy_original = accuracy_score(y_test, y_pred_original)

print(f"\n=== KNN with High Correlation Data ===")
print(f"CV Accuracy: {cv_scores_original.mean():.4f} (+/- {cv_scores_original.std()
    *2:.4f})")
print(f"Test Accuracy: {test_accuracy_original:.4f}")
```

Minh họa Tác động của Correlation

```
# Calculate distance to see the impact of correlation
point1 = np.array([100, 1076.4, 200, 3, 10, 5])  # area_m2=100, area_ft=1076.4,
    area_double=200
point2 = np.array([101, 1087.2, 202, 3, 10, 5])  # area_m2=101, area_ft=1087.2,
    area_double=202

point1_scaled = scaler.transform([point1])[0]
point2_scaled = scaler.transform([point2])[0]

# Euclidean distance
distance = np.sqrt(np.sum((point1_scaled - point2_scaled) ** 2))

print(f"\n=== PH N T CH K H O NG C CH ===")
print(f"Point 1 (scaled): {point1_scaled}")
print(f"Point 2 (scaled): {point2_scaled}")
print(f"Euclidean distance: {distance:.4f}")

# Ph n t ch contribution c a t ng feature
feature_names = X.columns
diff_squared = (point1_scaled - point2_scaled) ** 2
for i, feature in enumerate(feature_names):
    contribution = diff_squared[i]
    percentage = (contribution / diff_squared.sum()) * 100
    print(f"{feature}: {contribution:.6f} ({percentage:.1f}%)")

print(f"\ n T ng c ng : {diff_squared.sum():.6f}")
print(f"Distance: {np.sqrt(diff_squared.sum()):.6f}")
```

Kết quả dự kiến:

```
=== PHÂN TÍCH KHOẢNG CÁCH ===
area_m2: 0.000400 (13.3%)
area_ft: 0.000400 (13.3%)
area_double: 0.000400 (13.3%)
rooms: 0.000000 (0.0%)
age: 0.000000 (0.0%)
distance_cbd: 0.000000 (0.0%)
```

Vấn đề: 3 features về diện tích (area_m2, area_ft, area_double) cung cấp thông tin giống hệt nhau nhưng chiếm 40% trọng số trong khoảng cách!

Remove Correlated Features

```
# Loi b features c correlation cao
def remove_highly_correlated_features(df, threshold=0.9):
    corr_matrix = df.corr().abs()

    # Tm upper triangle ca correlation matrix
    upper_triangle = corr_matrix.where(
        np.triu(np.ones(corr_matrix.shape), k=1).astype(bool)
    )

    # Tm features cn drop
    to_drop = [column for column in upper_triangle.columns
               if any(upper_triangle[column] > threshold)]

    return df.drop(columns=to_drop), to_drop

# Remove correlated features
X_reduced, dropped_features = remove_highly_correlated_features(
    X, threshold=0.9
)

print(f"Dropped features: {dropped_features}")
print(f"Remaining features: {X_reduced.columns.tolist()}")
print(f"Original features: {X.shape[1]} -> Reduced: {X_reduced.shape[1]}")

# Re-split vi reduced features
X_train_red, X_test_red, y_train_red, y_test_red = train_test_split(
    X_reduced, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)

