



ĐỒ ÁN MÔN HỌC

PHÂN TÍCH VÀ TRỰC QUAN HOÁ DỮ LIỆU

Ngành: **KHOA HỌC DỮ LIỆU**

Chuyên ngành: **KHOA HỌC DỮ LIỆU**

Giảng viên hướng dẫn : Ths. Lê Nhật Tùng

Sinh viên thực hiện :

2286400009 - Bùi Minh Huy

2286400042 - Trần Lê Vân

2286400028 - Nguyễn Thị Thanh Tâm

Lớp: 22DKHA1

TP. Hồ Chí Minh, 2025

LỜI CAM ĐOAN

Chúng tôi, **Bùi Minh Huy, Trần Lê Vân, Nguyễn Thị Thanh Tâm** xin cam đoan rằng:

Tất cả thông tin và phân tích trình bày trong báo cáo này được thực hiện một cách chính xác và trung thực. Mọi dữ liệu, nhận định hoặc ý kiến được trích dẫn từ các nguồn khác đều đã được nêu rõ nguồn gốc và trích dẫn đúng quy định. Chúng tôi cam đoan rằng không có bất kỳ hành vi sao chép hoặc sử dụng thông tin không hợp pháp nào từ các nguồn khác. Bài báo cáo này là kết quả của công trình nghiên cứu độc lập của chúng tôi và chưa từng được công bố tại bất kỳ nơi nào khác. Chúng tôi cam đoan đã tuân thủ nghiêm ngặt các quy tắc và quy định của môn học, bao gồm việc tham khảo và áp dụng các công cụ nghiên cứu một cách hợp lệ. Nếu phát hiện có bất kỳ sự gian lận nào, chúng tôi xin hoàn toàn chịu trách nhiệm về nội dung bài báo cáo của mình. Chúng tôi hy vọng rằng bài báo cáo này sẽ cung cấp những thông tin hữu ích cho các nhà nghiên cứu, doanh nghiệp, góp phần vào việc hiểu rõ hơn về mạng xã hội ngày nay.

TP. Hồ Chí Minh, ngày 28 tháng 3 năm 2025

Sinh viên

MỤC LỤC

CHƯƠNG 1: GIỚI THIỆU TỔNG QUAN	1
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT	1
2.1 Hồi quy logistic (Logistic Regression)	1
2.1.1 Khái niệm	1
2.1.2 Các loại hồi quy logistic	1
2.1.3 Hàm Sigmoid	2
2.1.4 Cách thức hoạt động của mô hình	2
2.1.5 Nhược điểm của hồi quy logistic	3
2.2 K-Nearest Neighbors (KNN)	3
2.2.1 Khái niệm	3
2.2.2 Cách thức hoạt động của thuật toán KNN	3
2.2.3 Ưu điểm	3
2.2.4 Nhược điểm	3
2.3 Support Vector Machine (SVM)	4
2.3.1 Khái niệm	4
2.3.2 Các loại SVM	4
2.3.3 Cách hoạt động	4
2.3.4 Margin trong SVM	4
2.3.5 Thủ thuật Kernel	4
2.3.6 Ưu điểm	5
2.3.7 Nhược điểm	5
2.4 Decision Tree	5
2.4.1 Khái niệm	5
2.4.2 Cách hoạt động	5
2.4.3 Công thức	5
2.4.4 Ưu điểm	6
2.4.5 Nhược điểm	6
2.4.6 Các vấn đề sau khi thực hiện áp dụng mô hình vào dữ liệu cần dự đoán	7
2.5 Random Forest	7
2.5.1 Khái niệm	7
2.5.2 Cách hoạt động của Random Forest	7
2.5.3 Công thức	7
2.5.4 Ưu điểm	8
2.5.5 Nhược điểm	8

2.6 UMAP	8
2.6.1 Khái niệm	8
2.6.2 Mục tiêu của UMAP	9
2.6.3 Quy trình thực hiện UMAP	9
2.6.4 Ưu điểm	9
2.6.5 Nhược điểm	9
CHƯƠNG 3: GIỚI THIỆU BỘ DỮ LIỆU	9
CHƯƠNG 4: THỰC NGHIỆM	21
4.1 Kiểm tra outlier	21
4.2 Giảm chiều dữ liệu bằng UMAP	22
4.3 Chuẩn bị dữ liệu cho mô hình học máy	25
4.4 Xây dựng hàm đánh giá tổng quát mô hình học máy	26

DANH SÁCH HÌNH

1	Số lượng mẫu dữ liệu theo người dùng và hoạt động	13
2	Phân bố tBodyAccMagmean theo hoạt động	15
3	Phân bố tBodyAccMagmean theo hoạt động sitting, standing, laying	16
4	Phân bố tBodyAccMagmean theo hoạt động walking, walking_dowstairs, walking_upstairs	17
5	Phân bố giữa tBodyAccMagmean và Activity	18
6	Phân bố giữa angleXgravityMean và Activity	19
7	trực quan hóa UMAP	23

CHƯƠNG 1: GIỚI THIỆU TỔNG QUAN

Trong thời đại của điện toán di động và thiết bị thông minh, việc theo dõi và nhận dạng hoạt động con người (Human Activity Recognition - HAR) đã trở thành một lĩnh vực nghiên cứu đóng vai trò quan trọng trong nhiều ngành như trí tuệ nhân tạo, khoa học dữ liệu, y học và công nghệ cảm biến. HAR đóng vai trò cốt lõi trong các ứng dụng như giám sát sức khỏe, phát hiện té ngã, điều khiển nhà thông minh. Ngày nay, nhu cầu càng ngày gia tăng về các thiết bị công nghệ có thể hiểu hành vi con người dẫn đến việc phát triển các mô hình HAR là chính xác, hiệu quả và có khả năng triển khai thực tế là vô cùng cần thiết.

Một trong những yếu tố chính thúc đẩy sự phát triển của HAR là sự phổ biến của các thiết bị di động thông minh và đồng hồ thông minh, vốn được trang bị sẵn các cảm biến quán tính bao gồm gia tốc kế (accelerometer) và con quay hồi chuyển (gyroscope). Những cảm biến này cho phép thu thập dữ liệu về chuyển động của người sử dụng với độ chính xác cao, chi phí thấp và tính khả dụng cao trong đời sống hàng ngày. Nhờ vậy, hệ thống HAR có thể được lắp đặt mà không cần sử dụng các thiết bị đắt tiền hoặc lắp đặt phức tạp.

Bên cạnh tiềm năng ứng dụng rộng rãi, việc xây dựng các mô hình HAR vẫn có nhiều khó khăn thách thức như dữ liệu cảm biến thường có số chiều lớn, có nhiều dữ liệu nhiễu và có tính biến động cao do phụ thuộc vào thói quen và hình thể của mỗi người. Bên cạnh đó, một số hoạt động có thể có mẫu tín hiệu tương tự nhau khiến cho các bài toán phân loại trở nên khó khăn hơn. Vì vậy, cần có một quy trình xử lý dữ liệu bài bản bao gồm các bước tiền xử lý dữ liệu, giảm chiều dữ liệu và huấn luyện mô hình học máy để có thể đạt được hiệu quả cao trong việc nhận dạng hoạt động con người.

Trong nghiên cứu này, chúng em đã tiến hành khai thác bộ dữ liệu “**Human Activity Recognition with Smartphones**” do UCI Machine Learning Repository cung cấp, một bộ dữ liệu được sử dụng rộng rãi trong cộng đồng nghiên cứu HAR. Chúng em đề xuất một quy trình học máy toàn diện bao gồm phân tích đặc trưng, giảm chiều dữ liệu bằng UMAP, PCA, TSNE và huấn luyện bằng các mô hình học máy như Random Forest, Decision Tree, Logistic Regression, Support Vector Machine (SVM) để phân loại các hoạt động với mục tiêu là nâng cao độ chính xác và hiệu quả của mô hình. Những kết quả này sẽ cung cấp cái nhìn thực nghiệm rõ ràng cho các nhà nghiên cứu, đồng thời làm nền móng cho việc triển khai các hệ thống nhận dạng hoạt động trong thế giới thực.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1 Hồi quy logistic (Logistic Regression)

2.1.1 Khái niệm

Hồi quy logistic (Logistic Regression) là một thuật toán học máy có giám sát (supervised learning), đồng thời cũng là một phương pháp thống kê phổ biến, được sử dụng rộng rãi trong việc phân tích và dự đoán dữ liệu phân loại. Mô hình này đặc biệt hiệu quả trong các bài toán phân loại nhị phân, nơi mà biến phụ thuộc chỉ có hai giá trị khả dĩ như có/không, đúng/sai hoặc 1/0.

Hồi quy logistic thường được ưu tiên sử dụng trong các bài toán dự đoán khi biến phụ thuộc không phải là biến liên tục mà là biến nhị phân hoặc phân loại, giúp cung cấp cái nhìn định lượng và chính xác về mối quan hệ giữa các biến độc lập và xác suất xảy ra của sự kiện cần phân tích.

2.1.2 Các loại hồi quy logistic

Hồi quy logistic nhị phân (Binary Logistic Regression). Hồi quy logistic nhị phân dự đoán mối quan hệ giữa các biến phụ thuộc nhị phân và độc lập. Một số ví dụ về đầu ra của loại hồi quy này có thể là thành công/thất bại, 0/1 hoặc đúng/sai.

Hồi quy logistic đa thức : (Multinomial Logistic Regression). Biến phụ thuộc phân loại có hai hoặc nhiều kết quả rời rạc trong loại hồi quy đa thức. Hồi quy logistic đa thức có nhiều hơn hai kết quả có thể xảy ra .

Hồi quy logistic thứ tự (Ordinal Logistic Regression). Hồi quy logistic thứ tự áp dụng khi biến phụ thuộc ở trạng thái có thứ tự (tức là thứ tự).

2.1.3 Hàm Sigmoid

Hồi quy logistic dự đoán xác suất rơi vào một trong hai lớp (binary classification), thường được ký hiệu là 0 hoặc 1.

Để biểu diễn xác suất này, sử dụng hàm sigmoid, có dạng S-shaped và giới hạn giá trị đầu ra trong khoảng từ 0 đến 1.

Công thức của hàm sigmoid:

$$S(z) = \frac{1}{1 + e^{-z}}$$

Trong đó:

- $s(z)$ = đầu ra trong khoảng từ 0 đến 1 (giá trị xác suất ước lượng).
- z = đầu vào của hàm (giá trị dự đoán của thuật toán, ví dụ như $mx+b$).
- e = hằng số Euler, và là cơ sở của logarithm tự nhiên.

Đặc điểm mô hình:

- Phân lớp và dự đoán: Dự đoán biến phụ thuộc nhị phân hoặc danh mục từ một hoặc nhiều biến độc lập.
- Xác định mức độ ảnh hưởng của biến độc lập: Xác định cách thức và mức độ mà các biến độc lập ảnh hưởng đến xác suất của sự kiện hoặc lớp mục tiêu.
- Tính toán xác suất sự kiện: Cung cấp ước lượng xác suất cho một sự kiện xảy ra dựa trên biến độc lập.

2.1.4 Cách thức hoạt động của mô hình

Hồi quy logistic sử dụng hàm logistic (còn gọi là hàm sigmoid) để chuyển đổi giá trị dự đoán thành xác suất. Hàm logistic có dạng:

Công thức của mô hình hồi quy logistic:

$$P(Y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \dots + \beta_k X_k)}}$$

Trong đó:

$P(Y = 1)$ là xác suất để sự kiện $Y = 1$ xảy ra (ví dụ: sự kiện thành công, top 1,...). X_1, X_2, \dots, X_k là các biến độc lập. $\beta_0, \beta_1, \dots, \beta_k$ là hệ số mô hình cần được ước lượng. e là cơ số của logarithm tự nhiên.

Cách hoạt động

- **Ước lượng hệ số mô hình:** Hệ số β của mô hình được ước lượng thông qua quy trình tối ưu hóa, thường là phương pháp Maximum Likelihood Estimation (MLE). MLE tìm cách tối đa hóa xác suất của dữ liệu quan sát dựa trên hệ số β .
- **Phân lớp:** Dựa vào xác suất được dự đoán từ hàm logistic, quyết định phân loại một quan sát vào lớp 1 nếu $P(Y = 1) \geq$ một ngưỡng cụ thể (thường là 0.5) và ngược lại là lớp 0. Ví dụ: Nếu $P(Y = 1) > 0.5$, quan sát được phân loại là lớp 1.
- **Đánh giá mô hình:** Mô hình hồi quy logistic thường được đánh giá thông qua các chỉ số như độ chính xác (accuracy), precision, recall, điểm số F1, hoặc thông qua ROC và AUC.

2.1.5 Nhược điểm của hồi quy logistic

- Không thích hợp với biến phụ thuộc liên tục.
- khó khăn trong việc mô hình hóa mối quan hệ phức tạp hoặc không tuyến tính mà không cần biến đổi dữ liệu.
- Không hiệu quả khi xử lý dữ liệu có nhiều biến độc lập hoặc có sự tương quan cao giữa các biến.

2.2 K-Nearest Neighbors (KNN)

2.2.1 Khái niệm

K-Nearest Neighbors (KNN) là một trong những thuật toán học máy đơn giản nhất nhưng hiệu quả, thuộc nhóm học có giám sát (supervised learning). Thuật toán được sử dụng cho cả hai bài toán phân lớp (classification) và hồi quy (regression), tuy nhiên nó phổ biến hơn trong các bài toán phân lớp. Ý tưởng của thuật toán này là nó không học một điều gì từ tập dữ liệu học (nên KNN được xếp vào loại lazy learning), mọi tính toán được thực hiện khi nó cần dự đoán nhãn của dữ liệu mới.

