**TRƯỜNG ĐẠI HỌC THỦY LỢI**

**KHOA CÔNG NGHỆ THÔNG TIN**



**Báo cáo bài tập lớn**

**Học PHẦN học máy**

Dự đoán Khoảng giá điện thoại

**Giảng viên:** Tạ Quang Chiểu

**Nhóm thực hiện:** 12

**Lớp:** 64TTNT2

**Tên sinh viên:**

1. Nguyễn Đình Huân – 2251262607

2. Hoàng Minh Hiếu – 2251262601

3. Trần Thế Minh – 2251262619

4. Nguyễn Vinh Quang – 2251262631

**LỜI CẢM ƠN**

Lời đầu tiên, chúng em xin chân thành cảm ơn Khoa Công nghệ Thông tin, Trường Đại học Thuỷ Lợi đã tạo điều kiện tốt nhất để chúng em được học môn Học máy. Đặc biệt chúng em xin chân thành cảm ơn thầy Tạ Quang Chiểu – Giảng viên giảng dạy bộ môn Học máycủa lớp chúng em trong kỳ này.  Chúng em vô cùng biết ơn và xin kính chúc Thầy luôn mạnh khoẻ để tiếp tục cống hiến cho sự nghiệp giáo dục và đạt được nhiều thành công hơn nữa.

Với tất cả sự cố gắng của nhóm trong quá trình thực hiện đề tài, song do trình độ, hiểu biết nên đề tài khó tránh khỏi những thiếu sót. Chúng em mong nhận được sự góp ý, chỉ bảo của Thầy để đề tài được hoàn thiện hơn.

# CHƯƠNG 1: GIỚI THIỆU BÀI TOÁN

**1.1 Giới thiệu bài toán**

Trong thời đại công nghệ hiện nay, điện thoại di động đã trở thành một phần không thể thiếu trong cuộc sống của con người. Việc lựa chọn một chiếc điện thoại phù hợp với nhu cầu và khả năng tài chính của người dùng luôn là một vấn đề quan trọng. Với sự đa dạng của các sản phẩm trên thị trường, việc xác định giá trị của một chiếc điện thoại dựa trên các đặc tính kỹ thuật của nó là một bài toán thú vị và đầy thử thách.

Bài toán này liên quan đến việc dự đoán khoảng giá của điện thoại thông qua các yếu tố như dung lượng bộ nhớ, tốc độ xử lý, kích thước màn hình, công nghệ kết nối (4G, 3G), và các thông số kỹ thuật khác. Việc xây dựng một mô hình dự đoán giá điện thoại không chỉ giúp người tiêu dùng đưa ra quyết định mua sắm thông minh hơn mà còn có thể ứng dụng trong các hệ thống thương mại điện tử, hỗ trợ bán hàng và marketing.

Trong bài tập lớn này, chúng tôi sẽ áp dụng mô hình phân loại SVM (Support Vector Machine) kết hợp với chiến lược OVO (One-vs-One) để dự đoán khoảng giá của điện thoại. Để cải thiện hiệu suất của mô hình, các phương pháp tiền xử lý dữ liệu như StandardScaler sẽ được sử dụng để chuẩn hóa dữ liệu, giúp các thuật toán học máy hoạt động hiệu quả hơn. Ngoài ra, chúng tôi sẽ sử dụng các công cụ trực quan hóa dữ liệu như Heatmap và Boxplot để phân tích và trực quan hóa các mối quan hệ giữa các yếu tố kỹ thuật và giá của điện thoại.

Dữ liệu sử dụng trong bài toán bao gồm các thông số kỹ thuật của điện thoại, bao gồm dung lượng bộ nhớ, số lõi CPU, tốc độ xử lý, trọng lượng, và các tính năng như màn hình cảm ứng, kết nối Wi-Fi, 4G, và 3G. Dữ liệu này sẽ được xử lý và phân tích để xây dựng mô hình dự đoán giá chính xác.

Mục tiêu của bài tập là áp dụng các kỹ thuật học máy để giải quyết bài toán phân loại giá điện thoại, đồng thời nâng cao khả năng hiểu và áp dụng các phương pháp phân tích dữ liệu trong thực tế.

### 1.2 Các mô hình và phương pháp sử dụng

Trong bài tập này, chúng tôi sẽ sử dụng các mô hình và phương pháp học máy cũng như các kỹ thuật tiền xử lý và phân tích dữ liệu để giải quyết bài toán dự đoán khoảng giá điện thoại. Dưới đây là danh sách các mô hình và phương pháp được áp dụng:

#### 1.2.1 ****Mô hình SVM (Support Vector Machine) với chiến lược OVO (One-vs-One)****

* **Mô tả**: SVM là một thuật toán học máy mạnh mẽ, được sử dụng phổ biến trong các bài toán phân loại. Trong trường hợp này, chúng tôi sử dụng SVM để phân loại khoảng giá của điện thoại thành các lớp khác nhau. Chiến lược OVO (One-vs-One) sẽ được áp dụng để xử lý bài toán phân loại đa lớp, với mỗi cặp lớp được xử lý riêng biệt.
* **Ưu điểm**: SVM có khả năng phân loại tốt với dữ liệu có chiều cao và không gian lớn. Khi kết hợp với chiến lược OVO, SVM có thể xử lý hiệu quả các bài toán phân loại nhiều lớp.
* **Cách sử dụng**: SVM với chiến lược OVO sẽ được huấn luyện trên các đặc tính kỹ thuật của điện thoại (như bộ nhớ, số lõi, tốc độ xử lý, v.v.) và kết quả phân loại sẽ giúp xác định khoảng giá của từng chiếc điện thoại.

#### 1.2.2 ****StandardScaler (Chuẩn hóa dữ liệu)****

* **Mô tả**: StandardScaler là một phương pháp chuẩn hóa dữ liệu, giúp điều chỉnh các đặc tính dữ liệu sao cho có cùng đơn vị đo lường và giá trị trung bình bằng 0, độ lệch chuẩn bằng 1. Điều này giúp các mô hình học máy như SVM hoạt động hiệu quả hơn.
* **Ưu điểm**: Chuẩn hóa giúp cải thiện hiệu suất của các mô hình học máy, đặc biệt là với các thuật toán yêu cầu dữ liệu có độ đồng đều về thang đo.
* **Cách sử dụng**: Dữ liệu đầu vào sẽ được chuẩn hóa trước khi đưa vào mô hình SVM để đảm bảo các đặc tính có tầm ảnh hưởng tương đương trong quá trình huấn luyện.

#### 1.2.3 ****Heatmap (Bản đồ nhiệt)****

* **Mô tả**: Heatmap là một công cụ trực quan hóa dữ liệu, giúp thể hiện mối quan hệ giữa các đặc tính kỹ thuật của điện thoại. Các yếu tố tương quan giữa các đặc tính sẽ được thể hiện bằng màu sắc, giúp người phân tích dễ dàng nhận diện các mối liên hệ.
* **Ưu điểm**: Heatmap giúp phát hiện các mối quan hệ giữa các biến độc lập và biến phụ thuộc, từ đó hỗ trợ trong việc tối ưu hóa mô hình và phân tích dữ liệu.
* **Cách sử dụng**: Sử dụng heatmap để phân tích mối quan hệ giữa các đặc tính kỹ thuật của điện thoại (như dung lượng bộ nhớ, số lõi, tốc độ xử lý) và giá của điện thoại.

#### 1.2.4 ****Boxplot (Biểu đồ hộp)****

* **Mô tả**: Boxplot là một phương pháp trực quan hóa giúp hiển thị sự phân bố của dữ liệu, bao gồm các chỉ số như giá trị trung vị, các phần tử ngoài phạm vi (outliers), và sự phân tán của dữ liệu.
* **Ưu điểm**: Boxplot giúp nhận diện nhanh chóng các giá trị bất thường trong dữ liệu, hỗ trợ việc phân tích và xử lý dữ liệu.
* **Cách sử dụng**: Boxplot sẽ được sử dụng để trực quan hóa sự phân bố của giá điện thoại trong từng khoảng giá, từ đó đánh giá sự phân tán và tìm kiếm các giá trị ngoại lai trong dữ liệu.

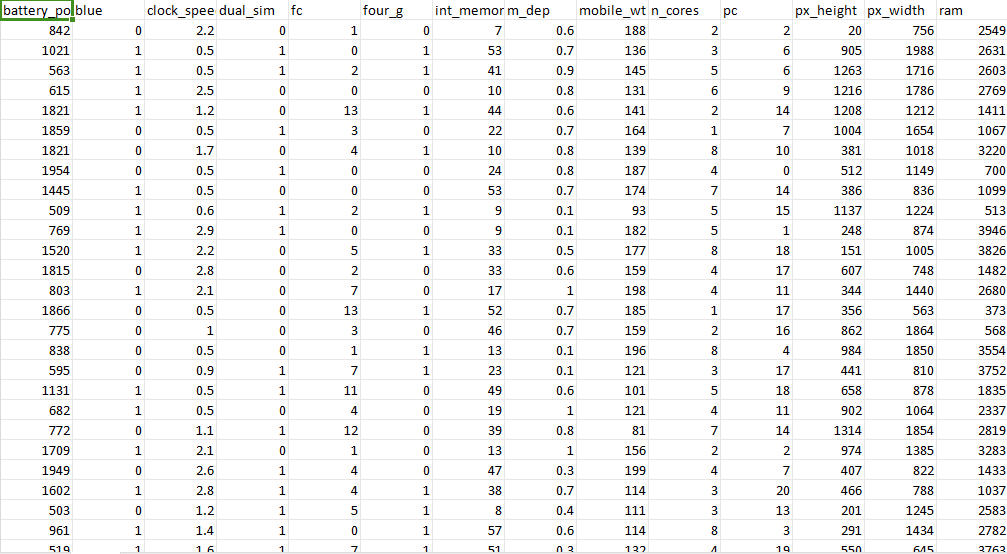
**1.2.5 IQR (Interquartile Range)**

* **Mô tả:** một phương pháp thống kê dùng để xác định giá trị ngoại lai (outliers) dựa trên khoảng tứ phân vị của dữ liệu.