# Re-scale
scaler_red = StandardScaler()
X_train_red_scaled = scaler_red.fit_transform(X_train_red)
X_test_red_scaled = scaler_red.transform(X_test_red)
```

Phân tích Detailed về Distance Impact

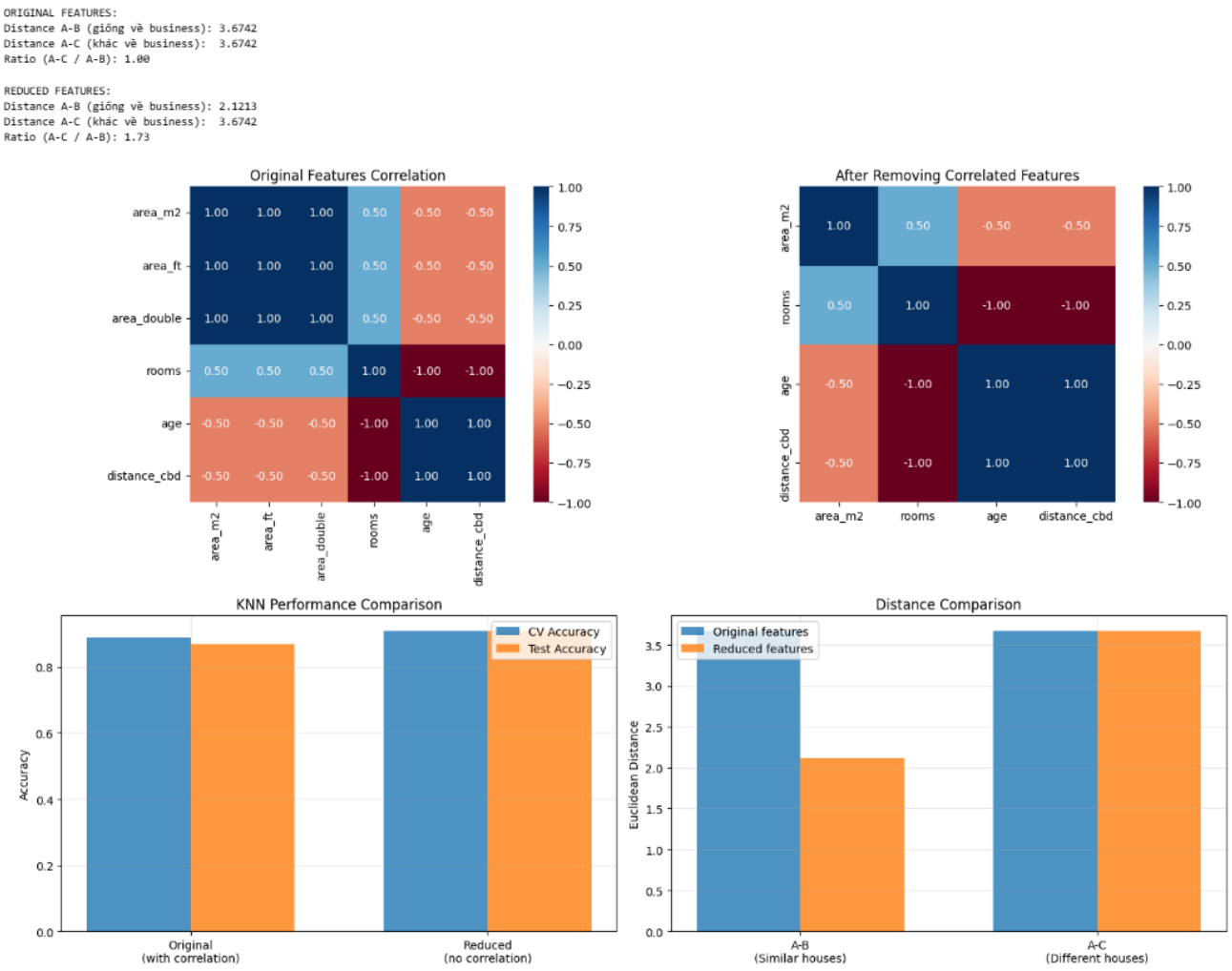
```
print("\n=== CASE STUDY: DISTANCE IMPACT ===")

# Case 1: Same business logic but different correlated features
house_A = {
    'area_m2': 120,
    'area_ft': 120 * 10.764,
    'area_double': 120 * 2,
    'rooms': 3,
    'age': 5,
    'distance_cbd': 8
}

house_B = {
    'area_m2': 121,
    'area_ft': 121 * 10.764,
    'area_double': 121 * 2,
    'rooms': 3,
    'age': 5,
    'distance_cbd': 8
}

```