2.2.2 Cách thức hoạt động của thuật toán KNN

Bước 1: Chọn một số nguyên K (số lượng hàng xóm gần nhất cần xét).

Bước 2: Tính khoảng cách của data input với các data trong có trong tập data train, có các cách tính khoảng cách như: Minkowski, Euclid, Manhattan, ... tùy mục đích sử dụng mà chúng ta sử dụng cách tính khoảng cách, thông dụng nhất là cách tính Euclid.

Bước 3: Sau khi tính khoảng cách từ data input tới toàn bộ data trong tập training, chọn ra K lân cận với khoảng cách ngắn nhất, với K được chọn ở bước số 1.

Bước 4: Thực hiện phân loại, kết quả sẽ theo label có tỉ lệ voting cao nhất.

Lựa chọn giá trị K

- Giá trị K quá nhỏ (ví dụ K = 1): mô hình nhạy cảm với nhiễu (noise), dễ bị overfitting.
- Giá trị K quá lớn: mô hình quá “mềm”, dễ bị underfitting

2.2.3 Ưu điểm

- Đơn giản, dễ hiểu: Thuật toán có nguyên lý hoạt động trực quan, dễ cài đặt.
- Không cần huấn luyện: Không tồn thời gian xây dựng mô hình.
- Hiệu quả với dữ liệu nhỏ: Hoạt động tốt với tập dữ liệu kích thước vừa phải.
- Ứng dụng đa dạng: Có thể áp dụng cho cả bài toán phân lớp và hồi quy.
- Không có giả định về dữ liệu: Là mô hình phi tham số, không yêu cầu dữ liệu tuân theo phân phối cụ thể.
- Thích ứng tốt với dữ liệu mới: Dễ dàng cập nhật mô hình bằng cách thêm mẫu mới.

2.2.4 Nhược điểm

- Chi phí dự đoán cao: Phải tính khoảng cách đến mọi mẫu dữ liệu, tốn kém với dữ liệu lớn.
- Nhạy cảm với quy mô dữ liệu: Các đặc trưng có phạm vi lớn sẽ áp đảo các đặc trưng có phạm vi nhỏ.
- Nhạy cảm với nhiễu và dữ liệu thừa: Kém hiệu quả khi có nhiều đặc trưng không liên quan.
- Vấn đề với dữ liệu không cân bằng: Các lớp thịnh hành có xu hướng áp đảo các lớp thiểu số.
- Khó chọn k tối ưu: Giá trị k thích hợp phụ thuộc vào dữ liệu cụ thể.

2.3 Support Vector Machine (SVM)

2.3.1 Khái niệm

SVM là một thuật toán học máy có giám sát (supervised learning), dùng để giải quyết các bài toán phân loại và hồi quy.

Mục tiêu chính của SVM là tìm một siêu phẳng (hyperplane) tốt nhất để phân tách các điểm dữ liệu thuộc hai lớp khác nhau sao cho khoảng cách (margin) giữa siêu phẳng và các điểm gần nhất của mỗi lớp là lớn nhất.

Điểm dữ liệu gần nhất đó được gọi là vector hỗ trợ (support vector). SVM sử dụng các hàm kernel để biểu diễn không gian dữ liệu ban đầu vào không gian cao chiều hơn, giúp phân loại tốt hơn đối với các bài toán phức tạp.

2.3.2 Các loại SVM

Trong thuật toán SVM, có hai loại chính là Linear SVM (SVM tuyến tính) và Non-linear SVM (SVM phi tuyến).

Linear SVM là loại SVM mà ta có thể phân chia dữ liệu bằng một đường thẳng. Điều này áp dụng cho các bài toán có dữ liệu tuyến tính và có thể tách biệt bằng một đường thẳng.

Non-linear SVM được sử dụng khi không thể phân chia dữ liệu bằng một đường thẳng. Trong trường hợp này, ta sẽ sử dụng các phương pháp biến đổi dữ liệu sao cho chúng trở thành tuyến tính, sau đó áp dụng Linear SVM để giải quyết bài toán.

2.3.3 Cách hoạt động

- Chọn siêu phẳng phân tách: SVM tìm kiếm siêu phẳng sao cho có khoảng cách margin lớn nhất giữa hai lớp dữ liệu.
- Tối đa hóa margin: Có gắng duy trì khoảng cách lớn nhất từ siêu phẳng tới các điểm dữ liệu gần nhất.
- Áp dụng kernel nếu cần: Nếu dữ liệu không phân tách tuyến tính, áp dụng kernel trick để biến đổi không gian dữ liệu.
- Đánh giá và điều chỉnh tham số để tối ưu mô hình.

2.3.4 Margin trong SVM

Margin là khoảng cách từ siêu phẳng phân tách (hyperplane) đến các điểm dữ liệu gần nhất thuộc hai lớp khác nhau – các điểm này được gọi là vector hỗ trợ (support vectors). Trong hình dung đơn giản, ví dụ như bài toán phân loại quả táo và quả lê đặt trên mặt bàn, margin chính là khoảng cách từ cây que (đại diện cho siêu phẳng) đến quả táo và quả lê gần cây que nhất. Điều quan trọng trong SVM là thuật toán luôn tìm cách tối đa hóa margin này, nhằm tạo ra một siêu phẳng phân tách có khoảng cách lớn nhất đến các điểm dữ liệu gần ranh giới nhất của mỗi lớp. Nhờ đó, SVM có thể tăng cường khả năng tổng quát hóa và giảm thiểu nguy cơ phân loại sai khi xử lý các điểm dữ liệu mới chưa từng thấy trước đó.

2.3.5 Thủ thuật Kernel

Kernel là một hàm ánh xạ dữ liệu từ không gian ít nhiều sang không gian nhiều chiều hơn, từ đó ta tìm được siêu phẳng phân tách dữ liệu. Một cách trực quan, kỹ thuật này giống như việc bạn gấp tờ giấy lại để có thể dùng kéo cắt một lỗ tròn trên nó.

2.3.6 Ưu điểm

- Xử lý trên không gian số chiều cao: SVM là một công cụ tính toán hiệu quả trong không gian chiều cao, trong đó đặc biệt áp dụng cho các bài toán phân loại văn bản và phân tích quan điểm nơi chiều có thể cực kỳ lớn.
- Tiết kiệm bộ nhớ: Do chỉ có một tập hợp con của các điểm được sử dụng trong quá trình huấn luyện và ra quyết định thực tế cho các điểm dữ liệu mới.
- Tính linh hoạt - phân lớp thường là phi tuyến tính. Khả năng áp dụng Kernel mới phép linh động giữa các phương pháp tuyến tính và phi tuyến tính từ đó khiên cho hiệu suất phân loại lớn hơn.

2.3.7 Nhược điểm

- Thuật toán SVM có độ phức tạp tính toán cao khi số lượng dữ liệu lớn.
- SVM yêu cầu dữ liệu huấn luyện là tuyến tính hoặc phi tuyến tính.
- Thuật toán SVM cần lựa chọn tham số tốt để đạt được kết quả tốt nhất.

2.4 Decision Tree

2.4.1 Khái niệm

Cây quyết định (Decision Tree) là một trong những thuật toán phổ biến nhất trong lĩnh vực máy học (Machine Learning). Là một công cụ mạnh mẽ, thường được sử dụng để giải quyết các bài toán về phân loại (classification) và dự đoán (regression) trong khai phá dữ liệu. Cây quyết định là một thuật toán máy học dùng để dự đoán hoặc phân loại dữ liệu dựa trên các bước ra quyết định nối tiếp nhau.

2.4.2 Cách hoạt động

Cách hoạt động của cây quyết định rất đơn giản:

- Đầu tiên, cây quyết định sẽ xem xét toàn bộ tập dữ liệu.
- Sau đó, chia dữ liệu thành các nhóm nhỏ dựa trên một số đặc điểm hoặc điều kiện nhất định (ví dụ như tuổi tác, thu nhập, màu sắc, nhiệt độ..).
- Việc chia nhỏ này sẽ tiếp tục thực hiện đến khi dữ liệu ở mỗi nhóm trở nên rõ ràng và dễ dàng để dự đoán hoặc phân loại.

Trong cây quyết định, mỗi “nút” (node) đại diện cho một đặc điểm (ví dụ: “tuổi”, “mức thu nhập”, “thời tiết”). Các “nhánh” (branch) nối giữa các nút chính là các điều kiện để chia dữ liệu (ví dụ: tuổi lớn hơn hay nhỏ hơn 30). Cuối cùng, các “nút lá” (leaf node) là kết quả cuối cùng mà cây quyết định đưa ra, ví dụ như “mua hàng” hay “không mua hàng”, “đi chơi thể thao” hay “ở nhà”.

Nói cách khác, cây quyết định giống như một chuỗi các câu hỏi đơn giản được sắp xếp theo từng bước, giúp ta dễ dàng đi đến một quyết định cuối cùng.

2.4.3 Công thức

Gini Impurity

Công thức Gini Impurity được sử dụng trong thuật toán cây quyết định để đo lường độ không chính xác của một dự đoán khi phân loại một tập dữ liệu.

Cần lưu ý là Gini Impurity càng nhỏ (gần 0) thì tập dữ liệu đó càng “thuần khiết”, nghĩa là các mẫu trong cùng một nhóm có xu hướng thuộc vào cùng một lớp. Ngược lại, nếu Gini Impurity cao (gần 1), thì việc phân loại các mẫu trong nhóm đó trở nên không chắc chắn.

Giả sử khi đang xem xét một tập dữ liệu chia thành K nhóm, mỗi nhóm chứa một phần tỷ lệ p_i với $i=1,2,\dots,K$.

Công thức tính độ bất thuần Gini (Gini Impurity):

$$I_G = 1 - \sum_{i=1}^K p_i^2$$

Trong đó:

- I_G là Gini Impurity.
- p_i là tỷ lệ các mẫu thuộc vào lớp i .

Khi xây dựng cây quyết định, chúng ta cần chọn thuộc tính và giá trị phân chia sao cho Gini Impurity sau phân chia là nhỏ nhất, tức là mức độ “thuần khiết” cho dữ liệu thuộc về từng nhóm con được tạo ra.

Entropy

Entropy trong cây quyết định là một khái niệm được sử dụng để đo lường sự không chắc chắn trong dữ liệu (Trung bình surprise). Trong ngữ cảnh của cây quyết định, entropy thường được sử dụng để đo lường mức độ không chắc chắn của phân phối lớp trong tập dữ liệu.

Entropy được tính bằng công thức sau:

$$\text{Entropy}(S) = - \sum_{i=1}^c p_i \log_2(p_i)$$

Trong đó:

- S là tập dữ liệu.
- c là số lớp trong tập dữ liệu.
- p_i là tỷ lệ của lớp i trong tập dữ liệu.

Entropy càng cao khi tỷ lệ của các lớp trong tập dữ liệu gần bằng nhau, và càng thấp khi một lớp chiếm đa số.

Khi xây dựng cây quyết định, chúng ta cố gắng chia tập dữ liệu sao cho entropy sau khi chia là thấp nhất có thể. Điều này giúp cây quyết định có thể học được các quy tắc quyết định hiệu quả từ dữ liệu.

Quyết định về cách chia tập dữ liệu dựa trên entropy thường được thực hiện bằng cách so sánh entropy trước và sau khi chia, và chọn cách chia mà giảm entropy nhiều nhất.

2.4.4 Ưu điểm

- Mô hình sinh ra các quy tắc dễ hiểu cho người đọc, tạo ra bộ luật với mỗi nhánh lá là một luật của cây.
- Dữ liệu đầu vào có thể là dữ liệu missing, không cần chuẩn hóa hoặc tạo biến giả.
- Có thể làm việc với cả dữ liệu số và dữ liệu phân loại.
- Có thể xác thực mô hình bằng cách sử dụng các kiểm tra thống kê.
- Có khả năng làm với dữ liệu lớn.

2.4.5 Nhược điểm

- Mô hình cây quyết định phụ thuộc rất lớn vào dữ liệu. Thậm chí, với một sự thay đổi nhỏ trong bộ dữ liệu, cấu trúc mô hình cây quyết định có thể thay đổi hoàn toàn.
- Cây quyết định hay gặp vấn đề overfitting

2.4.6 Các vấn đề sau khi thực hiện áp dụng mô hình vào dữ liệu cần dự đoán

Underfitting: là hiện tượng kết quả độ chênh lệch của mô hình được huấn luyện và kết quả độ chênh lệch của dữ liệu cần dự đoán đạt giá trị mức cao giống nhau, do mô hình chưa được huấn luyện đầy đủ. Cần xem lại cấu trúc của mô hình (tăng thêm độ phức tạp) để có thể huấn luyện các tập dữ liệu khó và tăng thêm dữ liệu huấn luyện để tăng hiệu suất của mô hình.

Overfitting: là hiện tượng kết quả của mô hình được huấn luyện quá tốt (độ chênh lệch thấp) nhưng khi áp dụng vào dữ liệu cần dự đoán thì mô hình đạt hiệu suất kém (độ chênh lệch cao) do mô hình đã học quá sát với dữ liệu huấn luyện và không có khả năng tổng quát hóa các dữ liệu cần dự đoán. Cần sử dụng một số các phương pháp tránh overfitting như tăng độ đa dạng của dữ liệu, giảm thiểu độ phức tạp của mô hình.

=> Giải pháp : Pruning solution là được áp dụng đối với trường hợp mô hình huấn luyện bị overfitting khi sử dụng mô hình Decision Tree bằng cách hạn chế kích thước, chiều sâu của mô hình này.