#### 1.2.6 ****Các phương pháp phân tích và đánh giá mô hình****

* **Cross-validation (Kiểm tra chéo)**: Sử dụng kỹ thuật kiểm tra chéo để đánh giá hiệu suất của mô hình SVM. Điều này giúp đảm bảo rằng mô hình không bị overfitting và có thể tổng quát hóa tốt với dữ liệu chưa thấy.
* **Accuracy (Độ chính xác)**: Đo lường độ chính xác của mô hình phân loại để đánh giá khả năng dự đoán đúng khoảng giá của điện thoại.
* **Confusion Matrix (Ma trận nhầm lẫn)**: Được sử dụng để phân tích chi tiết về kết quả phân loại của mô hình, từ đó xác định được số lượng dự đoán đúng và sai cho mỗi lớp.

**Dữ liệu và các thuộc tính**



Dữ liệu mà chúng ta sử dụng có 20 thuộc tính kỹ thuật của các mẫu điện thoại, với 1 biến mục tiêu là **price\_range** (phân khúc giá). Các thuộc tính có thể bao gồm:

1. **Battery\_power**: Công suất pin của điện thoại.
2. **Blue**: Tính năng Bluetooth (0 hoặc 1).
3. **Clock\_speed**: Tốc độ xung nhịp của bộ vi xử lý.
4. **Dual\_sim**: Có hỗ trợ 2 SIM hay không.
5. **Fc**: Chất lượng camera trước.
6. **Four\_g**: Có hỗ trợ 4G hay không.
7. **Int\_memory**: Dung lượng bộ nhớ trong (GB).
8. **M\_dep**: Độ dày của điện thoại.
9. **Mobile\_wt**: Trọng lượng của điện thoại.
10. **N\_cores**: Số lõi của bộ vi xử lý.
11. **Pc**: Chất lượng camera chính.
12. **Px-height**: Chiều cao màn hình theo pixel
13. **Px-width**: Chiều rộng màn hình theo pixel
14. **Ram**: Dung lượng RAM (GB).
15. **Screen\_height**: Chiều cao của màn hình.
16. **Screen\_width**: Chiều rộng của màn hình.
17. **Talk\_time**: Thời gian gọi điện thoại liên tục.
18. **Three\_g**: Có hỗ trợ 3G hay không.
19. **Touch\_screen**: Có màn hình cảm ứng hay không.
20. **Wifi**: Có hỗ trợ Wi-Fi hay không.
21. **Price\_range**: Phân khúc giá của điện thoại (mục tiêu).

**Mục tiêu**

Mục tiêu của bài toán là dự đoán giá điện thoại thông qua các thuộc tính kỹ thuật. Giá điện thoại sẽ được phân thành các nhóm sau:

· **Mức giá thấp** (Low price) = 0

· **Mức giá trung bình thấp** (Low-medium price) = 1

· **Mức giá trung bình cao** (High-medium price) = 2

· **Mức giá cao** (High price) = 3

**Cách thức thực hiện**

**Tiền xử lý dữ liệu**:  
Dữ liệu sẽ được tiền xử lý để loại bỏ các giá trị thiếu (nếu có), chuẩn hóa các thuộc tính liên tục, và mã hóa các thuộc tính phân loại (nếu có). Chúng ta sẽ chia dữ liệu thành hai phần: một phần huấn luyện và một phần kiểm tra.

**Áp dụng SVM + OVO**:  
Sử dụng mô hình SVM kết hợp với phương pháp OVO để phân loại các mẫu điện thoại vào các phân khúc giá khác nhau. Với mỗi cặp phân khúc giá, một mô hình SVM sẽ được huấn luyện để phân biệt giữa hai phân khúc đó.

**Huấn luyện mô hình**:  
Sau khi dữ liệu đã được chuẩn bị, chúng ta sẽ huấn luyện mô hình SVM cho từng cặp phân khúc giá, sử dụng phương pháp OVO. Các mô hình sẽ được huấn luyện trên các đặc trưng của điện thoại và phân khúc giá mục tiêu.

**Dự đoán và đánh giá mô hình**:  
Khi mô hình đã được huấn luyện, chúng ta sẽ sử dụng tập kiểm tra để đánh giá hiệu quả của mô hình, sử dụng các chỉ số như độ chính xác (accuracy), F1-score, và confusion matrix để kiểm tra khả năng phân loại chính xác của mô hình.

# CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

## ****2.1. Tính Tương Quan****

### ****2.1.1 Khái niệm và Nguyên lý Hoạt động****

* **Mục tiêu:** Đo lường mức độ phụ thuộc giữa hai biến số trong dữ liệu.
* **Nguyên lý hoạt động:**
  1. Tính toán sự thay đổi của hai biến dựa trên giá trị trung bình của chúng.
  2. Hệ số tương quan (correlation coefficient) rrr được tính theo công thức:



Trong đó:

*  và : Các giá trị của hai biến X và Y.
*  và  : Giá trị trung bình của X và Y.
* r: Hệ số tương quan (giá trị nằm trong khoảng [−1,1]).

### ****2.1.2 Ý nghĩa của Hệ số Tương Quan:****

* r>0**:** Hai biến có mối quan hệ đồng biến (cùng tăng hoặc cùng giảm).
* r<0**:** Hai biến có mối quan hệ nghịch biến (một biến tăng, biến kia giảm).
* r=0**:** Hai biến không có mối quan hệ tuyến tính.
* **Lưu ý:** Hệ số tương quan chỉ đo mối quan hệ tuyến tính, không áp dụng cho quan hệ phi tuyến tính.

## ****2.2. Biểu đồ Heatmap****

### ****2.2.1 Khái niệm****

* Heatmap là công cụ trực quan hóa dữ liệu, hiển thị các giá trị tương quan giữa các biến trong ma trận tương quan bằng cách sử dụng mã màu.

### ****2.2.2 Cách Hoạt Động****

**Tạo Ma Trận Tương Quan:**

* + Ma trận tương quan là bảng vuông với các hệ số tương quan giữa từng cặp biến trong tập dữ liệu.
  + Ví dụ: Nếu tập dữ liệu có nnn biến, ma trận tương quan sẽ có kích thước n×nn \times nn×n.

**Ánh xạ Hệ Số Tương Quan Thành Màu Sắc:**

* + Mỗi ô trong ma trận tương quan được ánh xạ đến một màu cụ thể dựa trên giá trị của hệ số tương quan:
    - **Màu xanh đậm:** Tương quan dương mạnh (r→+1).
    - **Màu đỏ đậm:** Tương quan âm mạnh (r→−1).
    - **Màu trung tính (trắng/nhạt):** Tương quan gần bằng 0.
  + Thang màu thường sử dụng: xanh - đỏ.

**Hiển Thị Bảng Mã Màu:**

* + Heatmap được vẽ dưới dạng ma trận vuông. Các ô trên đường chéo chính luôn có giá trị tương quan là 1 (một biến tương quan hoàn hảo với chính nó).

### ****2.2.3 Lợi Ích Hoạt Động của Heatmap:****

* **Nhanh chóng nhận biết mối quan hệ:** Nhờ mã màu, bạn có thể ngay lập tức phát hiện biến nào có tương quan cao hoặc thấp.
* **Phát hiện vấn đề đa cộng tuyến:** Nếu có hai biến có hệ số tương quan gần 1 hoặc -1, bạn có thể cân nhắc loại bỏ một trong hai để tránh dư thừa thông tin.
* **Lựa chọn đặc trưng quan trọng:** Giúp xác định biến nào có mối quan hệ chặt chẽ nhất với biến mục tiêu để sử dụng trong mô hình.

## ****2.3. Giá trị phổ biến trong dữ liệu****

### ****2.3.1 Giá trị liên tục (Continuous Values)****

#### ****Khái niệm:****

* Là các giá trị có thể nhận bất kỳ giá trị nào trong một khoảng (số thực).
* **Ví dụ:**
  + RAM (GB), tốc độ xử lý (GHz), dung lượng pin (mAh), độ phân giải màn hình (px\_width, px\_height), thời gian sử dụng pin (giờ), v.v.

#### ****Đặc điểm:****

* Không thể đếm được, mà thường được đo lường.
* Có thể có số thập phân (ví dụ: 3.14 GHz).
* Cần chuẩn hóa (scaling) trước khi đưa vào mô hình học máy.

#### ****Ứng dụng:****

* Thường được sử dụng trong các bài toán dự đoán liên quan đến các biến số liên tục, ví dụ:
  + Dự đoán giá sản phẩm, thời gian thực hiện một công việc, v.v.

### ****2.3.2 Giá trị rời rạc (Discrete Values)****

#### ****Khái niệm:****

* Là các giá trị có thể đếm được, thường là số nguyên.
* **Ví dụ:**
  + Số lõi CPU (2, 4, 8), số camera trên điện thoại, số lượng cổng kết nối, v.v.

#### ****Đặc điểm:****

* Không có giá trị trung gian (không có số thập phân).
* Thường là hữu hạn và có thể đếm được.

#### ****Ứng dụng:****

* Giá trị rời rạc thường được sử dụng trong các bài toán phân loại hoặc thống kê.

### ****2.3.3 Giá trị danh mục (Categorical Values)****

#### ****Khái niệm:****

* Là các giá trị biểu diễn các nhóm hoặc danh mục, không phải số đo lường.
* **Ví dụ:**
  + Loại kết nối (3G, 4G, 5G), có hỗ trợ Wi-Fi không (0/1), màn hình cảm ứng (Touchscreen: Yes/No).

#### ****Đặc điểm:****

* Thường là dạng dữ liệu không số.
* Có thể là nhị phân (binary: 0/1) hoặc đa danh mục (multiclass: A, B, C).
* Cần được **mã hóa** (encoding) trước khi đưa vào mô hình học máy.

#### ****Phương pháp mã hóa:****

**Label Encoding:**

* + Gán mỗi danh mục một số nguyên (0, 1, 2,...).
  + Thích hợp với các thuật toán như cây quyết định.

**One-Hot Encoding:**

* + Tạo cột riêng biệt cho mỗi danh mục và gán giá trị 0 hoặc 1.
  + Thích hợp cho các thuật toán dựa trên khoảng cách, như SVM.