```
# Case 2: Different business logic
house_C = {
    'area_m2': 120,
    'area_ft': 120 * 10.764,
    'area_double': 120 * 2,
    'rooms': 2,
    'age': 20,
    'distance_cbd': 15
}

# Convert to arrays
houses_original = np.array([
    list(house_A.values()),
    list(house_B.values()),
    list(house_C.values())
])

houses_reduced = np.array([
    [house_A['area_m2'], house_A['rooms'], house_A['age'], house_A['distance_cbd']],
    [house_B['area_m2'], house_B['rooms'], house_B['age'], house_B['distance_cbd']],
    [house_C['area_m2'], house_C['rooms'], house_C['age'], house_C['distance_cbd']]
])

# Scale houses
houses_original_scaled = scaler.transform(houses_original)
houses_reduced_scaled = scaler_red.transform(houses_reduced)

# Calculate distances
from scipy.spatial.distance import euclidean

# Distance A-B vs A-C  with original features
dist_AB_orig = euclidean(houses_original_scaled[0], houses_original_scaled[1])
dist_AC_orig = euclidean(houses_original_scaled[0], houses_original_scaled[2])

# Distance A-B vs A-C with reduced features
dist_AB_red = euclidean(houses_reduced_scaled[0], houses_reduced_scaled[1])
dist_AC_red = euclidean(houses_reduced_scaled[0], houses_reduced_scaled[2])

print(f"ORIGINAL FEATURES:")
print(f"Distance A-B ( g i ng   v   business): {dist_AB_orig:.4f}")
print(f"Distance A-C (kh c  v   business):  {dist_AC_orig:.4f}")
print(f"Ratio (A-C / A-B): {dist_AC_orig / dist_AB_orig:.2f}")

print(f"\nREDUCED FEATURES:")
print(f"Distance A-B ( g i ng   v   business): {dist_AB_red:.4f}")
print(f"Distance A-C (kh c  v   business):  {dist_AC_red:.4f}")
print(f"Ratio (A-C / A-B): {dist_AC_red / dist_AB_red:.2f}")

print(f"\n=== PH N  T CH ===")
if dist_AC_orig / dist_AB_orig < dist_AC_red / dist_AB_red:
    print("    Sau khi remove correlation: KNN ph n  b i t   t t  h n  g i a ")
    print("   nh   g i ng  vs kh c  v    m t  business logic!")
```

Visualize Impact

```
#  T o  visualization cho impact
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# 1. Correlation heatmap  t r   c  v  sau
```

```
sns.heatmap(correlation_matrix, annot=True, cmap='RdBu', center=0,
            ax=axes[0,0], square=True, fmt='.2f')
axes[0,0].set_title('Original Features Correlation')

corr_reduced = X_reduced.corr()
sns.heatmap(corr_reduced, annot=True, cmap='RdBu', center=0,
            ax=axes[0,1], square=True, fmt='.2f')
axes[0,1].set_title('After Removing Correlated Features')

# 2. Accuracy comparison
methods = ['Original\n(with correlation)', 'Reduced\n(no correlation)']
cv_scores = [cv_scores_original.mean(), cv_scores_reduced.mean()]
test_scores = [test_accuracy_original, test_accuracy_reduced]

x = np.arange(len(methods))
width = 0.35

axes[1,0].bar(x - width/2, cv_scores, width, label='CV Accuracy', alpha=0.8)
axes[1,0].bar(x + width/2, test_scores, width, label='Test Accuracy', alpha=0.8)
axes[1,0].set_ylabel('Accuracy')
axes[1,0].set_title('KNN Performance Comparison')
axes[1,0].set_xticks(x)
axes[1,0].set_xticklabels(methods)
axes[1,0].legend()
axes[1,0].grid(True, alpha=0.3)