2.5 Random Forest

2.5.1 Khái niệm

Random Forest là một thuật toán học máy thuộc nhóm học có giám sát (supervised learning), được sử dụng phổ biến trong cả bài toán phân loại và hồi quy.

Khác với Decision Tree chỉ dựa vào một cây duy nhất, Random Forest kết hợp nhiều cây quyết định để tạo ra một mô hình tổng hợp, có khả năng dự đoán chính xác và ổn định hơn.

2.5.2 Cách hoạt động của Random Forest

- Tạo ra nhiều cây khác nhau từ các tập dữ liệu nhỏ, được chọn ngẫu nhiên từ tập dữ liệu gốc.
- Mỗi cây sẽ đưa ra dự đoán riêng.
- Mô hình sẽ lấy trung bình (với hồi quy) hoặc bỏ phiếu theo số đông (với phân loại) để đưa ra kết quả cuối cùng.

2.5.3 Công thức

Bài toán phân loại (classification) Bài toán phân loại, mỗi cây quyết định trong rừng sẽ đưa ra một nhãn dự đoán. Sau đó, mô hình Random Forest sẽ lấy nhãn xuất hiện nhiều nhất trong các dự đoán của từng cây làm kết quả cuối cùng.

Công thức được biểu diễn như sau:

$$\hat{y} = \text{mode}(h_1(x), h_2(x), \dots, h_T(x))$$

Trong đó:

- \hat{y} là kết quả dự đoán cuối cùng của mô hình.
- $h_t(x)$ là kết quả dự đoán của cây thứ t với đầu vào x .
- T là tổng số cây trong mô hình Random Forest.
- mode là hàm chọn giá trị xuất hiện nhiều nhất.

Bài toán hồi quy (Regression) Với bài toán hồi quy, mỗi cây trong rừng sẽ đưa ra một giá trị số. Kết quả cuối cùng được tính bằng cách lấy trung bình cộng các giá trị dự đoán của tất cả các cây.

Được tính bằng công thức sau:

$$\hat{y} = \frac{1}{T} \sum_{t=1}^T h_t(x)$$

Trong đó:

- \hat{y} là giá trị dự đoán cuối cùng.
- $h_t(x)$ là giá trị dự đoán từ cây thứ t .
- T là tổng số cây trong mô hình.

2.5.4 Ưu điểm

- Độ chính xác cao: mô hình tổng hợp nhiều cây, Random Forest thường cho kết quả chính xác hơn so với cây quyết định đơn lẻ.
- Chống overfitting tốt: kết hợp nhiều cây được huấn luyện từ dữ liệu và thuộc tính ngẫu nhiên giúp mô hình tránh học quá sát vào dữ liệu huấn luyện.
- Làm việc tốt với dữ liệu lớn và có nhiều đặc trưng: Random Forest xử lý hiệu quả dữ liệu có số chiều lớn và phức tạp.
- Không yêu cầu chuẩn hóa dữ liệu: Dữ liệu đầu vào không cần phải được chuẩn hóa hoặc xử lý đặc biệt như một số thuật toán khác.
- Có thể đo lường mức độ quan trọng của các thuộc tính (feature importance): Giúp phân tích và chọn ra các yếu tố ảnh hưởng nhiều nhất đến kết quả dự đoán.

2.5.5 Nhược điểm

- Khó giải thích mô hình: Random Forest gồm nhiều cây nên khó để xem toàn bộ quá trình mô hình đưa ra kết quả, khó để giải thích được mô hình.
- Thời gian huấn luyện và dự đoán lâu hơn: Mô hình gồm nhiều cây nên tốn nhiều thời gian và tài nguyên hơn khi xử lý dữ liệu lớn.
- Chiếm nhiều bộ nhớ: Lưu trữ nhiều cây có thể tốn nhiều RAM, đặc biệt khi số lượng cây lớn.
- Dễ bị bias nếu dữ liệu không cân bằng: Trong trường hợp dữ liệu bị lệch, Random Forest có thể dự đoán thiên lệch theo lớp đó nếu không xử lý cân bằng dữ liệu trước.

2.6 UMAP

2.6.1 Khái niệm

Uniform Manifold Approximation and Projection (UMAP) là một kỹ thuật giảm chiều dữ liệu, tương tự như t-SNE, thường được sử dụng để trực quan hóa dữ liệu. Ngoài ra, UMAP còn có thể được sử dụng như một phương pháp giảm chiều phi tuyến tổng quát trong các bài toán học máy. Thuật toán UMAP được xây dựng dựa trên ba giả định chính về cấu trúc dữ liệu: 1. Dữ liệu phân bố đều trên một đa tạp Riemannian (Riemannian manifold); 2. Metric Riemannian là hằng số cục bộ (hoặc có thể được xấp xỉ là như vậy); 3. Ông phân phôi (local connectivity) được kết nối cục bộ.

2.6.2 Mục tiêu của UMAP

- Giảm số lượng biến: UMAP chuyển đổi dữ liệu sang một không gian có số chiều thấp hơn, phù hợp cho các tác vụ xử lý và phân tích tiếp theo.
- Giữ lại cấu trúc dữ liệu: UMAP cố gắng bảo toàn cả cấu trúc cục bộ (local structure) lẫn cấu trúc tổng thể (global structure) của dữ liệu trong không gian mới.
- Bảo tồn mối quan hệ phi tuyến: Khác với PCA, UMAP có khả năng nắm bắt và biểu diễn các mối quan hệ phi tuyến giữa các điểm dữ liệu.
- Trực quan hóa dữ liệu: UMAP đặc biệt hiệu quả trong việc trực quan hóa dữ liệu nhiều chiều trong không gian 2D hoặc 3D, với độ chính xác và sắc nét cao hơn so với nhiều kỹ thuật khác như t-SNE.

2.6.3 Quy trình thực hiện UMAP

- Chuẩn hóa dữ liệu (nếu cần).
- Tìm k hàng xóm gần nhất cho mỗi điểm và xây dựng đồ thị fuzzy biểu diễn cấu trúc cục bộ.
- Tính toán xác suất kết nối giữa các điểm dựa trên khoảng cách và hàm kernel.
- Khai tạo các điểm trong không gian thấp chiều và tối ưu hóa đồ thị sao cho bảo toàn cấu trúc so với đồ thị ban đầu.
- Chiếu dữ liệu sang không gian mới để phục vụ trực quan hóa hoặc các bước phân tích tiếp theo.

2.6.4 Ưu điểm

- Bảo toàn cấu trúc cục bộ tốt.
- Có thể dùng cho cả supervised & unsupervised.
- UMAP cho phép lưu và áp dụng mô hình lên dữ liệu mới.

2.6.5 Nhược điểm

- Không giải thích được biến gốc.
- Phụ thuộc vào tham số.

CHƯƠNG 3: GIỚI THIỆU BỘ DỮ LIỆU

Bộ dữ liệu **Human Activity Recognition with Smartphones** được xây dựng nhằm phục vụ cho các nghiên cứu về nhận diện hành vi con người thông qua dữ liệu cảm biến thu thập từ thiết bị di động. Tập dữ liệu được thu thập từ 30 người tham gia (goi là *subjects*) (15 nam và 15 nữ, độ tuổi từ 19 đến 48) thực hiện sáu hoạt động thường ngày như Đi bộ (Walking), đi lên cầu thang (Walking Upstairs), đi xuống cầu thang (WalkingDownstairs), ngồi (Sitting), đứng (Standing)và nằm (Laying).

Dữ liệu được ghi lại bằng một điện thoại thông minh (Samsung Galaxy S II) đeo ở thắt lưng của người dùng. Thiết bị này sử dụng hai loại cảm biến là **accelerometer** và **gyroscope** được sử dụng để ghi lại chuyển động theo 3 trục X, Y, Z với tần số lấy mẫu 50Hz.

Mỗi chuỗi tín hiệu được chia thành các **cửa sổ trượt** có độ dài 2.56 giây, tương ứng với 128 lần đo. Từ mỗi cửa sổ, các đặc trưng (features) đã được trích xuất từ **miền thời gian** và **miền tần số** để tạo ra một tập hợp dữ liệu có cấu trúc sẵn sàng cho mô hình học máy.

Các tín hiệu chính được sử dụng để tạo đặc trưng bao gồm:

- tBodyAcc-XYZ: Gia tốc cơ thể theo 3 trục trong miền thời gian
- tGravityAcc-XYZ: Gia tốc do trọng lực
- tBodyGyro-XYZ: Tốc độ quay từ con quay hồi chuyển
- tBodyAccJerk-XYZ, tBodyGyroJerk-XYZ: Jerk - đo sự thay đổi đột ngột của chuyển động
- Mag: Độ lớn vector gia tốc, được tính bằng chuẩn Euclidean:

$$\text{Mag} = \sqrt{X^2 + Y^2 + Z^2}$$

- fBodyAcc-XYZ, fBodyGyro-XYZ: Biến miền tần số được tạo từ FFT

Từ các tín hiệu trên, 561 đặc trưng thống kê đã được trích xuất, bao gồm:

- mean(), std(), mad(), max(), min(): Đặc trưng thống kê truyền thông
- sma(): Signal Magnitude Area
- energy(): Tổng bình phương chia số phần tử
- entropy(), iqr(), arCoeff(), correlation()
- meanFreq(), skewness(), kurtosis(), bandsEnergy(), angle()

Toàn bộ dữ liệu được chia thành hai phần: Tập huấn luyện (train.csv): bao gồm 7352 mẫu. Tập kiểm tra (test.csv): bao gồm 2947 mẫu. Mỗi mẫu tương ứng với một cửa sổ thời gian 2.56 giây, được biểu diễn bằng **561 đặc trưng đầu vào (features)**, cùng với một mã định danh người thực hiện (subject) và một nhãn hoạt động (Activity), các nhãn này được mã hóa từ 1 đến 6 tương ứng với:

Giá trị nhãn	Hoạt động
1	WALKING
2	WALKING_UPSTAIRS
3	WALKING_DOWNSTAIRS
4	SITTING
5	STANDING
6	LAYING

```
# install.packages("showtext")
# install.packages("ggplot2")

# train <- read.csv('/Users/huy/Documents/doanthaytung/archive/train.csv')
# train <- read.csv("D:/BT/clonegit/doanthaytung/archive/train.csv")
train <- read.csv('archive/train.csv')
# head(train)
dim(train)
```

```
## [1] 7352 563
```

```
# test <- read.csv('/Users/huy/Documents/doanthaytung/archive/test.csv')
# test <- read.csv("D:/BT/clonegit/doanthaytung/archive/test.csv")
test <- read.csv("archive/test.csv")
# head(test)
dim(test)
```

```
## [1] 2947 563
```

- Kiểm tra kiểu dữ liệu của các biến trong tập huấn luyện để thống kê số lượng cột thuộc từng kiểu dữ liệu trong bảng dữ liệu train. Kết quả sẽ cung cấp một cái nhìn tổng quan về cấu trúc dữ liệu, giúp đánh giá mức độ sẵn sàng của tập dữ liệu cho các bước xử lý tiếp theo.

```
table(sapply(train, class))
```

```
##  
## character   integer   numeric  
##          1         1        561
```

Theo kết quả trả về, bộ dữ liệu train bao gồm: 561 biến kiểu numeric, 1 biến kiểu character, 1 biến kiểu integer.

- Kiểm tra xem có biến nào được lưu dưới dạng chuỗi ký tự (character) hay không. Đảm bảo các biến phân loại sẽ được xử lý đúng cách trong mô hình học máy.

```
names(train)[sapply(train, class) == "character"]
```

```
## [1] "Activity"
```

Kết quả cho thấy chỉ có một biến duy nhất là “Activity” đang được lưu dưới dạng character. Cho thấy nhãn phân loại hiện tại vẫn chưa được chuyển đổi về dạng factor, cần được xử lý lại để đảm bảo phù hợp với các thuật toán phân loại.

- Kiểm tra danh sách đầy đủ các giá trị duy nhất (tức là các lớp phân loại) mà biến này chứa.

```
unique(train$Activity)
```

```
## [1] "STANDING"           "SITTING"            "LAYING"  
## [4] "WALKING"             "WALKING_DOWNSTAIRS" "WALKING_UPSTAIRS"
```

- Tiến hành chuyển đổi kiểu dữ liệu này sang kiểu factor.

```
train$Activity <- as.factor(train$Activity)  
test$Activity <- as.factor(test$Activity)
```

- Để đảm bảo chất lượng dữ liệu đầu vào cho mô hình học máy, chúng em thực hiện kiểm tra các giá trị bị thiếu (missing values) trong cả hai tập dữ liệu huấn luyện và kiểm tra.

```
cat("Giá trị thiếu ở tập train:", sum(is.na(train)), "\n")
```

```
## Giá trị thiếu ở tập train: 0
```

```
cat("Giá trị thiếu ở tập test:", sum(is.na(test)), "\n")
```

```
## Giá trị thiếu ở tập test: 0
```

Kết quả cho thấy không có giá trị thiêu trong cả tập huấn luyện (train) và tập kiểm tra (test), với tổng số lượng NA bằng 0.