### ****2.3.4 Giá trị nhị phân (Binary Values)****

#### ****Khái niệm:****

* Là tập con của giá trị danh mục, chỉ có hai trạng thái: 0 hoặc 1, Yes hoặc No.
* **Ví dụ:**
  + Có hỗ trợ Wi-Fi (0/1), hỗ trợ Dual SIM (0/1), màn hình cảm ứng (0/1).

#### ****Đặc điểm:****

* Dễ dàng xử lý trong mô hình học máy.
* Không cần mã hóa thêm nếu đã ở dạng nhị phân.

### ****2.3.5 Giá trị ngoại lai (Outliers)****

#### ****Khái niệm:****

* Là các giá trị nằm ngoài phạm vi "bình thường" của dữ liệu.
* Đã trình bày chi tiết ở phần trước.

### ****2.3.6 Giá trị mục tiêu (Target Values)****

#### ****Khái niệm:****

* Là biến mà mô hình học máy cố gắng dự đoán.
* Có thể là:
  + **Liên tục:** Ví dụ: Giá sản phẩm (trong bài toán hồi quy).
  + **Danh mục:** Ví dụ: Phân loại điện thoại (giá rẻ, trung cấp, cao cấp).

### ****2.3.7 So sánh và xử lý các loại giá trị****

| **Loại giá trị** | **Ví dụ** | **Xử lý trước khi huấn luyện** |
| --- | --- | --- |
| **Liên tục** | RAM, pin, px\_height | Chuẩn hóa (StandardScaler) |
| **Rời rạc** | Số lõi CPU, số camera | Có thể để nguyên |
| **Danh mục** | 3G/4G/5G | Mã hóa (One-Hot hoặc Label) |
| **Nhị phân** | Có/Không (0/1) | Không cần xử lý thêm |
| **Ngoại lai** | Giá trị quá lớn/nhỏ | Loại bỏ hoặc biến đổi |

## ****2.4. Biểu đồ Boxplot****

### ****2.4.1 Khái niệm****

* **Boxplot** (hay còn gọi là biểu đồ hộp) là một biểu đồ thống kê giúp tóm tắt phân phối dữ liệu thông qua các số liệu chính:
  + **Giá trị nhỏ nhất (Minimum)**,
  + **Q1 (Phân vị thứ 25)**,
  + **Median (Phân vị thứ 50)**,
  + **Q3 (Phân vị thứ 75)**,
  + **Giá trị lớn nhất (Maximum)**.
* Ngoài ra, Boxplot còn hiển thị các **giá trị ngoại lai** dưới dạng các điểm nằm ngoài khoảng "bình thường" của dữ liệu.

### ****2.4.2 Cấu trúc của Boxplot****

Boxplot gồm các thành phần chính:

**Hộp (Box):**

* + Thể hiện khoảng giá trị giữa Q1 (phân vị thứ 25) và Q3 (phân vị thứ 75).
  + **Chiều dài của hộp (Interquartile Range - IQR):** IQR=Q3−Q1

**Đường ở giữa hộp (Median):**

* + Thể hiện giá trị trung vị (Q2), là giá trị chia đôi dữ liệu.

**Whiskers (râu):**

* + Râu dưới: Điểm nhỏ nhất nằm trong khoảng Q1−1.5×IQR.
  + Râu trên: Điểm lớn nhất nằm trong khoảng Q3+1.5×IQR.

**Outliers (Ngoại lai):**

* + Các điểm nằm ngoài khoảng :

Lower Bound=Q1−1.5×IQR

Upper Bound=Q3+1.5×IQR

* + Những giá trị này được biểu diễn dưới dạng các chấm hoặc dấu sao bên ngoài whiskers.

### ****2.4.3 Ý nghĩa của Boxplot****

#### ****Phân phối dữ liệu****

* **Dải phân vị:** Chiều dài của hộp (IQR) cho biết độ phân tán dữ liệu.
* **Vị trí của Median:**
  + Nếu Median gần Q1: Dữ liệu bị lệch trái.
  + Nếu Median gần Q3: Dữ liệu bị lệch phải.
  + Nếu Median nằm giữa: Dữ liệu phân phối đều.

#### ****Giá trị ngoại lai****

* Các điểm nằm ngoài whiskers là giá trị ngoại lai.
* Boxplot giúp nhanh chóng phát hiện và định lượng các giá trị bất thường.

#### ****So sánh giữa các nhóm****

* Khi Boxplot được vẽ cho các nhóm khác nhau (theo danh mục hoặc lớp), nó cho phép:
  + So sánh phạm vi giá trị giữa các nhóm.
  + Xác định sự khác biệt về trung vị giữa các nhóm.

### ****2.4.4 Hạn chế của Boxplot****

* Boxplot không hiển thị phân phối chi tiết bên trong dữ liệu (ví dụ: dạng hình chuông hay không).
* Khi dữ liệu quá nhiều giá trị ngoại lai, Boxplot có thể khó đọc.
* Không phù hợp để biểu diễn dữ liệu rời rạc.

### ****2.4.5 Kết luận****

* **Boxplot** là công cụ hữu ích để:
  + Tóm tắt dữ liệu.
  + Phát hiện giá trị ngoại lai.
  + So sánh các nhóm dữ liệu.
* Trong bài toán của bạn, Boxplot được dùng để:
  + Kiểm tra phân phối các giá trị liên tục (như RAM, pin, độ phân giải).
  + Phát hiện và xử lý giá trị ngoại lai trước khi chuẩn hóa dữ liệu.
  + So sánh các nhóm **price\_range** để hiểu rõ hơn về đặc điểm của từng nhóm.

## ****2.5. Chuẩn hóa dữ liệu bằng StandardScaler****

### ****2.5.1 Khái niệm chuẩn hóa dữ liệu****

#### ****Định nghĩa:****

* **Chuẩn hóa dữ liệu** là quá trình biến đổi các đặc điểm (features) của dữ liệu sao cho:
  + Có giá trị trung bình bằng 0.
  + Có độ lệch chuẩn bằng 1.

#### ****Tại sao cần chuẩn hóa?****

* Một số thuật toán học máy (như SVM, KNN, PCA) bị ảnh hưởng bởi sự khác biệt về đơn vị và phạm vi giá trị giữa các đặc điểm.
* Nếu không chuẩn hóa, các đặc điểm có phạm vi lớn hơn sẽ chi phối mô hình, làm giảm hiệu suất.

#### ****Công thức chuẩn hóa:****

Mỗi giá trị x được chuẩn hóa theo công thức:



Trong đó:

* z: Giá trị đã chuẩn hóa.
* x: Giá trị gốc.
* : Trung bình của đặc điểm.
* σ: Độ lệch chuẩn của đặc điểm.

### ****2.5.2 StandardScaler****

#### ****Khái niệm:****

* **StandardScaler** là một công cụ trong thư viện scikit-learn được sử dụng để chuẩn hóa dữ liệu.
* Nó áp dụng công thức chuẩn hóa cho từng đặc điểm trong tập dữ liệu, đảm bảo rằng:
  + Trung bình của mỗi đặc điểm = 0.
  + Độ lệch chuẩn của mỗi đặc điểm = 1.

#### ****Cách hoạt động của StandardScaler:****

1. **Tính toán các thông số:**
   * Tính trung bình μ\muμ và độ lệch chuẩn σ cho mỗi đặc điểm trong tập huấn luyện.
2. **Chuẩn hóa dữ liệu:**
   * Áp dụng công thức ​để chuẩn hóa các giá trị.

### ****2.5.3 Ảnh hưởng của chuẩn hóa đến SVM****

#### ****Trước khi chuẩn hóa:****

* Các đặc điểm có phạm vi giá trị lớn (ví dụ: RAM) sẽ chi phối mô hình, làm cho các đặc điểm nhỏ hơn (như px\_height) bị bỏ qua.

#### ****Sau khi chuẩn hóa:****

* Tất cả các đặc điểm được đưa về cùng một phạm vi (trung bình = 0, độ lệch chuẩn = 1).
* Mô hình SVM có thể tính toán khoảng cách và biên phân cách hiệu quả hơn.

#### ****Ví dụ minh họa:****

* Nếu bạn không chuẩn hóa:
  + Biên phân cách của SVM có thể bị lệch theo hướng của đặc điểm có giá trị lớn.
* Nếu bạn chuẩn hóa:
  + Biên phân cách được tối ưu hóa và hiệu suất dự đoán tăng lên.

### ****2.5.4 Khi nào không cần chuẩn hóa?****

* Nếu bạn sử dụng mô hình không nhạy cảm với đơn vị và phạm vi giá trị, như cây quyết định hoặc random forest, thì chuẩn hóa không bắt buộc.

### ****2.5.5 Kết luận****

* Chuẩn hóa dữ liệu bằng StandardScaler là bước bắt buộc khi sử dụng mô hình SVM.
* Nó giúp mô hình hoạt động hiệu quả hơn bằng cách đưa các đặc điểm về cùng một phạm vi.
* Trong bài toán của bạn, chuẩn hóa sẽ cải thiện đáng kể hiệu suất phân loại giá điện thoại.

**2.6 Khái niệm về SVM (Support Vector Machine)**

Support Vector Machine (SVM) là một thuật toán học máy giám sát được sử dụng chủ yếu cho các bài toán phân loại. SVM tìm kiếm một siêu phẳng (hyperplane) trong không gian đặc trưng sao cho nó có thể phân chia các lớp dữ liệu một cách tối ưu. Mục tiêu của SVM là tối đa hóa khoảng cách (margin) giữa các điểm dữ liệu của các lớp khác nhau, với mục đích làm giảm sai số phân loại.

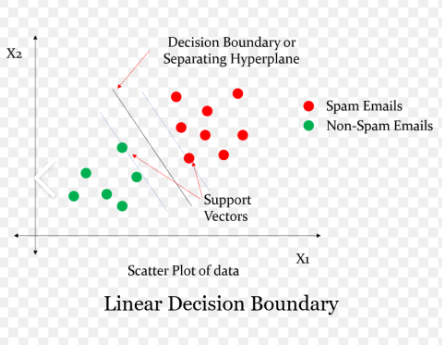
**2.6.1 Cấu trúc của SVM:**

* **Siêu phẳng (Hyperplane)**: Là một mặt phẳng trong không gian đặc trưng có thể phân chia các lớp dữ liệu.
* **Margin**: Khoảng cách giữa siêu phẳng và các điểm dữ liệu gần nhất của mỗi lớp.
* **Support Vectors**: Các điểm dữ liệu nằm gần siêu phẳng, quyết định vị trí của siêu phẳng và ảnh hưởng đến quá trình huấn luyện của SVM.