# 3. Distance comparison
distance_types = ['A-B\n(Similar houses)', 'A-C\n(Different houses)']
orig_distances = [dist_AB_orig, dist_AC_orig]
red_distances = [dist_AB_red, dist_AC_red]

x = np.arange(len(distance_types))
axes[1,1].bar(x - width/2, orig_distances, width, label='Original features', alpha
    =0.8)
axes[1,1].bar(x + width/2, red_distances, width, label='Reduced features', alpha=0.8)
axes[1,1].set_ylabel('Euclidean Distance')
axes[1,1].set_title('Distance Comparison')
axes[1,1].set_xticks(x)
axes[1,1].set_xticklabels(distance_types)
axes[1,1].legend()
axes[1,1].grid(True, alpha=0.3)

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



Hình 5: Ảnh hưởng feature correlation đến performance của KNN

7.5.3 Cách xử lý Feature Correlation

Khi phát hiện các features có correlation cao, chúng ta có nhiều cách tiếp cận để xử lý vấn đề này. Dưới đây là các phương pháp chính được sử dụng trong thực tế.

7.5.3.1 Loại bỏ Feature dư thừa

Phương pháp đơn giản và hiệu quả nhất là loại bỏ các features có correlation cao với nhau. Thông thường, chúng ta giữ lại feature có tương quan mạnh nhất với target variable hoặc feature có ý nghĩa business quan trọng hơn.

```
import numpy as np
import pandas as pd

def remove_correlated_features(X, threshold=0.8):
    """
    Remove features with high correlation

    Parameters:
    -----------
    X : array-like
```

```
        Feature matrix
    threshold : float
        Correlation threshold for removal (default: 0.8)

    Returns:
    --------
    X_reduced : array
        Feature matrix after removal
    dropped_features : list
        List of indices of dropped features
    """
    corr_matrix = np.corrcoef(X.T)
    corr_df = pd.DataFrame(corr_matrix)

    # Create upper triangle to avoid duplicate comparisons
    upper_tri = corr_df.where(
        np.triu(np.ones(corr_df.shape), k=1).astype(bool)
    )

    # Find features with correlation > threshold
    to_drop = [column for column in upper_tri.columns
               if any(abs(upper_tri[column]) > threshold)]

    # Remove features
    X_reduced = np.delete(X, to_drop, axis=1)

    return X_reduced, to_drop

# Use the function
X_reduced, dropped_features = remove_correlated_features(X, threshold=0.8)

print("Dropped features:", dropped_features)
print("Original shape:", X.shape)
print("Shape after removal:", X_reduced.shape)
```

7.5.3.2 Sử dụng VIF (Variance Inflation Factor)

VIF đo mức độ một feature bị dự đoán bởi các features khác. VIF > 10 thường là dấu hiệu của đa cộng tuyến mạnh (multicollinearity), trong khi VIF > 5 có thể được coi là có vấn đề về correlation.

Công thức VIF

VIF cho feature i được tính bằng:

$$VIF_i = \frac{1}{1 - R_i^2} \tag{25}$$

trong đó R_i^2 là hệ số xác định khi hồi quy feature i với tất cả các features khác.

Triển khai VIF

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
import pandas as pd

def calculate_vif(X, feature_names=None):
    """
    Calculate VIF for all features
```