- Bên cạnh việc kiểm tra giá trị thiêu, chúng em đánh giá tính duy nhất của các dữ liệu bằng cách xác định xem có các dòng dữ liệu nào bị trùng lặp trong cả hai tập huấn luyện và kiểm tra hay không.

```
cat("Số dòng bị trùng lặp trong tập train:", sum(duplicated(train)), "\n")
```

```
## Số dòng bị trùng lặp trong tập train: 0
```

```
cat("Số dòng bị trùng lặp trong tập test :", sum(duplicated(test)), "\n")
```

```
## Số dòng bị trùng lặp trong tập test : 0
```

Kết quả hiển thị không có dòng nào bị trùng lặp trong cả hai tập dữ liệu (train và test), với tổng số dòng trùng là 0.

```
columns <- colnames(train)
columns <- gsub("\\.", "", columns)
colnames(train) <- columns
colnames(test) <- columns
```

- Tiến hành xem phân phối dữ liệu qua các biểu sau:

```
library(ggplot2)
library(showtext)
```

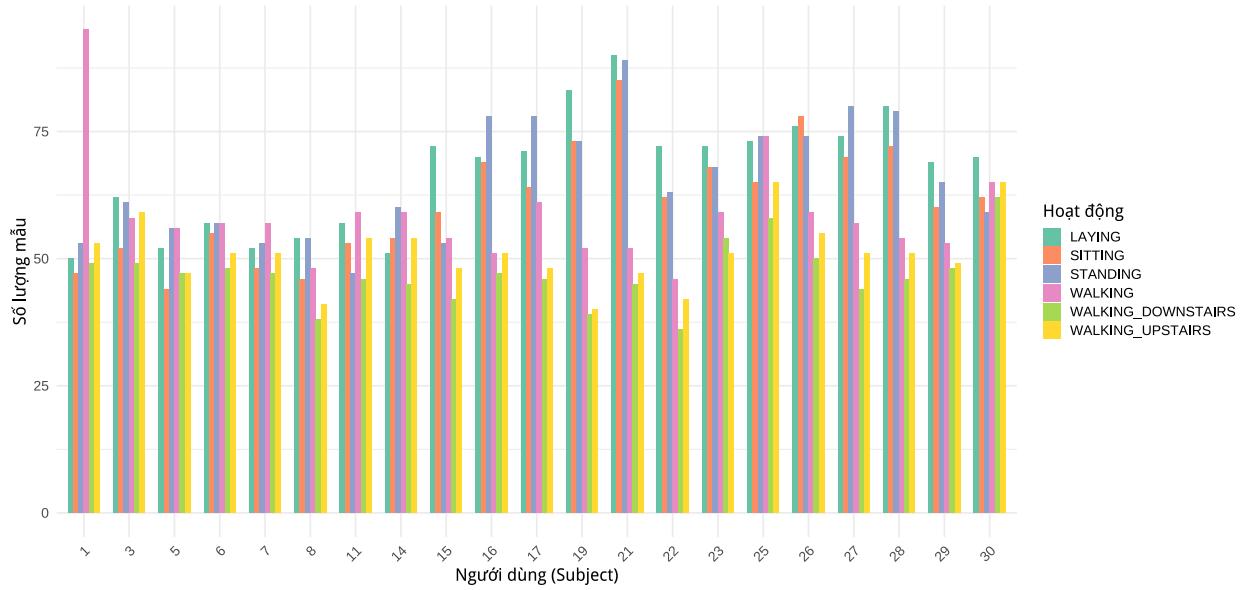
```
## Loading required package: sysfonts
```

```
## Loading required package: showtextdb
```

```
showtext_auto()

ggplot(train, aes(x = factor(subject), fill = Activity)) +
  geom_bar(position = "dodge", width = 0.7) +
  scale_fill_brewer(palette = "Set2") +
  labs(
    title = "Số lượng mẫu dữ liệu theo người dùng và hoạt động",
    x = "Người dùng (Subject)",
    y = "Số lượng mẫu",
    fill = "Hoạt động"
  ) +
  theme_minimal(base_size = 16) +
  theme(
    plot.title = element_text(hjust = 0.5, face = "bold", size = 20),
    axis.text.x = element_text(angle = 45, hjust = 1),
    legend.position = "right"
  )
```

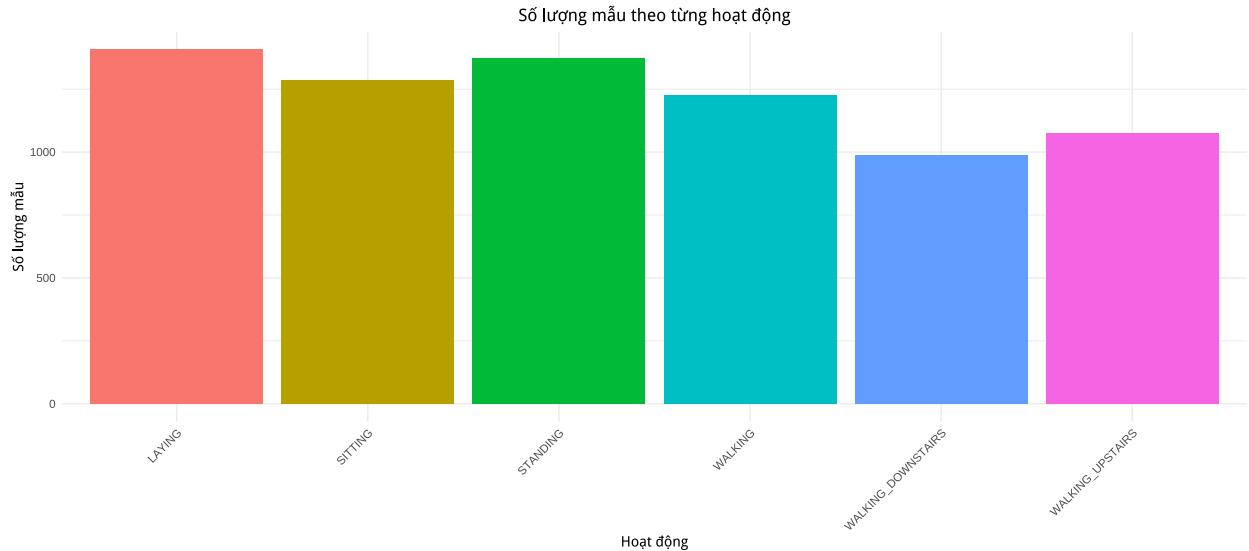
Số lượng mẫu dữ liệu theo người dùng và hoạt động



Hình 1: Số lượng mẫu dữ liệu theo người dùng và hoạt động

```
library(ggplot2)

ggplot(train, aes(x = Activity, fill = Activity)) +
  geom_bar() +
  labs(
    title = "Số lượng mẫu theo từng hoạt động",
    x = "Hoạt động",
    y = "Số lượng mẫu"
  ) +
  theme_minimal(base_size = 15) +
  theme(
    axis.text.x = element_text(angle = 45, vjust = 1, hjust=1),
    plot.title = element_text(hjust = 0.5, face = "bold")
  ) +
  guides(fill = "none")
```



Số lượng mẫu cho mỗi hoạt động dao động trong khoảng 1000 mẫu đến 1200 mẫu. Các hoạt động tĩnh như laying, sitting, standing có xu hướng chiếm tỷ lệ cao hơn một chút so với các hoạt động di chuyển như walking, walking_upstairs, walking_downstairs. Mức độ chênh lệch không qua lớn giữa các nhóm, là điểm thuận lợi cho các mô hình học máy tránh bị lệch nhẫn và đảm bảo khả năng học đều giữa các lớp.

```
library(ggplot2)

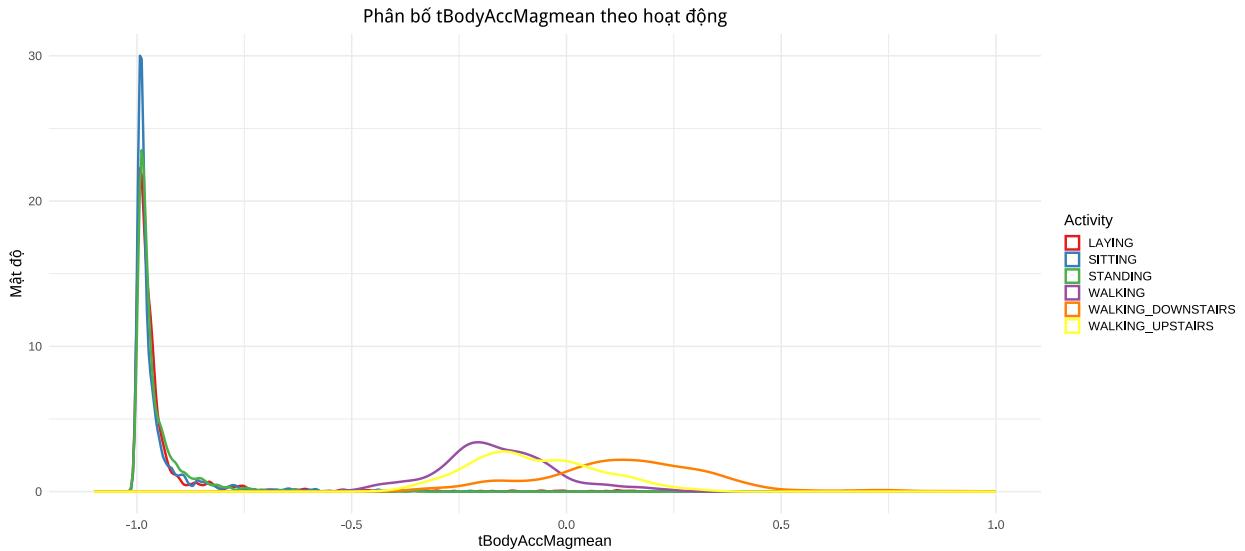
ggplot(train, aes(x = tBodyAccMagmean, color = Activity)) +
  geom_density(size = 1.2) +
  scale_color_brewer(palette = "Set1") +
  scale_x_continuous(limits = c(-1.1, 1)) +
  labs(
    title = "Phân bố tBodyAccMagmean theo hoạt động",
    x = "tBodyAccMagmean",
    y = "Mật độ"
  ) +
  theme_minimal(base_size = 16) +
  theme(
    plot.title = element_text(hjust = 0.5)
  )

## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
library(ggplot2)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
## 
##     filter, lag
```



Hình 2: Phân bố tBodyAccMagmean theo hoạt động

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(gridExtra)

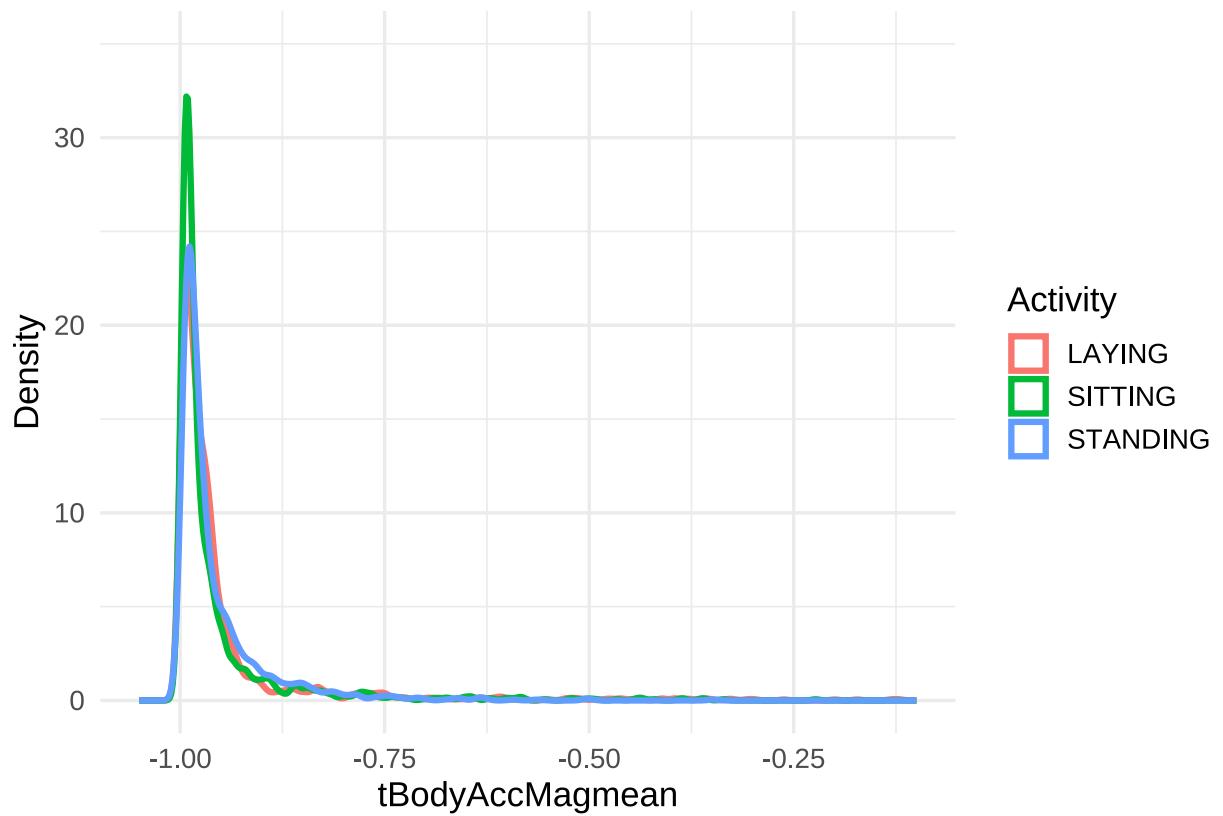
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine

p1 <- train %>%
  filter(Activity %in% c("SITTING", "STANDING", "LAYING")) %>%
  ggplot(aes(x = tBodyAccMagmean, color = Activity)) +
  geom_density(size = 1.2) +
  labs(
    title = "Static Activities (closer view)",
    x = "tBodyAccMagmean",
    y = "Density"
  ) +
  xlim(-1.05, -0.1) +
  ylim(0, 35) +
  theme_minimal(base_size = 14)
p1