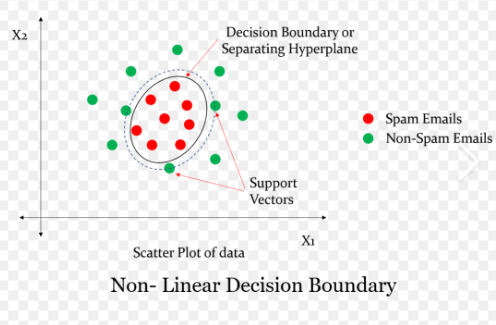
****2.6.2 Tìm hiểu cách hoạt động của SVM**:**

**+ Đầu tiên SVM có 2 trường hợp bài toán:**

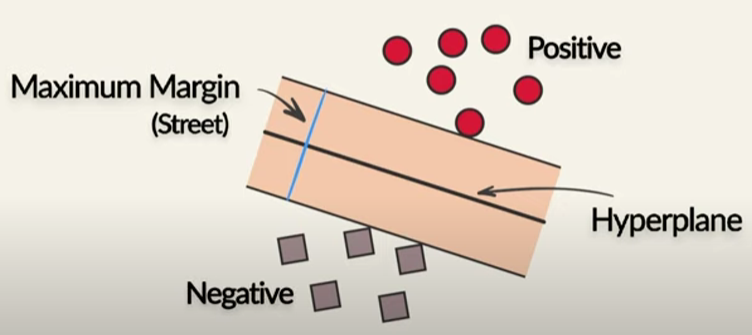
1. **Phân loại tuyến tính**: Nếu các lớp dữ liệu có thể phân chia một cách rõ ràng bằng một siêu phẳng, SVM sẽ tìm kiếm siêu phẳng đó sao cho khoảng cách giữa các lớp là lớn nhất.



1. **Phân loại phi tuyến tính**: Trong trường hợp dữ liệu không thể phân chia một cách rõ ràng bằng một siêu phẳng, SVM sẽ sử dụng kỹ thuật "hạt nhân" (kernel trick) để biến đổi không gian đặc trưng sao cho các lớp dữ liệu có thể phân chia được bằng một siêu phẳng trong không gian cao hơn.

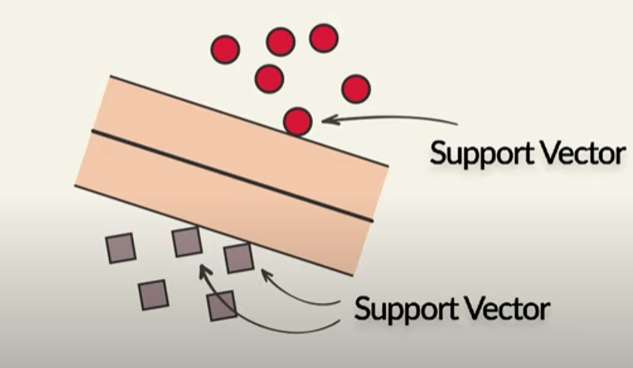


+ Tổng thể thì có rất nhiều cách để vẽ siêu phẳng và SVM sẽ giúp chúng ta chọn được siêu phẳng nào là tốt nhất.



+ Trên đây là 1 ví dụ về biểu đồ :

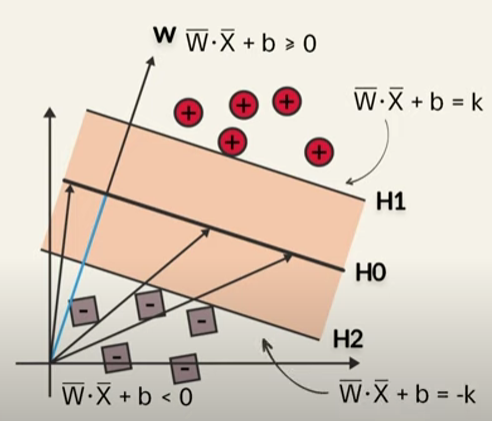
* Có 2 điểm dương và âm thể hiện bằng chấm đỏ và vuông xám
* Siêu phẳng ( hyperplane ) là đường màu đen cũng như là đường quyết định phân chia giữa 2 lớp dương và âm. Nó được quyết định bằng tối ưu hóa tối đa khoảng cách margin giữa các điểm dữ liệu gần nhất của 2 lớp
* Khoảng cách ( Margin ) là khoảng cách giữa siêu phẳng và các điểm dữ liệu gần nhất từ 2 lớp này.
* Mục tiêu của mô hình SVM là tìm siêu phẳng sao cho margin này là lớn nhất giúp mô hình phân loại các dữ liệu 1 cách hiệu quả hơn.



+ Tiếp theo hãy đến với khái niệm Support vector:

* Vector hỗ trợ là các điểm dữ liệu quan trọng đóng vai trò then chốt trong việc xác định siêu phẳng
* Nó là những điểm gần nhất so với siêu phẳng ( Hyperplane ).
* Các support vector là những điểm dữ liệu duy nhất mà SVM sử dụng để xác định siêu phẳng quyết định. Các điểm dữ liệu khác không ảnh hưởng đến vị trí của siêu phẳng. Có thể hiểu rằng di chuyển support vector thì vị trí của siêu phẳng sẽ thay đổi, việc di chuyển các vector khác ngoài support vector sẽ không làm thay đổi siêu phẳng.
* **Tối đa hóa khoảng cách (margin)**: SVM cố gắng tối đa hóa khoảng cách giữa siêu phẳng quyết định và các support vector, tạo ra một ranh giới phân chia rõ ràng giữa các lớp.

+ Giống với các mô hình học máy khác, SVM sẽ giúp chúng ta tìm siêu phẳng với công thức WX + b = y ( trong đó W là vector trọng số và b là bias )



* Ở hình trên nếu  sẽ được phân vào nữa trên ( giá trị dương )
* Còn  sẽ được phân vào nữa dưới ( giá trị âm )
* Các giá trị H1 và H2 lần lượt được tính theo công thức  và 

+ Vậy để tối ưu hóa tìm vector trọng số W và hệ số bias sao cho khoảng cách giữa các siêu phẳng và các support vector được tối đa hóa.

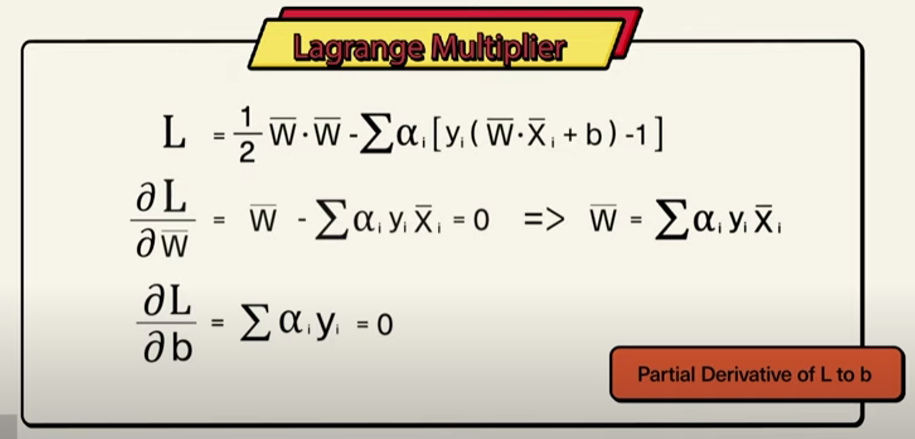
* Hàm mục tiêu cần tối thiểu hóa là: 
* Đây là nửa bình phương của độ lớn vector trọng số, giúp tối đa hóa margin

+ Các rằng buộc sẽ đảm bảo các dữ liệu được phân loại đúng +

+ Các điểm dữ liệu thỏa mãn ràng buộc với dấu bằng là các support vector, tức là: 

+ Các support vector nằm trên hoặc rất gần các đường biên (margin). Trên các đường biên này, hàm quyết định của SVM sẽ là: 

+ **Từ điều kiện gradient của Ràng Buộc Lagrange sẽ tìm ra được:**



****2.6.3 Các loại hạt nhân trong SVM**:**

* **Linear Kernel**: Phân chia dữ liệu trong không gian ban đầu mà không cần biến đổi.
* **Polynomial Kernel**: Dùng để tạo ra các siêu phẳng phân chia trong không gian đặc trưng phi tuyến tính.
* **Radial Basis Function (RBF) Kernel**: Dùng cho các bài toán phân loại phức tạp, biến đổi không gian dữ liệu sao cho các lớp có thể phân chia được.

**2.7 Khái niệm về OVO (One-vs-One) trong SVM**

Khi SVM được áp dụng cho các bài toán phân loại đa lớp (multiclass), một vấn đề phát sinh là SVM chỉ có thể phân loại hai lớp tại một thời điểm. Vì vậy, để áp dụng SVM cho các bài toán phân loại nhiều lớp, có một số chiến lược khác nhau. Một trong những chiến lược phổ biến là **One-vs-One (OVO)**.

**2.7.1 One-vs-One (OVO)**

Là phương pháp phân loại đa lớp trong đó mỗi cặp lớp sẽ được phân loại riêng biệt. Ví dụ, với một bài toán có 3 lớp (A, B, C), OVO sẽ xây dựng 3 mô hình phân loại SVM theo công thức n(n−1)/2 (3(3-1)/2=3)**:**

* Một mô hình phân loại giữa lớp A và lớp B.
* Một mô hình phân loại giữa lớp A và lớp C.
* Một mô hình phân loại giữa lớp B và lớp C.

Mỗi mô hình sẽ được huấn luyện trên một cặp lớp cụ thể và đưa ra dự đoán giữa hai lớp đó. Sau khi có các dự đoán từ tất cả các mô hình, kết quả cuối cùng sẽ được quyết định bằng cách sử dụng một phương pháp bỏ phiếu (voting). Cụ thể, lớp nào nhận được nhiều phiếu nhất từ các mô hình phân loại sẽ được chọn là lớp dự đoán.