```
7
8       Parameters:
9       -----------
10      X : array-like
11          Feature matrix
12      feature_names : list, optional
13          T n  c  c  features
14
15      Returns:
16      --------
17      vif_df : DataFrame
18          DataFrame  c h a  VIF scores
19      """
20      if feature_names is None:
21          feature_names = [f"Feature_{i}" for i in range(X.shape[1])]
22
23      vif_df = pd.DataFrame()
24      vif_df["Feature"] = feature_names
25      vif_df["VIF"] = [variance_inflation_factor(X, i)
26                       for i in range(X.shape[1])]
27
28      return vif_df.sort_values('VIF', ascending=False)
29
30  # VIF scores
31  vif_scores = calculate_vif(X, feature_names=['Feature1', 'Feature2', 'Feature3'])
32  print("VIF Scores:")
33  print(vif_scores)
34
35  # Features with VIF cao
36  high_vif_features = vif_scores[vif_scores["VIF"] > 10]
37  print(f"\nFeatures  c   VIF > 10:")
38  print(high_vif_features)
39
40  # Remove iteratively features with high VIF
41  def remove_high_vif_features(X, feature_names, threshold=10):
42      """
43       L o i   b   features  c   VIF cao theo  c  ch iterative
44      """
45      X_temp = X.copy()
46      remaining_features = feature_names.copy()
47      removed_features = []
48
49      while True:
50          vif_df = calculate_vif(X_temp, remaining_features)
51          max_vif = vif_df['VIF'].max()
52
53          if max_vif <= threshold:
54              break
55
56          feature_to_remove = vif_df.loc[vif_df['VIF'].idxmax(), 'Feature']
57          feature_index = remaining_features.index(feature_to_remove)
58
59          X_temp = np.delete(X_temp, feature_index, axis=1)
60          remaining_features.remove(feature_to_remove)
61          removed_features.append(feature_to_remove)
62
63          print(f"Removed {feature_to_remove} (VIF: {max_vif:.2f})")
64
65      return X_temp, remaining_features, removed_features
66
```

```
# Apply VIF-based removal
X_vif_reduced, remaining_features, removed_vif = remove_high_vif_features(
    X, ['Feature1', 'Feature2', 'Feature3'], threshold=10
)
```

7.5.3.3 Giảm chiều (Dimensionality Reduction)

Thay vì loại bỏ features, chúng ta có thể sử dụng các kỹ thuật giảm chiều để kết hợp thông tin từ các features tương quan cao.

Principal Component Analysis (PCA)

PCA kết hợp các features tương quan thành các trục mới không tương quan (principal components).

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

def apply_pca_for_correlation(X, variance_threshold=0.95):
    """
    Apply PCA for correlation

    Parameters:
    -----------
    X : array-like
        Feature matrix
    variance_threshold : float
        Variance want to keep (default: 0.95)

    Returns:
    --------
    X_pca : array
        Data sau PCA transformation
    pca : PCA object
        Fitted PCA object
    """
    # Standardiztion before PCA
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # PCA
    pca = PCA(n_components=variance_threshold)
    X_pca = pca.fit_transform(X_scaled)

    print(f"Original features: {X.shape[1]}")
    print(f"PCA components: {X_pca.shape[1]}")
    print(f"Explained variance ratio: {pca.explained_variance_ratio_.sum():.3f}")

    return X_pca, pca, scaler

# PCA
X_pca, pca_model, scaler = apply_pca_for_correlation(X, variance_threshold=0.95)

# Visualize explained variance
plt.figure(figsize=(10, 6))
plt.subplot(1, 2, 1)
plt.plot(range(1, len(pca_model.explained_variance_ratio_) + 1),
         pca_model.explained_variance_ratio_, 'bo-')
plt.xlabel('Principal Component')
```

```
plt.ylabel('Explained Variance Ratio')
plt.title('Scree Plot')
plt.grid(True)

plt.subplot(1, 2, 2)
cumsum = np.cumsum(pca_model.explained_variance_ratio_)
plt.plot(range(1, len(cumsum) + 1), cumsum, 'ro-')
plt.axhline(y=0.95, color='k', linestyle='--', alpha=0.7)
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Cumulative Explained Variance')
plt.grid(True)

plt.tight_layout()
plt.show()

# Print component weights to view feature contribution
print("\nPCA Component Analysis:")
feature_names = [f"Feature_{i}" for i in range(X.shape[1])]
components_df = pd.DataFrame(
    pca_model.components_[:3].T,  # Ch  ly 3 components  u
    columns=['PC{i+1}' for i in range(3)],
    index=feature_names
)
print(components_df.round(3))
```

Autoencoder

Autoencoders có thể học representations nén dữ liệu, đặc biệt hữu ích cho non-linear correlations.