p2 <- train %>%
  filter(Activity %in% c("WALKING", "WALKING_DOWNSTAIRS", "WALKING_UPSTAIRS")) %>%
  ggplot(aes(x = tBodyAccMagmean, color = Activity)) +
  geom_density(size = 1.2) +
```

Static Activities (closer view)



Hình 3: Phân bố $t\text{BodyAccMagmean}$ theo hoạt động sitting, standing, laying

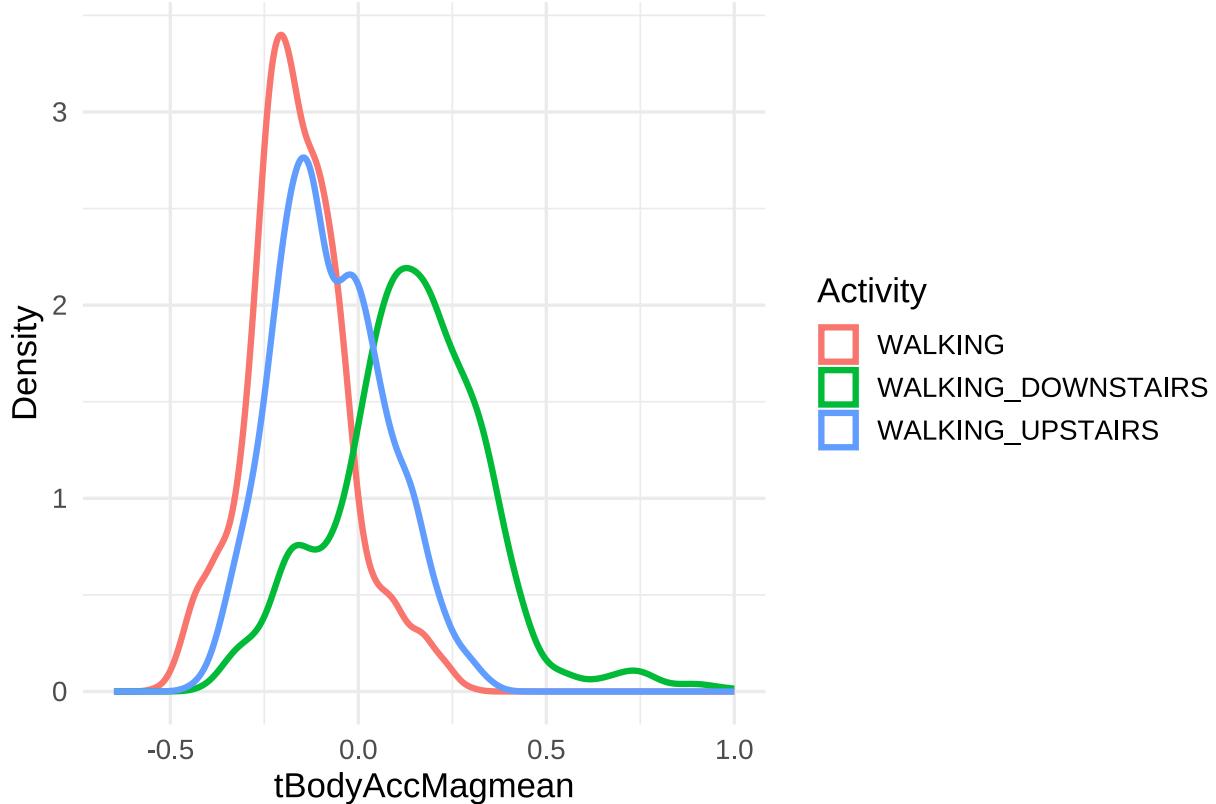
```

  labs(
    title = "Dynamic Activities (closer view)",
    x = "tBodyAccMagmean",
    y = "Density"
  ) + xlim(-0.65, 1) +
  theme_minimal(base_size = 14)

p2

```

Dynamic Activities (closer view)



Hình 4: Phân bố tBodyAccMagmean theo hoạt động walking, walking_downstairs, walking_upstairs

```

library(ggplot2)

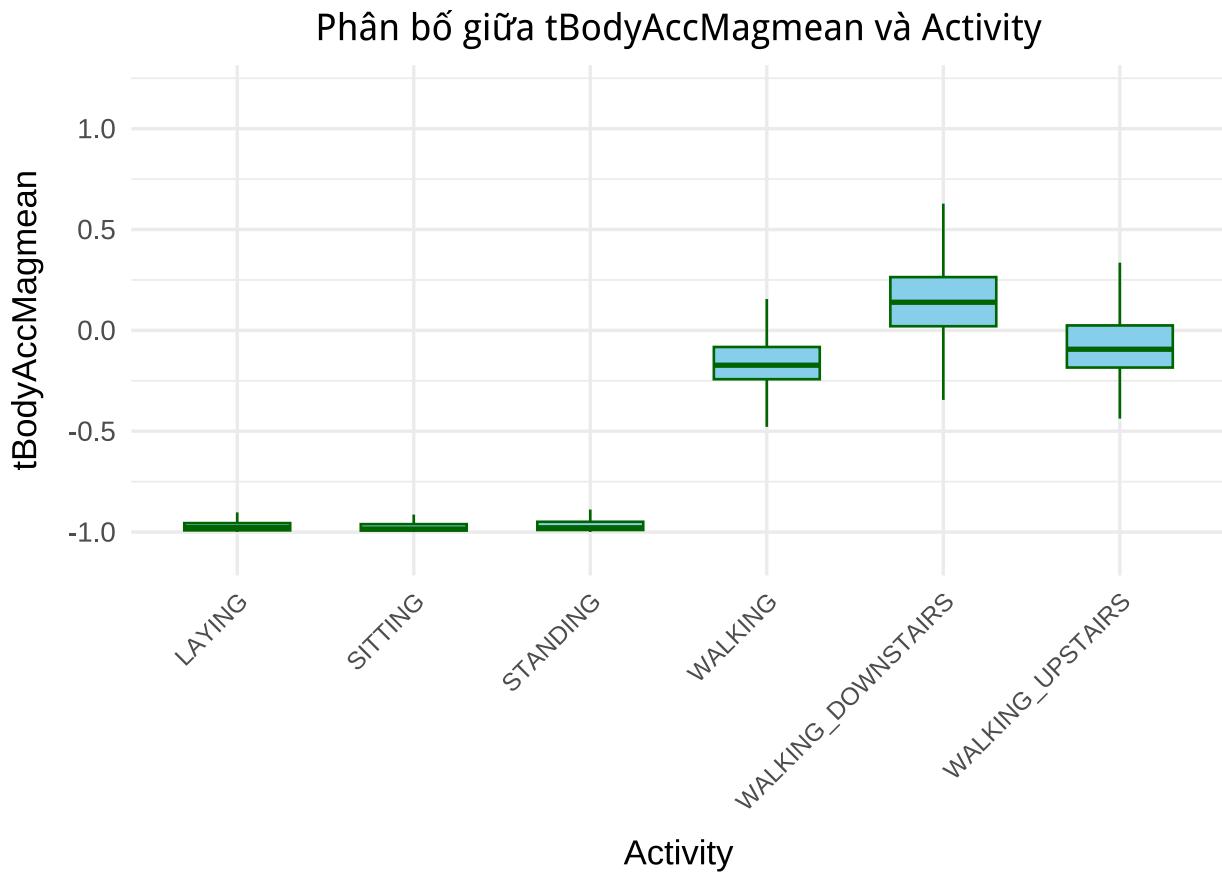
ggplot(train, aes(x = Activity, y = tBodyAccMagmean)) +
  geom_boxplot(
    outlier.shape = NA,
    fill = "skyblue",
    color = "darkgreen",
    width = 0.6
  ) +
  labs(
    title = "Phân bố giữa tBodyAccMagmean và Activity",
    y = "tBodyAccMagmean",

```

```

x = "Activity"
) +
coord_cartesian(ylim = c(-1.1, 1.2)) +
theme_minimal(base_size = 14) +
theme(
  plot.title = element_text(hjust = 0.5, size = 15, face = "bold"),
  axis.text.x = element_text(angle = 45, hjust = 1, size = 10),
  axis.title.x = element_text(margin = margin(t = 10)),
  axis.title.y = element_text(margin = margin(r = 10))
)

```



Hình 5: Phân bố giữa tBodyAccMagmean và Activity

```

library(ggplot2)

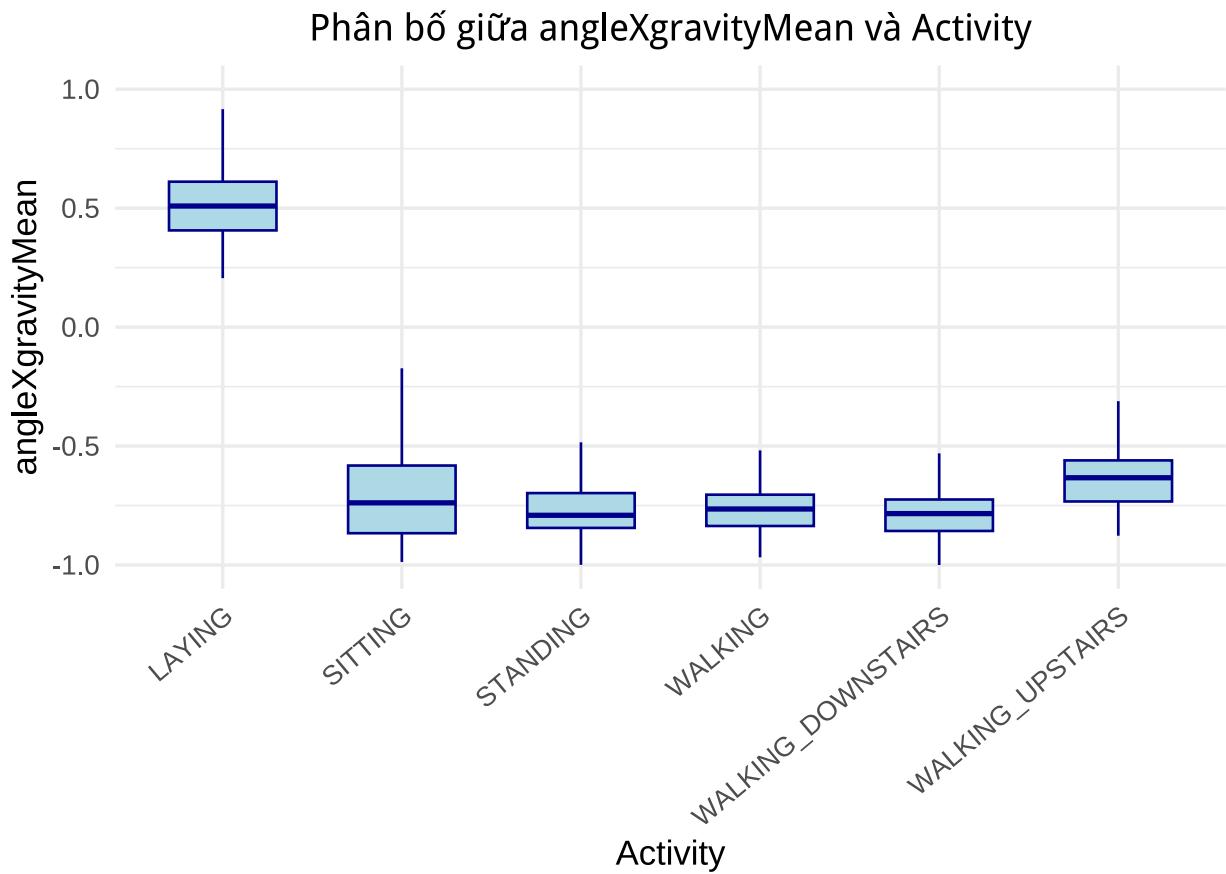
ggplot(train, aes(x = Activity, y = angleXgravityMean)) +
  geom_boxplot(
    fill = "lightblue",
    color = "darkblue",
    outlier.shape = NA,
    width = 0.6
  ) +
  labs(

```

```

    title = "Phân bố giữa angleXgravityMean và Activity",
    x = "Activity",
    y = "angleXgravityMean"
) +
theme_minimal(base_size = 14) +
theme(
  plot.title = element_text(hjust = 0.5, size = 15, face = "bold"),
  axis.text.x = element_text(angle = 40, hjust = 1, size = 11)
)

```



Hình 6: Phân bố giữa angleXgravityMean và Activity

```

library(ggplot2)