**2.7.2 Cách hoạt động của OVO**:

1. **Huấn luyện mô hình**: Với mỗi cặp lớp, một mô hình SVM được huấn luyện để phân loại hai lớp đó.
2. **Dự đoán**: Khi một điểm dữ liệu mới cần phân loại, mỗi mô hình phân loại sẽ đưa ra một kết quả dự đoán giữa hai lớp mà nó được huấn luyện.
3. **Bỏ phiếu**: Kết quả cuối cùng được xác định dựa trên kết quả bỏ phiếu từ tất cả các mô hình SVM. Lớp nhận được nhiều phiếu nhất sẽ là lớp dự đoán cho điểm dữ liệu.

****2.7.3 Ưu và nhược điểm của OVO**:**

* **Ưu điểm**:
  + Đảm bảo phân loại chính xác hơn trong các bài toán phân loại đa lớp vì mỗi mô hình chỉ phải phân loại giữa hai lớp.
  + Dễ dàng áp dụng với các thuật toán phân loại nhị phân như SVM.
* **Nhược điểm**:
  + Khi số lớp tăng lên, số lượng mô hình SVM cần huấn luyện tăng theo công thức n(n−1)/2, với n là số lớp. Điều này có thể dẫn đến chi phí tính toán rất lớn.
  + Việc kết hợp các mô hình có thể phức tạp và yêu cầu một phương pháp bỏ phiếu hiệu quả.

****2.7.4 Ứng dụng của SVM kết hợp OVO**:**

* **Phân loại văn bản**: SVM OVO có thể được sử dụng trong các bài toán phân loại văn bản, ví dụ như phân loại email spam, phân loại tin tức, v.v.
* **Nhận dạng hình ảnh**: SVM OVO có thể được áp dụng trong các hệ thống nhận dạng hình ảnh, chẳng hạn như nhận diện khuôn mặt, nhận diện chữ viết tay, v.v.
* **Phân tích dữ liệu y tế**: SVM OVO có thể được sử dụng để phân loại các bệnh lý từ dữ liệu y tế, ví dụ như phân loại các loại ung thư dựa trên đặc trưng tế bào.

**Tóm tắt**: SVM là một thuật toán phân loại mạnh mẽ, có khả năng phân chia các lớp dữ liệu tối ưu trong không gian đặc trưng. Khi áp dụng cho bài toán phân loại đa lớp, phương pháp One-vs-One (OVO) sẽ tạo ra nhiều mô hình phân loại nhị phân, mỗi mô hình xử lý một cặp lớp và kết quả cuối cùng được xác định bằng cách bỏ phiếu. Mặc dù OVO có thể yêu cầu tài nguyên tính toán lớn, nó mang lại độ chính xác cao trong các bài toán phân loại đa lớp.

**8. IQR( **Interquartile Range )****

**2.8.1 Khái niệm về IQR (Interquartile Range):**

+ IQR (Interquartile Range) là một chỉ số thống kê dùng để đo độ phân tán của dữ liệu, đặc biệt là trong việc xác định các giá trị ngoại lai (outliers). IQR được tính bằng hiệu giữa phân vị thứ 75 (Q3) và phân vị thứ 25 (Q1). IQR là một cách thức đơn giản nhưng hiệu quả để hiểu được sự phân bố của một bộ dữ liệu mà không bị ảnh hưởng bởi các giá trị cực trị (outliers).

Công thức tính IQR:

IQR=Q3−Q1

* **Q1** (Phân vị thứ 25): Là giá trị ở vị trí thứ 25% trong bộ dữ liệu khi đã được sắp xếp theo thứ tự tăng dần.
* **Q3** (Phân vị thứ 75): Là giá trị ở vị trí thứ 75% trong bộ dữ liệu khi đã được sắp xếp theo thứ tự tăng dần.

**2.8.2 Nguyên lý hoạt động của IQR:**

**+**  IQR giúp đánh giá sự phân tán của dữ liệu, tức là phạm vi giữa 25% dữ liệu nhỏ nhất (Q1) và 25% dữ liệu lớn nhất (Q3). Nó chỉ ra phần lớn dữ liệu nằm trong phạm vi này. Những giá trị nằm ngoài phạm vi này sẽ được coi là **ngoại lai (outliers)**.

**Cách xác định giá trị ngoại lai bằng IQR:**

* **Giới hạn dưới (Lower Bound)**: Bất kỳ giá trị nào nhỏ hơn Q1−1.5×IQR sẽ được coi là giá trị ngoại lai.

Lower Bound=Q1−1.5×IQR

* **Giới hạn trên (Upper Bound)**: Bất kỳ giá trị nào lớn hơn Q3+1.5×IQR sẽ được coi là giá trị ngoại lai.

Upper Bound=Q3+1.5×IQR

Những giá trị nằm ngoài phạm vi từ Lower Bound đến Upper Bound sẽ được xem là **ngoại lai**. Ví dụ:

* Nếu giá trị nhỏ hơn Lower Bound hoặc lớn hơn Upper Bound, chúng sẽ được đánh dấu là ngoại lai.

**2.8.3 Tầm quan trọng và ứng dụng của IQR:**

* **Phát hiện ngoại lai**: IQR giúp xác định những giá trị không bình thường hoặc bất thường trong dữ liệu mà có thể ảnh hưởng đến quá trình phân tích và mô hình hóa.
* **Đánh giá độ phân tán**: IQR cho phép bạn biết được sự phân tán của các giá trị trong dữ liệu. Một IQR lớn cho thấy sự phân tán rộng rãi của dữ liệu, trong khi một IQR nhỏ cho thấy sự phân tán chặt chẽ hơn.
* **Ổn định với ngoại lai**: Khác với phương pháp sử dụng độ lệch chuẩn, IQR không bị ảnh hưởng nhiều bởi các giá trị ngoại lai, vì vậy nó là một công cụ hữu ích trong việc làm sạch dữ liệu.

**2.8.4 Ví dụ minh họa:** Giả sử bạn có một bộ dữ liệu gồm các giá trị:

[1,2,3,4,5,6,7,8,9,10][1, 2, 3, 4, 5, 6, 7, 8, 9, 10][1,2,3,4,5,6,7,8,9,10]

* **Q1** = 3 (phân vị thứ 25)
* **Q3** = 8 (phân vị thứ 75)
* **IQR** = Q3 - Q1 = 8 - 3 = 5

Giới hạn dưới và giới hạn trên sẽ được tính như sau:

* **Lower Bound** = 3 - 1.5 \* 5 = -4.5 (không có giá trị nhỏ hơn -4.5)
* **Upper Bound** = 8 + 1.5 \* 5 = 15.5 (không có giá trị lớn hơn 15.5)

Vì không có giá trị nào nằm ngoài phạm vi này, bộ dữ liệu này không có ngoại lai.

Tuy nhiên, nếu có giá trị như 20 trong bộ dữ liệu, thì 20 sẽ là một ngoại lai vì nó vượt quá giới hạn trên (15.5).

# CHƯƠNG 3: ĐÁNH GIÁ MÔ HÌNH VÀ KẾT LUẬN

Phân tích chi tiết bài toán.

3.1 Tiền xử lý dữ liệu

* Vẽ biểu đồ boxplot, và heatmap:

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Đọc file dữ liệu từ đường dẫn được cung cấp

data = pd.read\_csv('data\_cleaned.csv')

# Tính ma trận tương quan

correlation\_matrix = data.corr(numeric\_only=True)

# Vẽ heatmap của ma trận tương quan

plt.figure(figsize=(12, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm", fmt=".2f", cbar=True)

plt.title("Correlation Heatmap")

plt.savefig('heatmap.png',dpi=300, bbox\_inches='tight')

plt.show()

# Vẽ boxplot cho tất cả các cột số trong dữ liệu

plt.figure(figsize=(14, 8))

sns.boxplot(data=data.select\_dtypes(include=["float64", "int64"]))

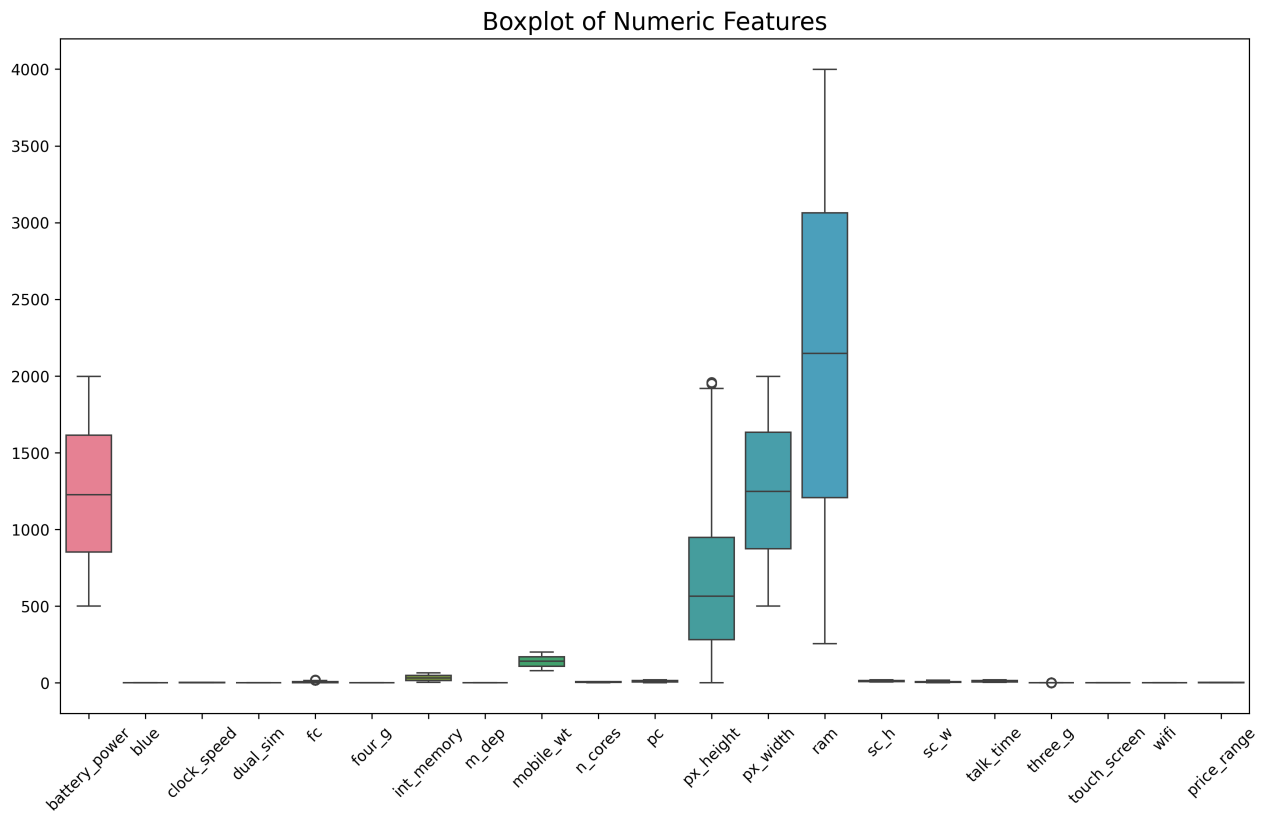
plt.title("Boxplot of Numeric Features", fontsize=16)