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam

def create_autoencoder(input_dim, encoding_dim):
    """
     To  autoencoder       gim  chiu  d   liu

    Parameters:
    -----------
    input_dim : int
         S  features ban  u
    encoding_dim : int
         S   chiu sau khi encode

    Returns:
    --------
    autoencoder : Model
        Full autoencoder model
    encoder : Model
        Encoder part only
    """
    # Input layer
    input_layer = Input(shape=(input_dim,))

    # Encoder
    encoded = Dense(64, activation='relu')(input_layer)
    encoded = Dense(32, activation='relu')(encoded)
    encoded = Dense(encoding_dim, activation='relu')(encoded)
```

```
31
32      # Decoder
33      decoded = Dense(32, activation='relu')(encoded)
34      decoded = Dense(64, activation='relu')(decoded)
35      decoded = Dense(input_dim, activation='linear')(decoded)
36
37      # Models
38      autoencoder = Model(input_layer, decoded)
39      encoder = Model(input_layer, encoded)
40
41      # Compile
42      autoencoder.compile(optimizer=Adam(learning_rate=0.001),
43                          loss='mse',
44                          metrics=['mae'])
45
46      return autoencoder, encoder
47
48  # Autoencoder
49  input_dim = X.shape[1]
50  encoding_dim = max(2, input_dim // 3)  #  G i m   x u ng  1/3  s   features
51
52  autoencoder, encoder = create_autoencoder(input_dim, encoding_dim)
53
54  # Standardization
55  scaler_ae = StandardScaler()
56  X_scaled = scaler_ae.fit_transform(X)
57
58  # Train autoencoder
59  history = autoencoder.fit(X_scaled, X_scaled,
60                            epochs=100,
61                            batch_size=32,
62                            validation_split=0.2,
63                            verbose=0)
64
65  # Encode
66  X_encoded = encoder.predict(X_scaled)
67
68  print(f"Original shape: {X.shape}")
69  print(f"Encoded shape: {X_encoded.shape}")
70  print(f"Compression ratio: {X_encoded.shape[1] / X.shape[1]:.2f}")
71
72  # Visualize training history
73  plt.figure(figsize=(12, 4))
74  plt.subplot(1, 2, 1)
75  plt.plot(history.history['loss'], label='Training Loss')
76  plt.plot(history.history['val_loss'], label='Validation Loss')
77  plt.title('Autoencoder Training Loss')
78  plt.xlabel('Epoch')
79  plt.ylabel('Loss')
80  plt.legend()
81  plt.grid(True)
82
83  plt.subplot(1, 2, 2)
84  plt.plot(history.history['mae'], label='Training MAE')
85  plt.plot(history.history['val_mae'], label='Validation MAE')
86  plt.title('Autoencoder Training MAE')
87  plt.xlabel('Epoch')
88  plt.ylabel('MAE')
89  plt.legend()
90  plt.grid(True)
```

```

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

7.5.3.4 So sánh các Phương pháp

Bảng 3: So sánh các phương pháp xử lý Feature Correlation

Phương pháp	**Ưu điểm**	**Nhược điểm**	**Khi nào sử dụng**
Loại bỏ Features	• Đơn giản • Giữ nguyên interpretability • Nhanh	• Mất thông tin • Cần domain knowledge	Khi có features rõ ràng redundant
VIF-based Removal	• Có statistical foundation • Iterative removal • Objective threshold	• Computational expensive • Có thể remove quá nhiều	Khi cần approach có statistical backing
PCA	• Giữ lại most variance • Linear combinations • Orthogonal components	• Mất interpretability • Linear assumptions	Khi correlation pattern linear
Autoencoder	• Handle non-linear • Flexible architecture • Can capture complex patterns	• Complex to implement • Need tuning • Black box	Khi có non-linear correlations