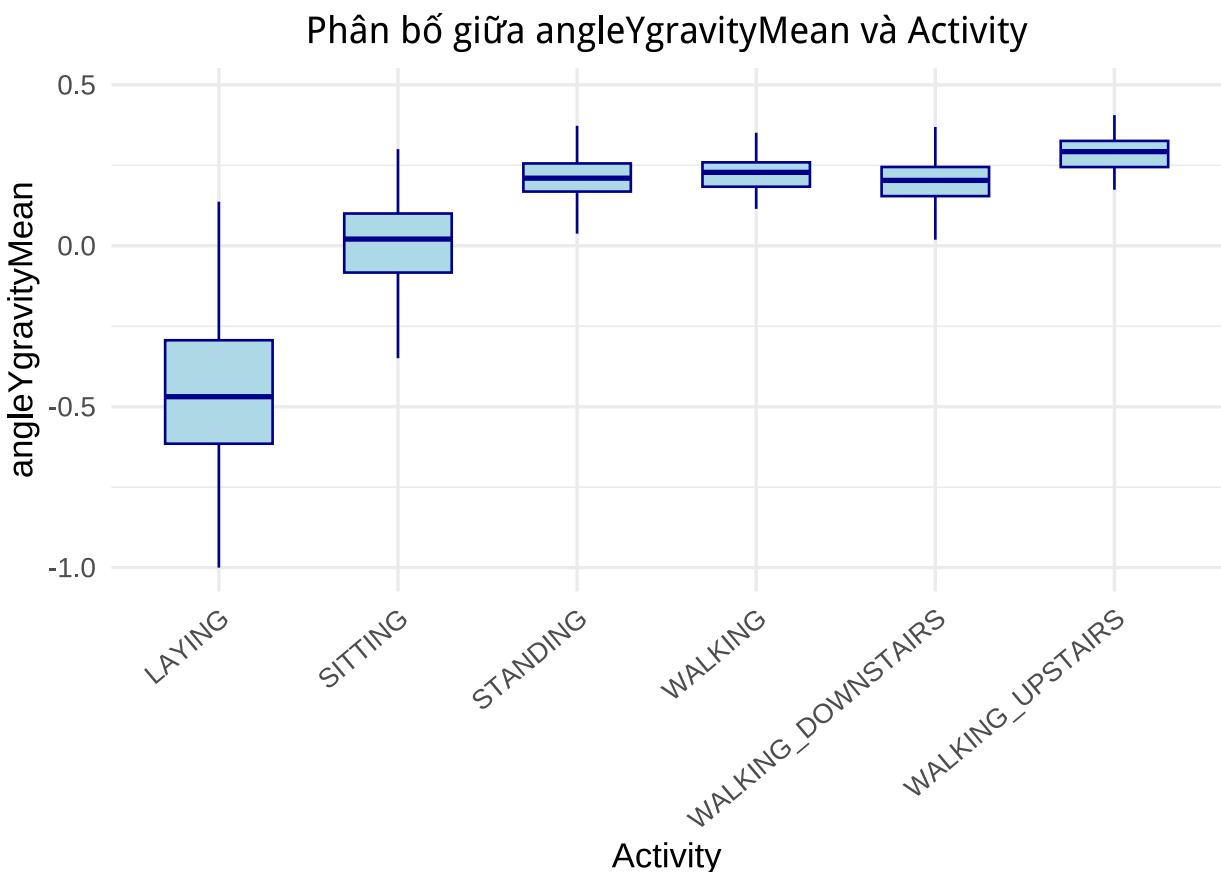
ggplot(train, aes(x = Activity, y = angleYgravityMean)) +
  geom_boxplot(
    fill = "lightblue",
    color = "darkblue",
    outlier.shape = NA,
    width = 0.6
) +
  labs(
    title = "Phân bố giữa angleYgravityMean và Activity",

```

```

x = "Activity",
y = "angleYgravityMean"
) +
theme_minimal(base_size = 14) +
theme(
  plot.title = element_text(hjust = 0.5, size = 15, face = "bold"),
  axis.text.x = element_text(angle = 40, hjust = 1, size = 11)
)

```



Đặc trưng tBodyAccMagmean – đại diện cho độ lớn trung bình của gia tốc cơ thể trong mỗi phân đoạn tín hiệu. Biểu đồ mật độ cho thấy ba hoạt động tĩnh đều có phân phối rất hẹp, tập trung gần giá trị -1.0, với đỉnh phân phối cao và sắc. Điều này phản ánh rằng khi người dùng không di chuyển, thiết bị ở thắt lưng ghi nhận gia tốc gần như bằng 0 trong toàn bộ cửa sổ thời gian. Trong đó, hoạt động SITTING có mật độ tập trung cao nhất, cho thấy đây là tư thế có ít dao động nhất; STANDING có phân bố hơi rộng hơn một chút, có thể do người thực hiện có xu hướng chuyển trọng lượng nhẹ nhàng khi đứng. Ngược lại, hoạt động LAYING lại xuất hiện sự phân tán hơn – phản ánh trạng thái nằm có thể mang nhiều tư thế khác nhau (nằm nghiêng, nằm ngửa...).

AngleXgravityMean cho thấy sự khác biệt rõ giữa tư thế nằm và các tư thế còn lại: trong khi LAYING có giá trị trung bình dương lớn ($\approx 0.5\text{--}0.7$), các hoạt động khác có xu hướng âm rõ rệt, thường nằm trong khoảng -0.6 đến -1.0. Là chỉ báo mạnh về góc nghiêng của cơ thể theo trục trước-sau, rất hữu ích trong việc phân biệt trạng thái nghỉ ngơi so với trạng thái hoạt động.

AngleYgravityMean – thể hiện góc nghiêng cơ thể theo trục ngang. LAYING có giá trị âm lớn (≈ -0.5), SITTING nằm quanh giá trị 0, trong khi STANDING và các hoạt động di chuyển đều có giá trị dương và tập trung gần nhau, phản ánh trạng thái thẳng đứng của cơ thể. Các đặc trưng liên quan đến góc độ không gian của thiết bị có khả năng phân biệt rõ tư thế mà không cần đến các đặc trưng vận tốc hay gia tốc.

Phân tích các đặc trưng tBodyAccMagmean, angleXgravityMean và angleYgravityMean không chỉ làm rõ cấu trúc phân bố của dữ liệu, mà còn cho thấy mỗi hoạt động trong bộ dữ liệu HAR đều thể hiện dấu hiệu đặc trưng về mặt hình học hoặc vận động học, đóng vai trò quan trọng trong việc xây dựng các mô hình học máy chính xác nhằm phân loại hành vi người dùng.

CHƯƠNG 4: THỰC NGHIỆM

4.1 Kiểm tra outlier

Tiến hành kiểm tra outlier trên tập train và tập test, xác định các điểm ngoại lai bằng phương pháp khoảng từ phân vị (Interquartile Range – IQR). Loại bỏ biến subject và Activity để tạo ra tập dữ liệu sensor_data, bao gồm 561 đặc trưng cảm biến, với mỗi cột trong tập dữ liệu xác định ranh giới bất thường dựa trên công thức:

$$\text{Lower bound} = Q1 - 1.5 \times IQR, \quad \text{Upper bound} = Q3 + 1.5 \times IQR$$

Trong đó $Q1$ và $Q3$ là các tứ phân vị thứ nhất và thứ ba tương ứng, và $IQR = Q3 - Q1$. Mọi giá trị vượt ngoài khoảng này được coi là ngoại lai.

```
subject_ids <- train$subject
activity_labels <- train$Activity
sensor_data <- train[, !(names(train) %in% c("Activity", "subject"))]
subject_ids_test <- test$subject
activity_labels_test <- test$Activity
sensor_data_test <- test[, !(names(test) %in% c("Activity", "subject"))]

is_outlier_iqr <- function(x) {
  Q1 <- quantile(x, 0.25, na.rm = TRUE)
  Q3 <- quantile(x, 0.75, na.rm = TRUE)
  IQR_val <- Q3 - Q1
  lower_bound <- Q1 - 1.5 * IQR_val
  upper_bound <- Q3 + 1.5 * IQR_val
  x < lower_bound | x > upper_bound
}

outlier_matrix <- mapply(is_outlier_iqr, sensor_data)

total_outlier_values <- sum(outlier_matrix)
total_cells <- nrow(sensor_data) * ncol(sensor_data)
percent_outlier_cells <- round(total_outlier_values / total_cells * 100, 2)

cat("Tổng số giá trị outlier:", total_outlier_values, "/", total_cells, "\n")
```

```
## Tổng số giá trị outlier: 163682 / 4124472
```

```
cat("Tỷ lệ giá trị outlier:", percent_outlier_cells, "%\n")
```

```
## Tỷ lệ giá trị outlier: 3.97 %
```

Kết quả có tổng cộng 163,682 giá trị được đánh dấu là ngoại lai, trên tổng số 4,124,472 giá trị dữ liệu. Điều này tương ứng với tỷ lệ 3.97%, một mức thấp, dữ liệu phần lớn ổn định và sạch.

- Tiến hành Chuẩn hóa outlier bằng kỹ thuật cắt ngưỡng (Percentile Capping) nhằm giảm ảnh hưởng của các outlier mà không loại bỏ chúng hoàn toàn khỏi tập dữ liệu. Thay thế các giá trị cực đoan bằng một ngưỡng giới hạn mềm dựa trên các phân vị phần trăm (percentiles) của phân phối dữ liệu. định nghĩa một hàm cap_outliers() để áp dụng quy tắc sau trên từng cột trong bảng dữ liệu: giá trị nhỏ hơn phân vị 1% (1st percentile) được gán bằng chính giá trị tại phân vị này, giá trị lớn hơn phân vị 99% (99th percentile) được gán bằng giá trị tại phân vị đó, các giá trị nằm giữa hai ngưỡng này được giữ nguyên.

```
cap_outliers <- function(df, lower = 0.01, upper = 0.99) {
  for (col in names(df)) {
    q_low <- quantile(df[[col]], lower, na.rm = TRUE)
    q_high <- quantile(df[[col]], upper, na.rm = TRUE)
    df[[col]] <- pmin(pmax(df[[col]], q_low), q_high)
  }
  df
}

sensor_data_capped <- cap_outliers(sensor_data)
sensor_data_test_capped <- cap_outliers(sensor_data_test)

train_capped <- cbind(sensor_data_capped, subject = subject_ids, Activity = activity_labels)
test_capped <- cbind(sensor_data_test_capped, subject = subject_ids_test, Activity = activity_labels_test)

# str(train_capped)
# str(test_capped)
```

4.2 Giảm chiều dữ liệu bằng UMAP

Kỹ thuật UMAP sử dụng để giảm chiều dữ liệu nhằm mục tiêu trực quan hóa và đánh giá khả năng phân tách giữa các nhãn hoạt động khác nhau. Việc giảm chiều không gian về 2 chiều giúp quan sát cấu trúc không gian nhúng của dữ liệu gốc gồm 561 đặc trưng đầu vào. Thủ nghiệm hệ thống trên lưới các tổ hợp siêu tham số khác nhau, số lượng lân cận (n_neighbors): 5, 15, 30 thể hiện mức độ chú trọng vào cấu trúc cục bộ hay toàn cục. Giá trị nhỏ nhấn mạnh mối quan hệ địa phương, giá trị lớn mang tính toàn cục cao hơn. Khoảng cách tối thiểu (min_dist): 0.001, 0.01, 0.1 điều chỉnh mức độ nén các cụm dữ liệu trong không gian nhúng. Giá trị càng nhỏ thì các cụm càng bị nén lại gần nhau, cho phép bảo toàn mật độ; giá trị lớn làm dàn trải không gian nhúng. Kết quả được trình bày thông qua ma trận trực quan gồm 9 biểu đồ, tương ứng với 9 tổ hợp siêu tham số khác nhau. Mỗi điểm trên biểu đồ đại diện cho một mẫu dữ liệu cảm biến, được tô màu theo nhãn Activity.

```
library(umap)
library(ggplot2)
library(gridExtra)

perform_umap_grid <- function(X_data, y_data,
                                neighbors_list = c(5, 15, 30),
                                min_dist_list = c(0.001, 0.01, 0.1)) {

  plots <- list()
  index <- 1

  for (n in neighbors_list) {
    for (d in min_dist_list) {
      config <- umap.defaults
      config$n_neighbors <- n
      config$min_dist <- d
```

```

config$n_components <- 2

umap_result <- umap(X_data, config = config)
df <- as.data.frame(umap_result$layout)
colnames(df) <- c("x", "y")
df$label <- y_data