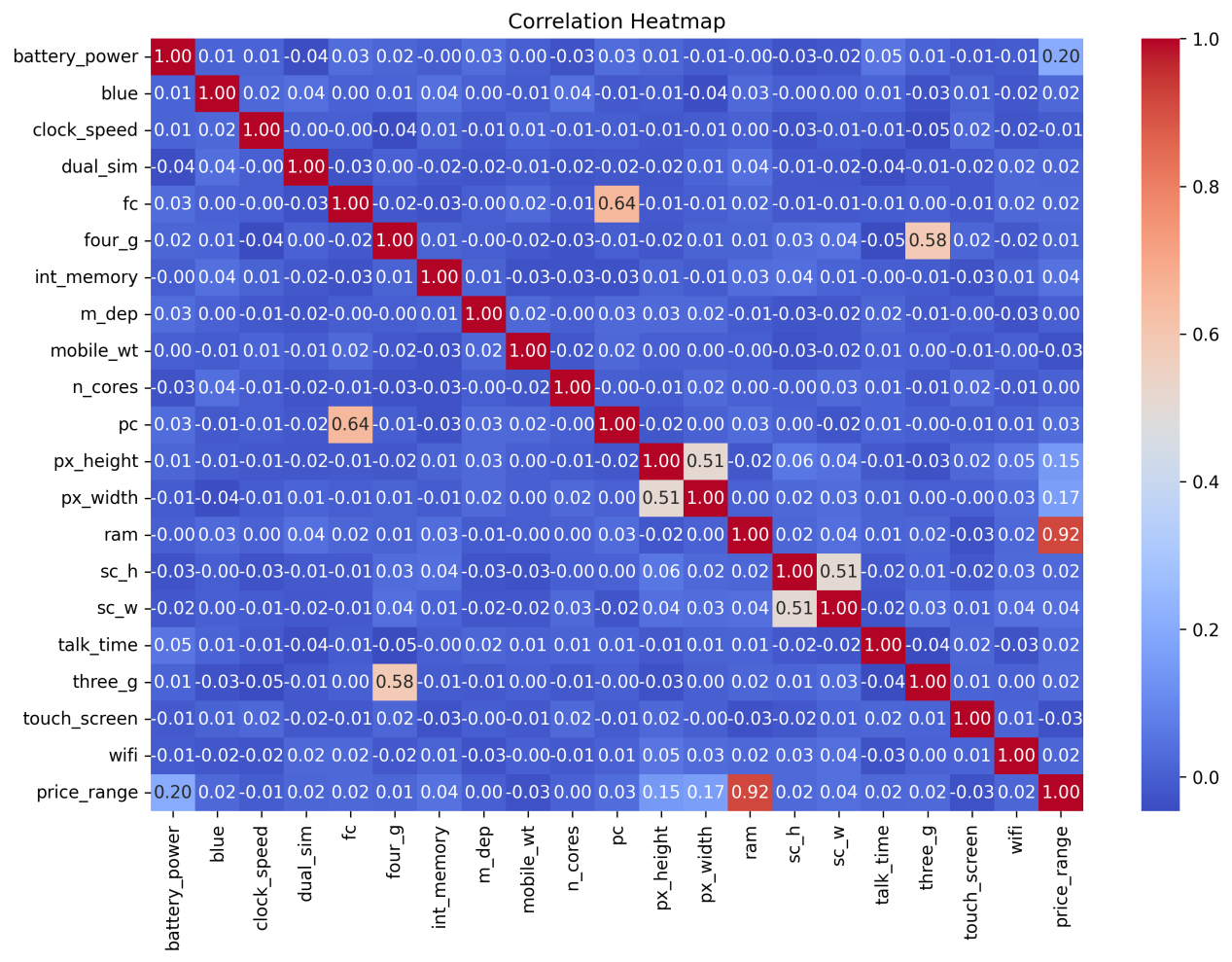
plt.xticks(rotation=45)

plt.savefig('boxplot.png',dpi=300, bbox\_inches='tight')

plt.show()

Kết quả: cho ra 2 biểu đồ heatmap và boxplot





+ Với 2 biểu đô trên chúng ta kết luận được:

!!! Vì mô hình SVM rất nhạy cảm với các giá trị liên tục nên:

* Data có mội vài giá trị ngoại lai cần phải sử lý
* Các giá trị liên tục nên được chuẩn hóa
* Biểu đồ heatmap cho thấy mối tương quan cao nhất giữa các thuộc tính là 3g với 4g ( 58% )

=> Data hiện tại các tương quan khá thấp nên hoàn toàn không cần phải giảm chiều ( PCA ) vì nếu giảm sẽ giảm hiệu suất mô hình.

+ Xử lý giá trị ngoại lai bằng IQR:

import pandas as pd

df = pd.read\_csv('train.csv')

# Lọc các cột kiểu số, nhưng bỏ qua cột dạng binary (có giá trị 0 hoặc 1)

numeric\_cols = df.select\_dtypes(include=['float64','int64']).columns

# Lọc ra các cột dạng binary (giá trị chỉ có 0 và 1)

binary\_cols = [col for col in numeric\_cols if df[col].nunique() == 2]

# Lấy các cột liên tục (loại bỏ binary cols)

continuous\_cols = [col for col in numeric\_cols if col not in binary\_cols]

# Tạo một bản sao của DataFrame

df\_cleaned = df.copy()

# Xử lý ngoại lai cho các cột liên tục

for col in continuous\_cols:

    # Tính Q1, Q3 và IQR

    Q1 = df\_cleaned[col].quantile(0.25)

    Q3 = df\_cleaned[col].quantile(0.75)

    IQR = Q3 - Q1

Sau khi thay thế giá trị ngoại lai ta thu được lại data\_cleaned sẵn sàng đưa vào mô hình.

# Ngưỡng giới hạn ngoại lai

    lower\_bound = Q1 - 1.5 \* IQR

    upper\_bound = Q3 + 1.5 \* IQR

    # Thay thế giá trị ngoại lai nhỏ hơn lower\_bound và lớn hơn upper\_bound

    df\_cleaned[col] = df\_cleaned[col].apply(lambda x: lower\_bound if x < lower\_bound else x)

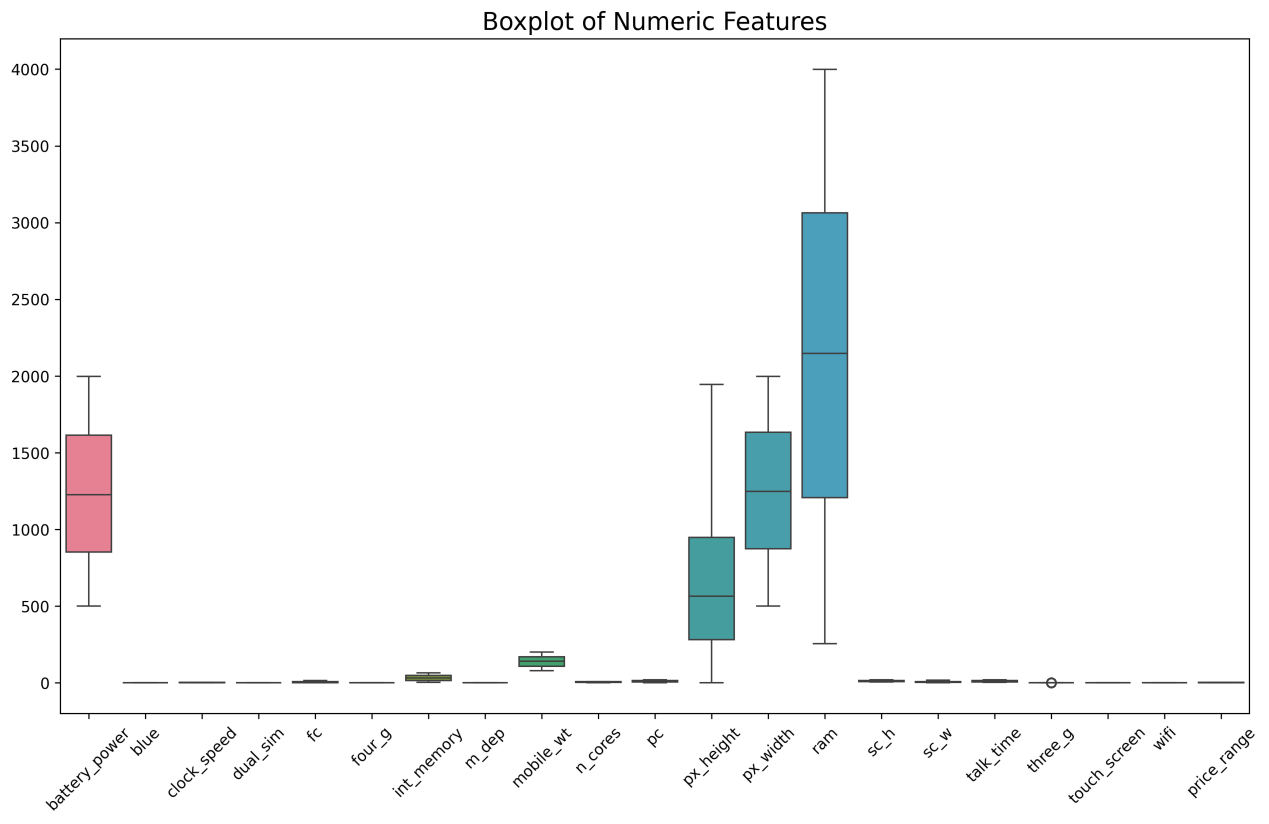
    df\_cleaned[col] = df\_cleaned[col].apply(lambda x: upper\_bound if x > upper\_bound else x)

# Xuất ra file CSV mới

df\_cleaned.to\_csv('data\_cleaned.csv', index=False)

print("Dữ liệu đã được xử lý và lưu vào file 'data\_cleaned.csv'.")

Biểu đồ boxplot sau khi đã xóa giá trị ngoại lại:



Bước tiếp theo là chuẩn hóa lại các dữ liệu liên tục:

import pandas as pd

from sklearn.preprocessing import StandardScaler

# Đọc dữ liệu từ file CSV

file\_path = 'data/test.csv'

data = pd.read\_csv(file\_path)

# Các cột liên tục mà bạn muốn chuẩn hóa

continuous\_columns = ['battery\_power', 'clock\_speed', 'fc', 'int\_memory', 'm\_dep', 'mobile\_wt',

                    'n\_cores', 'pc', 'px\_height', 'px\_width', 'ram', 'sc\_h', 'sc\_w', 'talk\_time']

# Tạo đối tượng chuẩn hóa

scaler = StandardScaler()

# Chuẩn hóa các cột liên tục đã xác định

data[continuous\_columns] = scaler.fit\_transform(data[continuous\_columns])

# Lưu dữ liệu đã chuẩn hóa vào file mới

standardized\_file\_path = 'data/test\_standardized.csv'

data.to\_csv(standardized\_file\_path, index=False)

# Hiển thị vài dòng đầu của dữ liệu đã chuẩn hóa

print(data.head())

+ Sau khi chuẩn hóa xong các cột được chuẩn hóa sẽ thay đổi giá trị với trung bình = 0 và độ lệch chuẩn = 1.

Tiếp theo chúng ta sẽ cho data vào mô hình để bắt đầu huấn luyện:

import matplotlib

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from itertools import combinations

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, accuracy\_score

matplotlib.use('TkAgg')

Thêm thư viện:

+ matplotlib: Vẽ biểu đồ, trực quan hóa dữ liệu.

+ numpy: Thực hiện các phép toán ma trận và số học. Đây là thư viện cốt lõi cho việc xử lý mảng, tính toán các giá trị liên quan đến trọng số và bias trong mô hình.

+ pandas: Quản lý và xử lý dữ liệu trong các cấu trúc bảng (DataFrame). Đây là thư viện dùng để làm việc với dữ liệu có cấu trúc như CSV hoặc các bảng dữ liệu.

+ itertools.combinations: Tạo ra tất cả các tổ hợp có thể của các phần tử từ một tập hợp.

+ sklearn.model\_selection.train\_test\_split: Chia dữ liệu thành các tập huấn luyện và kiểm tra.

+ sklearn.metrics.classification\_report và accuracy\_score: Đánh giá hiệu suất của mô hình phân loại.

 def \_\_init\_\_(self, C=0.1, max\_iter=20000,learning\_rate=0.001,

              class\_weights=None,momentum=0.9,patience=10):

        self.C = C  # Hệ số điều chỉnh (regularization)

        self.max\_iter = max\_iter  # Số lần lặp tối đa

        self.learning\_rate = learning\_rate  # Tốc độ học ban đầu

        self.momentum = momentum  # Hệ số momentum

        self.patience = patience  # Số epoch chờ trước khi early stopping

        self.models = {}  # Lưu trữ các mô hình cho từng cặp classes

        self.class\_weights = class\_weights  # Trọng số cho các class

Hàm init là hàm khởi tạo của lớp, được gọi khi bạn tạo một đối tượng của lớp này. Mục đích của hàm này là khởi tạo các tham số và cấu hình ban đầu cho mô hình One-vs-One Support Vector Machine (OvO SVM)

def \_adjust\_learning\_rate(self, epoch):

        """Điều chỉnh learning rate theo epoch"""

        return self.learning\_rate / (1 + epoch \* 0.01)

    def \_initialize\_weights(self, n\_features):

        """Khởi tạo trọng số theo phân phối normal với scale nhỏ"""

        return np.random.normal(0, 0.01, n\_features)

def \_compute\_loss(self, X, y, w, b):

        """Tính hinge loss và regularization loss"""

        margin = y \* (np.dot(X, w) + b)

        hinge\_loss = np.maximum(0, 1 - margin).mean()

        reg\_loss = 0.5 \* self.C \* np.dot(w, w)

        return hinge\_loss + reg\_loss

Hàm \_compute\_loss tính toán tổng tổn thất (loss) trong mô hình SVM, bao gồm hai phần chính: **hinge loss** và **regularization loss**.

+ tính margin của mỗi mẫu dữ liệu. Trong SVM, mỗi điểm dữ liệu sẽ có một giá trị margin, là sản phẩm giữa nhãn (label) của điểm dữ liệu y và giá trị dự đoán từ mô hình, được tính bởi công thức np.dot(X, w) + b.

+ y là nhãn của mẫu, có thể là +1 hoặc -1 trong SVM. Nếu y là +1, thì margin càng lớn càng tốt (dự đoán của mô hình càng đúng), và ngược lại nếu y là -1. Việc nhân với y giúp phân biệt các mẫu thuộc các lớp khác nhau.

**Hinge loss**, một hàm mất mát phổ biến trong SVM. Hinge loss có giá trị nhỏ khi mẫu được phân loại đúng và có giá trị lớn khi mẫu bị phân loại sai hoặc gần biên phân cách (decision boundary).

* Nếu margin ≥ 1, nghĩa là mẫu được phân loại đúng và cách xa decision boundary, nên hinge loss bằng 0.
* Nếu margin < 1, nghĩa là mẫu gần hoặc sai quyết định, nên hinge loss tăng lên.

**Regularization loss**, được tính theo công thức chuẩn trong SVM. Regularization loss giúp kiểm soát độ phức tạp của mô hình, ngăn mô hình học quá mức (overfitting) bằng cách hạn chế giá trị của các trọng số w.

* np.dot(w, w) tính bình phương của các trọng số, hay còn gọi là ||w||^2, giúp phạt các trọng số lớn.
* self.C: Hệ số điều chỉnh (regularization parameter). Nếu C lớn, mô hình sẽ ít bị phạt về trọng số và cố gắng phân loại chính xác hơn (dễ bị overfitting). Nếu C nhỏ, mô hình sẽ phạt trọng số nhiều hơn và cho phép có nhiều lỗi hơn trong quá trình huấn luyện.
* Hệ số 0.5 giúp làm giảm ảnh hưởng của regularization loss trong quá trình tối ưu hóa (do SVM tối thiểu hóa tổng loss).

def \_train\_binary\_svm(self, X, y):

        """Train SVM nhị phân với các cải tiến"""

        n\_samples, n\_features = X.shape

        # Tách validation set

        X\_train, X\_val, y\_train, y\_val = train\_test\_split(

            X, y, test\_size=0.2, random\_state=42)

        # Khởi tạo trọng số và bias

        w = self.\_initialize\_weights(n\_features)

        b = 0

Hàm \_train\_binary\_svm trong đoạn mã này thực hiện việc huấn luyện một mô hình SVM nhị phân (chỉ phân loại giữa hai lớp) với các cải tiến như **momentum**, **early stopping**, và **learning rate schedule**.

+ đầu tiên gắn n\_samples: Số lượng mẫu (số dòng trong X) n\_features: Số lượng đặc trưng (số cột trong X).

+ tiếp theo dữ liệu huấn luyện X\_train, y\_train và dữ liệu kiểm tra (validation) X\_val, y\_val được tách từ dữ liệu ban đầu, với 80% dữ liệu được dùng để huấn luyện và 20% dùng để kiểm tra mô hình trong quá trình huấn luyện (validation).

+ Trọng số w được khởi tạo ngẫu nhiên (có thể theo phân phối chuẩn) thông qua hàm \_initialize\_weights. Độ chệch b được khởi tạo bằng 0.

        # Khởi tạo các biến cho momentum và early stopping

        prev\_v\_w = np.zeros\_like(w)

        prev\_v\_b = 0

        best\_val\_loss = float('inf')

        patience\_counter = 0

# Điều chỉnh trọng số cho lớp dữ liệu

        if self.class\_weights:

            weights = np.array([

                self.class\_weights[1 if yi == 1 else 0] for yi in y\_train

            ])

        else:

            weights = np.ones\_like(y\_train)

        for epoch in range(self.max\_iter):

            # Điều chỉnh learning rate

            curr\_lr = self.\_adjust\_learning\_rate(epoch)

            # Training step với momentum

            for idx, x\_i in enumerate(X\_train):

                condition = y\_train[idx] \* (np.dot(x\_i, w) + b) >= 1

                if condition:

                    v\_w = self.momentum \* prev\_v\_w - curr\_lr \* (2 \* self.C \* w)

                    v\_b = self.momentum \* prev\_v\_b

                else:

                    v\_w = (self.momentum \* prev\_v\_w -

                           curr\_lr \* (2 \* self.C \* w - np.dot(x\_i, y\_train[idx])))

                    v\_b = self.momentum \* prev\_v\_b + curr\_lr \* y\_train[idx]

                # Cập nhật trọng số với momentum

                w += v\_w \* weights[idx]

                b += v\_b \* weights[idx]

                prev\_v\_w, prev\_v\_b = v\_w, v\_b

+ Vòng lặp if đầu tiên :

* Nếu self.class\_weights được cung cấp, thì sẽ tạo một mảng weights mà mỗi giá trị trọng số tương ứng với lớp của mẫu trong y\_train.
* Nếu lớp của mẫu là 1, trọng số là self.class\_weights[1].
* Nếu lớp của mẫu là 0, trọng số là self.class\_weights[0].
* Nếu không có class\_weights, thì mọi mẫu trong y\_train sẽ có trọng số bằng 1 (mảng weights sẽ có giá trị là 1 cho tất cả các phần tử).