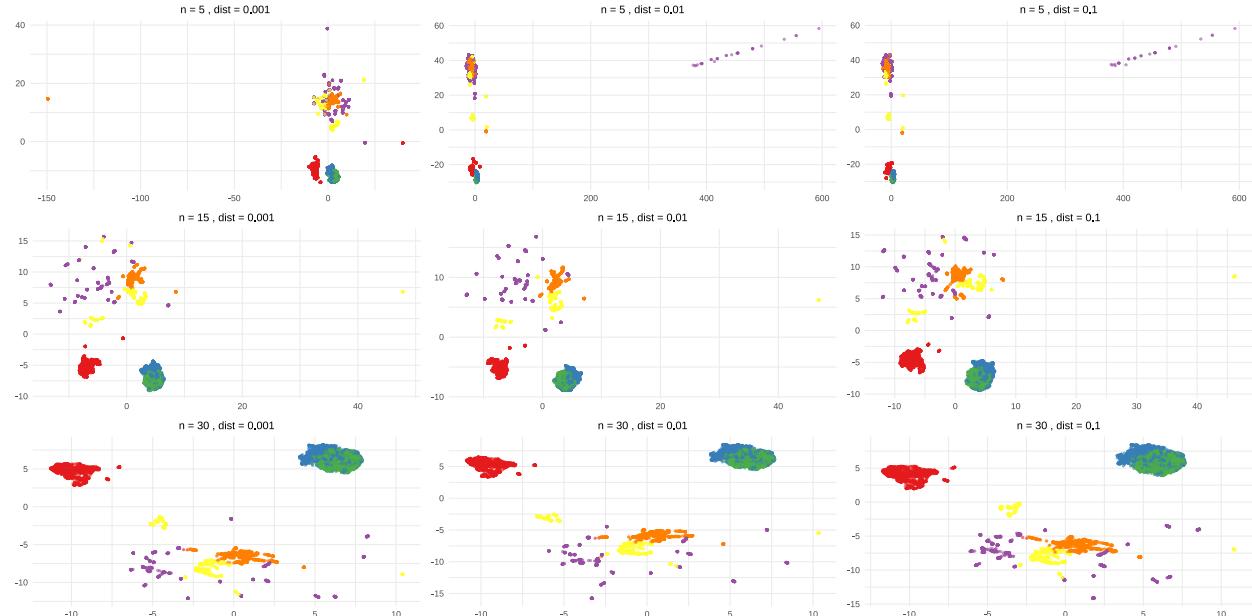
p <- ggplot(df, aes(x = x, y = y, color = label)) +
  geom_point(size = 0.6, alpha = 0.6) +
  scale_color_brewer(palette = "Set1") +
  theme_minimal(base_size = 10) +
  labs(title = paste("n =", n, ", dist =", d), x = NULL, y = NULL) +
  theme(legend.position = "none",
        plot.title = element_text(size = 10, hjust = 0.5))

plots[[index]] <- p
index <- index + 1
}
}

do.call(grid.arrange, c(plots, ncol = length(min_dist_list)))
}

perform_umap_grid(
  X_data = sensor_data_capped,
  y_data = train$Activity,
  neighbors_list = c(5, 15, 30),
  min_dist_list = c(0.001, 0.01, 0.1)
)

```



Hình 7: trực quan hóa UMAP

Khi **n_neighbors = 5** mạng lưới lân cận nhỏ, cấu trúc cục bộ lỏng lẻo và thiếu kết nối giữa các cụm, các cụm có

mạnh nhưng phân bố không đồng đều. **n_neighbors = 15** cấu trúc dữ liệu trở nên rõ ràng hơn. Khi kết hợp với min dist = 0.01, các cụm đại diện cho các chức năng khác nhau bắt đầu được hình thành rõ ràng và phân chia khá mạch lạc, thể hiện sự phân cấp cao giữa các đối tượng, các cụm nhỏ như sitting và laying bắt đầu được gom sát nhau. **n_neighbors = 30** thể hiện cấu trúc dữ liệu ổn định hơn. Dữ liệu được sắp xếp hợp lý, thấy được sự ổn định trong phân cụm. Đặc biệt với min_dist = 0.1, dữ liệu dàn rộng hơn trong không gian nhung, phù hợp cho việc phân tích tổng thể.

Kết quả phân tích thấy được các siêu tham số n_neighbors = 30 và min dist = 0.1 tạo ra hình ảnh không gian nhung chính xác và có sự phân biệt tốt nhất giữa các hoạt động. Đây là cơ sở quan trọng giúp xác định cấu hình tối ưu khi dùng UMAP như một bước tiền xử lý trong pipeline học máy.

- Dựa vào kết phân tích hệ thống các siêu tham số trong kỹ thuật giảm chiều UMAP thì cấu hình n_neighbors = 30 và min_dist = 0.01 đã được xác định là một trong những cấu hình hiệu quả nhất để đảm bảo cấu trúc dữ liệu và phân tách các nhãn rõ ràng. Chúng em sẽ tiến hành minh họa kết quả giảm chiều dữ liệu cảm biến từ 561 đặc trưng về 2 chiều, áp dụng các cấu hình tối ưu nêu trên, mỗi điểm là đại diện cho một mẫu dữ liệu và được tô màu theo loại hoạt động được gán nhãn.

```
library(umap)
library(ggplot2)

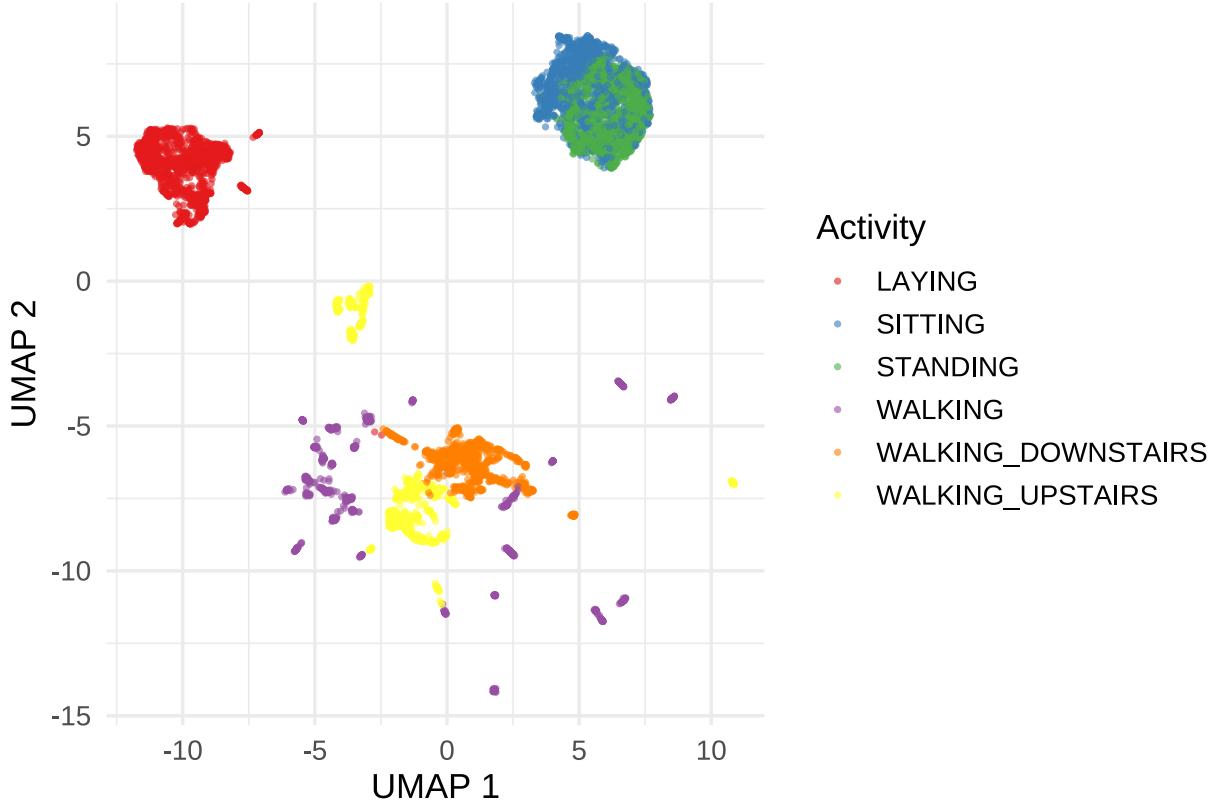
config <- umap.defaults
config$n_neighbors <- 30
config$min_dist <- 0.1
config$n_components <- 2

umap_result <- umap(sensor_data_capped, config = config)

umap_df <- as.data.frame(umap_result$layout)
colnames(umap_df) <- c("UMAP_1", "UMAP_2")
umap_df$Activity <- train$Activity

ggplot(umap_df, aes(x = UMAP_1, y = UMAP_2, color = Activity)) +
  geom_point(size = 0.7, alpha = 0.6) +
  scale_color_brewer(palette = "Set1") +
  theme_minimal(base_size = 14) +
  labs(
    title = "UMAP (n_neighbors = 30, min_dist = 0.01)",
    x = "UMAP 1", y = "UMAP 2"
  )
```

UMAP (`n_neighbors = 30, min_dist = 0.01`)



Các hoạt động như LAYING, SITTING và STANDING tạo thành ba cụm rõ ràng, thấy được tín hiệu cảm biến ở trong thái tĩnh có sự ổn định cao và dễ phân biệt. Các hoạt động như WALKING, WALKING_DOWNSTAIRS, WALKING_UPSTAIRS có xu hướng gần nhau hơn trong không gian 2 chiều, mặc dù vẫn tạo thành các cụm riêng biệt, nhưng vẫn có vẻ chòng lán, cho thấy được biên thiên phức tạp của vận động trong thực tế. Thấy được sự tách biệt rõ ràng giữa các nhóm động và tĩnh. Đây là bước tiền xử lý và kiểm định trực quan quan trọng trước khi đưa dữ liệu vào các thuật toán học máy như Random Forest, SVM, Logistic Regression và k-NN.

4.3 Chuẩn bị dữ liệu cho mô hình học máy

Sau khi hoàn thành các bước tiền xử lý, tập dữ liệu được chia thành hai thành phần chính: ma trận đặc trưng đầu vào và nhãn đầu ra tương ứng. Các biến định danh như subject và Activity được loại bỏ khỏi ma trận đặc trưng, nhằm đảm bảo mô hình chỉ học từ các tín hiệu cảm biến mà không bị ảnh hưởng bởi thông tin định danh.

```
X_train <- train_capped[, !(names(train_capped) %in% c("subject", "Activity"))]
y_train <- train_capped$Activity

X_test <- test_capped[, !(names(test_capped) %in% c("subject", "Activity"))]
y_test <- test_capped$Activity

dim(X_train)
```

```
## [1] 7352 561
```

```
length(y_train)
```

```
## [1] 7352
```

```
dim(X_test)
```

```
## [1] 2947 561
```

```
length(y_test)
```

```
## [1] 2947
```

4.4 xây dựng hàm đánh giá tổng quát mô hình học máy

Để đảm bảo sự thống nhất và tái sử dụng trong việc đào tạo và đánh giá các mô hình học máy, một hàm tổng quát có tên `train_and_evaluate_model` đã được thiết lập. Hàm thực hiện quy trình huấn luyện, dự đoán, đánh giá và trực quan hóa kết quả của một mô hình phân loại với cấu trúc linh hoạt, cho phép sử dụng nhiều thuật toán khác nhau như Random Forest, SVM, Logistic Regression, hoặc k-NN.

```
train_and_evaluate_model <- function(
  model_name,
  X_train, y_train, X_test, y_test,
  class_labels,
  tuneGrid = NULL,
  k_folds = 5
) {
  if (!require(caret)) install.packages("caret", quiet = TRUE)
  if (!require(ggplot2)) install.packages("ggplot2", quiet = TRUE)

  library(caret)
  library(ggplot2)

  y_train <- as.factor(y_train)
  y_test <- as.factor(y_test)

  ctrl <- trainControl(method = "cv", number = k_folds)

  set.seed(123)
  model_fit <- train(
    x = X_train,
    y = y_train,
    method = model_name,
    trControl = ctrl,
    tuneGrid = tuneGrid
  )

  y_pred <- predict(model_fit, X_test)

  # Accuracy
  accuracy <- mean(y_pred == y_test)
```

```

cat("-----\n")
cat("|      Accuracy      |\n")
cat("-----\n")
cat(sprintf("\n      %.4f\n\n", accuracy))

# Confusion matrix
cm <- table(Predicted = y_pred, Actual = y_test)
cat("-----\n")
cat("| Confusion Matrix |\n")
cat("-----\n")
print(cm)

# Plot confusion matrix
cm_df <- as.data.frame(cm)
colnames(cm_df) <- c("Predicted", "Actual", "Freq")

ggplot(data = cm_df, aes(x = Actual, y = Predicted, fill = Freq)) +
  geom_tile(color = "white") +
  geom_text(aes(label = Freq), color = "black", size = 5) +
  scale_fill_gradient(low = "white", high = "darkgreen") +
  labs(title = "Confusion Matrix", x = "True Label", y = "Predicted Label") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Classification report
cat("-----\n")
cat("| Classification Report |\n")
cat("-----\n")
print(confusionMatrix(y_pred, y_test)$byClass)

cat('-----\n')
cat('|      Best Parameters      |\n')
cat('-----\n')
print(model_fit$bestTune)

cat('-----\n')
cat('|      Best Accuracy      |\n')
cat('-----\n')
print(max(model_fit$results$Accuracy))

return(list(
  model = model_fit,
  accuracy = accuracy,
  confusion_matrix = cm,
  prediction = y_pred
))
}

grid_logistic <- expand.grid(
  alpha = c(0, 1),
  lambda = 1 / c(0.01, 0.1, 1, 10, 20, 30)
)

```

```

results_log <- train_and_evaluate_model(
  model_name = "glmnet",
  X_train = X_train, y_train = y_train,
  X_test = X_test, y_test = y_test,
  class_labels = labels,
  tuneGrid = grid_logistic,
  k_folds = 5
)

## Loading required package: caret

## Loading required package: lattice

## -----
## |      Accuracy      |
## -----
## 
##      0.9488
## 
## -----
## | Confusion Matrix |
## -----
## 
##          Actual
## Predicted      LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS
##   LAYING          537     0       0       0           0
##   SITTING         0     431     30       0           0
##   STANDING        0      57     502       0           0
##   WALKING         0      0       0     493          10
##   WALKING_DOWNSTAIRS 0      0       0       3         387
##   WALKING_UPSTAIRS 0      3       0       0          23
## 
##          Actual
## Predicted      WALKING_UPSTAIRS
##   LAYING          0
##   SITTING         0
##   STANDING        1
##   WALKING         22
##   WALKING_DOWNSTAIRS 2
##   WALKING_UPSTAIRS 446
## 
## -----
## | Classification Report |
## -----
## 
##          Sensitivity Specificity Pos Pred Value Neg Pred Value
## Class: LAYING           1.0000000 1.0000000 1.0000000 1.0000000
## Class: SITTING          0.8778004 0.9877850 0.9349241 0.9758648
## Class: STANDING          0.9436090 0.9759834 0.8964286 0.9874319
## Class: WALKING           0.9939516 0.9869441 0.9390476 0.9987614
## Class: WALKING_DOWNSTAIRS 0.9214286 0.9980214 0.9872449 0.9870841
## Class: WALKING_UPSTAIRS 0.9469214 0.9894992 0.9449153 0.9898990
## 
##          Precision Recall      F1 Prevalence
## Class: LAYING           1.0000000 1.0000000 1.0000000 0.1822192
## Class: SITTING          0.9349241 0.8778004 0.9054622 0.1666101
## Class: STANDING          0.8964286 0.9436090 0.9194139 0.1805226
## Class: WALKING           0.9390476 0.9939516 0.9657199 0.1683068

```

```

## Class: WALKING_DOWNSTAIRS 0.9872449 0.9214286 0.9532020 0.1425178
## Class: WALKING_UPSTAIRS 0.9449153 0.9469214 0.9459173 0.1598235
## Detection Rate Detection Prevalence Balanced Accuracy
## Class: LAYING 0.1822192 0.1822192 1.0000000
## Class: SITTING 0.1462504 0.1564303 0.9327927
## Class: STANDING 0.1703427 0.1900238 0.9597962
## Class: WALKING 0.1672888 0.1781473 0.9904479
## Class: WALKING_DOWNSTAIRS 0.1313200 0.1330166 0.9597250
## Class: WALKING_UPSTAIRS 0.1513403 0.1601629 0.9682103
## -----
## | Best Parameters |
## -----
## alpha lambda
## 1 0 0.03333333
## -----
## | Best Accuracy |
## -----
## [1] 0.9688516

grid_linear_svc <- expand.grid(C = c(0.125, 0.5, 1, 2, 8, 16))

results_svc <- train_and_evaluate_model(
  model_name = "svmLinear",
  X_train = X_train, y_train = y_train,
  X_test = X_test, y_test = y_test,
  class_labels = labels,
  tuneGrid = grid_linear_svc,
  k_folds = 5
)

## -----
## | Accuracy |
## -----
## 0.9606
## -----
## | Confusion Matrix |
## -----
## Predicted      Actual
##          LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS
## LAYING 537 0 0 0 0
## SITTING 0 436 16 0 0
## STANDING 0 53 516 0 0
## WALKING 0 0 0 493 5
## WALKING_DOWNSTAIRS 0 0 0 2 399
## WALKING_UPSTAIRS 0 2 0 1 16
## Predicted      Actual
##          WALKING_UPSTAIRS
## LAYING 0
## SITTING 0
## STANDING 0
## WALKING 16
## WALKING_DOWNSTAIRS 5

```

```

##      WALKING_UPSTAIRS          450
## -----
## | Classification Report |
## -----
##                               Sensitivity Specificity Pos Pred Value Neg Pred Value
## Class: LAYING           1.0000000 1.0000000 1.0000000 1.0000000
## Class: SITTING          0.8879837 0.9934853 0.9646018 0.9779559
## Class: STANDING         0.9699248 0.9780538 0.9068541 0.9932717
## Class: WALKING          0.9939516 0.9914321 0.9591440 0.9987670
## Class: WALKING_DOWNSTAIRS 0.9500000 0.9972299 0.9827586 0.9917355
## Class: WALKING_UPSTAIRS 0.9554140 0.9923263 0.9594883 0.9915254
##                               Precision Recall          F1 Prevalence
## Class: LAYING          1.0000000 1.0000000 1.0000000 0.1822192
## Class: SITTING          0.9646018 0.8879837 0.9247084 0.1666101
## Class: STANDING         0.9068541 0.9699248 0.9373297 0.1805226
## Class: WALKING          0.9591440 0.9939516 0.9762376 0.1683068
## Class: WALKING_DOWNSTAIRS 0.9827586 0.9500000 0.9661017 0.1425178
## Class: WALKING_UPSTAIRS 0.9594883 0.9554140 0.9574468 0.1598235
##                               Detection Rate Detection Prevalence Balanced Accuracy
## Class: LAYING          0.1822192          0.1822192 1.0000000
## Class: SITTING          0.1479471          0.1533763 0.9407345
## Class: STANDING         0.1750933          0.1930777 0.9739893
## Class: WALKING          0.1672888          0.1744147 0.9926918
## Class: WALKING_DOWNSTAIRS 0.1353919          0.1377672 0.9736150
## Class: WALKING_UPSTAIRS 0.1526977          0.1591449 0.9738702
## -----
## |      Best Parameters   |
## -----
##      C
## 1 0.125
## -----
## |      Best Accuracy    |
## -----
## [1] 0.9844929

grid_dt <- expand.grid(maxdepth = seq(3, 9, by = 2))

results_dt <- train_and_evaluate_model(
  model_name = "rpart2",
  X_train = X_train,
  y_train = y_train,
  X_test = X_test,
  y_test = y_test,
  class_labels = labels,
  tuneGrid = grid_dt,
  k_folds = 5
)

## -----
## |      Accuracy          |
## -----
## 
##      0.8402
## 
```

```

## -----
## | Confusion Matrix |
## -----
##          Actual
## Predicted      LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS
##   LAYING        537     0       0       0           0
##   SITTING        0    400     107      0           0
##   STANDING       0     91     425      0           0
##   WALKING        0     0       0     430         40
##   WALKING_DOWNSTAIRS 0     0       0     11        282
##   WALKING_UPSTAIRS 0     0       0     55         98
##          Actual
## Predicted      WALKING_UPSTAIRS
##   LAYING          0
##   SITTING          0
##   STANDING          0
##   WALKING          61
##   WALKING_DOWNSTAIRS 8
##   WALKING_UPSTAIRS 402
## -----
## | Classification Report |
## -----
##          Sensitivity Specificity Pos Pred Value Neg Pred Value
## Class: LAYING        1.0000000 1.0000000 1.0000000 1.0000000
## Class: SITTING       0.8146640 0.9564332 0.7889546 0.9627049
## Class: STANDING      0.7988722 0.9623188 0.8236434 0.9559852
## Class: WALKING       0.8669355 0.9587923 0.8097928 0.9726821
## Class: WALKING_DOWNSTAIRS 0.6714286 0.9924812 0.9368771 0.9478458
## Class: WALKING_UPSTAIRS 0.8535032 0.9382068 0.7243243 0.9711538
##          Precision Recall      F1 Prevalence
## Class: LAYING        1.0000000 1.0000000 1.0000000 0.1822192
## Class: SITTING       0.7889546 0.8146640 0.8016032 0.1666101
## Class: STANDING      0.8236434 0.7988722 0.8110687 0.1805226
## Class: WALKING       0.8097928 0.8669355 0.8373905 0.1683068
## Class: WALKING_DOWNSTAIRS 0.9368771 0.6714286 0.7822469 0.1425178
## Class: WALKING_UPSTAIRS 0.7243243 0.8535032 0.7836257 0.1598235
##          Detection Rate Detection Prevalence Balanced Accuracy
## Class: LAYING        0.18221921 0.1822192 1.0000000
## Class: SITTING       0.13573125 0.1720394 0.8855486
## Class: STANDING      0.14421446 0.1750933 0.8805955
## Class: WALKING       0.14591110 0.1801832 0.9128639
## Class: WALKING_DOWNSTAIRS 0.09569053 0.1021378 0.8319549
## Class: WALKING_UPSTAIRS 0.13640991 0.1883271 0.8958550
## -----
## |      Best Parameters |
## -----
## maxdepth
## 3      7
## -----
## |      Best Accuracy |
## -----
## [1] 0.8860176

```

```

library(randomForest)

## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

## 
## Attaching package: 'randomForest'

## The following object is masked from 'package:gridExtra':
## 
##     combine

## The following object is masked from 'package:dplyr':
## 
##     combine

## The following object is masked from 'package:ggplot2':
## 
##     margin

grid_rf <- expand.grid(mtry = c(16, 20, 24, 28, 32))
results_rf <- train_and_evaluate_model(
  model_name = "rf",
  X_train = X_train,
  y_train = y_train,
  X_test = X_test,
  y_test = y_test,
  class_labels = labels,
  tuneGrid = grid_rf,
  k_folds = 5
)

## -----
## |      Accuracy      |
## -----
## 
##      0.9355
## 
## -----
## | Confusion Matrix |
## -----
## 
##          Actual
## Predicted      LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS
## LAYING           537      0       0       0           0
## SITTING          0     448      29       0           0
## STANDING         0      43     503       0           0
## WALKING          0      0       0     480          19
## WALKING_DOWNSTAIRS 0      0       0       8        354
## WALKING_UPSTAIRS 0      0       0       8          47
## 
##          Actual

```

```

## Predicted          WALKING_UPSTAIRS
##   LAYING                      0
##   SITTING                     0
##   STANDING                    0
##   WALKING                     30
##   WALKING_DOWNSTAIRS           6
##   WALKING_UPSTAIRS            435
## -----
## | Classification Report |
## -----
##                               Sensitivity Specificity Pos Pred Value Neg Pred Value
## Class: LAYING             1.0000000 1.0000000 1.0000000 1.0000000
## Class: SITTING            0.9124236 0.9881922 0.9392034 0.9825911
## Class: STANDING           0.9454887 0.9821946 0.9212454 0.9879217
## Class: WALKING            0.9677419 0.9800082 0.9073724 0.9933830
## Class: WALKING_DOWNSTAIRS 0.8428571 0.9944598 0.9619565 0.9744087
## Class: WALKING_UPSTAIRS   0.9235669 0.9777868 0.8877551 0.9853480
##                               Precision    Recall      F1 Prevalence
## Class: LAYING            1.0000000 1.0000000 1.0000000 0.1822192
## Class: SITTING           0.9392034 0.9124236 0.9256198 0.1666101
## Class: STANDING          0.9212454 0.9454887 0.9332096 0.1805226
## Class: WALKING           0.9073724 0.9677419 0.9365854 0.1683068
## Class: WALKING_DOWNSTAIRS 0.9619565 0.8428571 0.8984772 0.1425178
## Class: WALKING_UPSTAIRS  0.8877551 0.9235669 0.9053070 0.1598235
##                               Detection Rate Detection Prevalence Balanced Accuracy
## Class: LAYING            0.1822192          0.1822192 1.0000000
## Class: SITTING           0.1520190          0.1618595 0.9503079
## Class: STANDING          0.1706820          0.1852732 0.9638417
## Class: WALKING           0.1628775          0.1795046 0.9738750
## Class: WALKING_DOWNSTAIRS 0.1201222          0.1248728 0.9186585
## Class: WALKING_UPSTAIRS  0.1476077          0.1662708 0.9506768
## -----
## |      Best Parameters   |
## -----
##   mtry
## 1 16
## -----
## |      Best Accuracy    |
## -----
## [1] 0.9829978

```

```

grid_knn <- expand.grid(k = seq(1, 30, by = 2))

results_knn <- train_and_evaluate_model(
  model_name = "knn",
  X_train = X_train,
  y_train = y_train,
  X_test = X_test,
  y_test = y_test,
  class_labels = labels,
  tuneGrid = grid_knn,
  k_folds = 5
)

```

```

## -----
## | Accuracy |
## -----
## 
##      0.8799
## 
## -----
## | Confusion Matrix |
## -----
##          Actual
## Predicted    LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS
##   LAYING        535      1       0       0           0
##   SITTING        2     393      80       0           0
##   STANDING       0     96     452       0           0
##   WALKING        0      0       0     472         53
##   WALKING_DOWNSTAIRS 0      0       0     13        317
##   WALKING_UPSTAIRS 0      1       0     11         50
##          Actual
## Predicted    WALKING_UPSTAIRS
##   LAYING          0
##   SITTING          0
##   STANDING          0
##   WALKING          29
##   WALKING_DOWNSTAIRS 18
##   WALKING_UPSTAIRS 424
## -----
## | Classification Report |
## -----
##          Sensitivity Specificity Pos Pred Value Neg Pred Value
## Class: LAYING        0.9962756  0.9995851  0.9981343  0.9991705
## Class: SITTING        0.8004073  0.9666124  0.8273684  0.9603560
## Class: STANDING       0.8496241  0.9602484  0.8248175  0.9666528
## Class: WALKING        0.9516129  0.9665443  0.8519856  0.9899707
## Class: WALKING_DOWNSTAIRS 0.7547619  0.9877325  0.9109195  0.9603694
## Class: WALKING_UPSTAIRS 0.9002123  0.9749596  0.8724280  0.9809021
##          Precision Recall      F1 Prevalence
## Class: LAYING        0.9981343  0.9962756  0.9972041  0.1822192
## Class: SITTING        0.8273684  0.8004073  0.8136646  0.1666101
## Class: STANDING       0.8248175  0.8496241  0.8370370  0.1805226
## Class: WALKING        0.8519856  0.9516129  0.8990476  0.1683068
## Class: WALKING_DOWNSTAIRS 0.9109195  0.7547619  0.8255208  0.1425178
## Class: WALKING_UPSTAIRS 0.8724280  0.9002123  0.8861024  0.1598235
##          Detection Rate Detection Prevalence Balanced Accuracy
## Class: LAYING        0.1815405            0.1818799  0.9979303
## Class: SITTING        0.1333560            0.1611809  0.8835099
## Class: STANDING       0.1533763            0.1859518  0.9049363
## Class: WALKING        0.1601629            0.1879878  0.9590786
## Class: WALKING_DOWNSTAIRS 0.1075670            0.1180862  0.8712472
## Class: WALKING_UPSTAIRS 0.1438751            0.1649135  0.9375860
## -----
## | Best Parameters |
## -----
## k
## 1 1

```

```
## -----  
## |      Best Accuracy      |  
## -----  
## [1] 0.9717066  
  
best_k <- results_knn$model$bestTune$k  
cat("Giá trị k tối ưu là:", best_k, "\n")  
  
## Giá trị k tối ưu là: 1
```