+ Vòng lặp for tiếp theo là vòng lặp huấn luyện epoch loop:

* Thực hiện huấn luyện qua nhiều epoch (số vòng lặp) với số lượng tối đa self.max\_iter.
* Tiếp theo vòng lặp for sẽ Duyệt qua từng mẫu x\_i trong tập huấn luyện X\_train và tính toán xem mẫu đó có được phân loại đúng hay không.
* Tiếp theo là **Cập nhật trọng số và bias với Momentum.**

            # Kiểm tra early stopping trên validation set

            val\_loss = self.\_compute\_loss(X\_val, y\_val, w, b)

            if val\_loss < best\_val\_loss:

                best\_val\_loss = val\_loss

                patience\_counter = 0

                best\_w, best\_b = w.copy(), b

            else:

                patience\_counter += 1

                if patience\_counter >= self.patience:

                    print(f"Early stopping at epoch {epoch}")

                    return best\_w, best\_b

        return w, b

+ Kiểm tra early stopping trên validation set:

* Tính toán giá trị mất mát (loss) trên tập validation X\_val và y\_val tại thời điểm hiện tại của trọng số w và độ chệch b.
* Lưu lại trạng thái mô hình tốt nhất (trọng số w và độ chệch b) nếu mất mát trên validation giảm xuống.
* Nếu không có sự cải thiện nào về val\_loss, tăng biến đếm patience\_counter. Nếu số vòng lặp không có cải thiện đạt tới ngưỡng self.patience, ngừng huấn luyện.
* Trả về w,b.

def fit(self, X, y):

        """Train mô hình OvO SVM với monitoring"""

        self.classes = np.unique(y)

        class\_pairs = list(combinations(self.classes, 2))

        print("Bắt đầu training các classifier:")

        for class\_1, class\_2 in class\_pairs:

            print(f"\nTraining classifier cho classes {class\_1} vs {class\_2}")

            # Lọc data cho cặp class hiện tại

            mask = (y == class\_1) | (y == class\_2)

            X\_pair = X[mask]

            y\_pair = np.where(y[mask] == class\_1, 1, -1)

            # Train và lưu model

            w, b = self.\_train\_binary\_svm(X\_pair, y\_pair)

            self.models[(class\_1, class\_2)] = (w, b)

            print(f"Đã hoàn thành training classifier {class\_1} vs {class\_2}")

       return self

+ thực hiện huấn luyện một **One-vs-One (OvO) SVM**

* Khởi tạo các lớp và cặp lớp self.classes, class\_pairs
* for class\_1, class\_2 in class\_pairs:

print(f"\nTraining classifier cho classes {class\_1} vs {class\_2}") để huấn luyện một bộ phân loại cho mỗi cặp lớp.

* **Lọc dữ liệu cho cặp lớp hiện tại**:

mask: Lọc các chỉ số của các mẫu dữ liệu mà lớp y thuộc một trong hai lớp trong cặp class\_1 hoặc class\_2. mask là một mảng boolean.

X\_pair: Chỉ lấy các mẫu dữ liệu trong X mà có lớp là class\_1 hoặc class\_2.

y\_pair: Gán nhãn cho các mẫu trong X\_pair. Nếu lớp là class\_1, nhãn sẽ là 1, còn nếu là class\_2, nhãn sẽ là -1. Đây là cách áp dụng SVM cho phân loại nhị phân.

* Huấn luyện SVM cho cặp lớp và lưu mô hình

self.\_train\_binary\_svm(X\_pair, y\_pair) gọi phương thức để huấn luyện

self.models[(class\_1, class\_2)] = (w, b) lưu mô hình vừa huấn luyện

def predict(self, X):

        """Dự đoán lớp của mẫu dữ liệu"""

        votes = np.zeros((X.shape[0], len(self.classes)))

        for (class\_1, class\_2), (w, b) in self.models.items():

            decision = np.dot(X, w) + b

            votes[:, class\_1] += (decision > 0).astype(int)

            votes[:, class\_2] += (decision <= 0).astype(int)

        return np.argmax(votes, axis=1)

Hàm thực hiện phương thức predict cho mô hình ovo svm giúp dự đoán lớp của các mẫu dữ liệu.

* Khởi tạo vote ( X.shape[0]: Số mẫu trong tập dữ liệu , len(self.classes): Số lớp trong mô hình)
* Lặp qua các mô hình One-vs-One
* Dự đoán mô hình hiện tại decision = np.dot(X, w) + b
* Cập nhật phiếu bầu cho mỗi lớp
* Dự đoán lớp cuối cùng cho mỗi mẫu.

Hàm main thực hiện các bước sau:

1. **Tải và kiểm tra dữ liệu huấn luyện**: Đọc dữ liệu từ tệp CSV và kiểm tra phân phối các lớp trong tập huấn luyện.
2. **Tách features và labels**: Tách dữ liệu đầu vào (features) và nhãn (labels).
3. **Chia dữ liệu**: Chia dữ liệu thành tập huấn luyện và kiểm tra (80-20%).
4. **Tính trọng số cho các lớp**: Tính trọng số nghịch đảo cho mỗi lớp để cân bằng dữ liệu.
5. **Huấn luyện mô hình**: Khởi tạo và huấn luyện mô hình **One-vs-One SVM**.
6. **Đánh giá mô hình**: Dự đoán và tính toán độ chính xác, báo cáo chi tiết trên tập kiểm tra.
7. **Vẽ kết quả**: Vẽ đồ thị phân phối dữ liệu cho mỗi cặp lớp.
8. **Lưu mô hình**: Lưu các mô hình đã huấn luyện vào tệp.

Hàm main thực hiện các bước sau:

if \_\_name\_\_ == "\_\_main\_\_":

    # Load và kiểm tra dữ liệu training

    train\_path = 'data/train\_standardized.csv'

    train\_data = pd.read\_csv(train\_path)

    print("Phân phối các classes trong tập training:")

    print(train\_data['price\_range'].value\_counts())

    # Tách features và labels

    X = train\_data.drop(columns=['price\_range']).values

    y = train\_data['price\_range'].values

    # Chia dữ liệu thành training và test sets

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    # Tính trọng số nghịch đảo cho các class để cân bằng

    class\_counts = np.bincount(y\_train)

    total\_samples = len(y\_train)

    class\_weights = {i: total\_samples / (len(class\_counts) \* count)

                     for i, count in enumerate(class\_counts)}

    print("\nTrọng số cho từng class:")

    for class\_idx, weight in class\_weights.items():

        print(f"Class {class\_idx}: {weight:.3f}")

    # Khởi tạo và train model với hyperparameters đã điều chỉnh

    model = OVOSVM()

    model.fit(X\_train, y\_train)

    # Đánh giá mô hình trên tập test

    y\_pred = model.predict(X\_test)

    accuracy = accuracy\_score(y\_test, y\_pred)

    report = classification\_report(y\_test, y\_pred)

    print("\nĐộ chính xác trên tập test:", accuracy)

    print("\nBáo cáo chi tiết:\n", report)

    # Vẽ kết quả cho từng cặp classes

    class\_pairs = list(combinations(np.unique(y\_train), 2))

    for class\_pair in class\_pairs:

        plot\_training\_data(X\_train, y\_train, model, class\_pair)

    # Lưu model

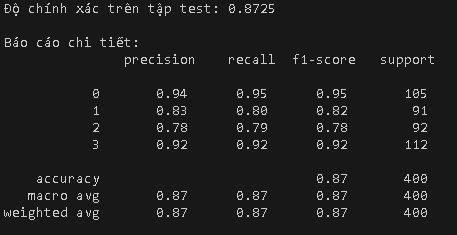
    np.save('ovo\_svm\_models.npy', model.models)

    print("\nĐã train xong và lưu model!")

* **Tải và kiểm tra dữ liệu huấn luyện**: Đọc dữ liệu từ tệp CSV và kiểm tra phân phối các lớp trong tập huấn luyện.
* **Tách features và labels**: Tách dữ liệu đầu vào (features) và nhãn (labels).
* **Chia dữ liệu**: Chia dữ liệu thành tập huấn luyện và kiểm tra (80-20%).
* **Tính trọng số cho các lớp**: Tính trọng số nghịch đảo cho mỗi lớp để cân bằng dữ liệu.
* **Huấn luyện mô hình**: Khởi tạo và huấn luyện mô hình **One-vs-One SVM**.
* **Đánh giá mô hình**: Dự đoán và tính toán độ chính xác, báo cáo chi tiết trên tập kiểm tra.
* **Vẽ kết quả**: Vẽ đồ thị phân phối dữ liệu cho mỗi cặp lớp.
* **Lưu mô hình**: Lưu các mô hình đã huấn luyện vào tệp.

Sau khi chạy mô hình kết quả trả về sẽ gồm :

* 1 bảng báo cáo độ chính xác



* 6 biểu đồ ovo svm

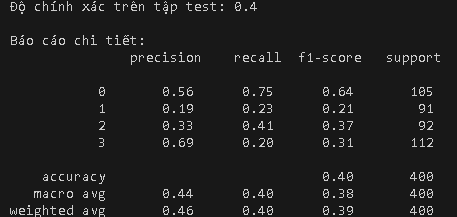
Đánh giá mô hình

+ Bọn em đã tinh chỉnh nhiều mô hình với các:

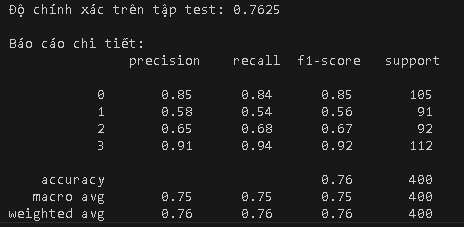
* C,max-inner,leaning\_rate,momentum,patience

Ví dụ:



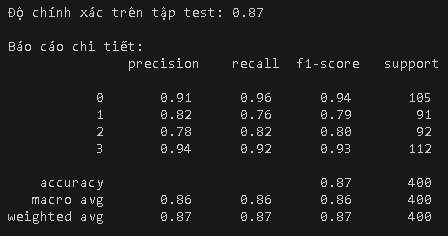






+ Kết quả cho thấy





Cho ra đọ chính xác cao nhất là 87%

Độ chính xác tổng thể

Độ chính xác trên tập test: 87%: Điều này có nghĩa là mô hình của bạn đúng 87% trong các dự đoán trên tập dữ liệu kiểm tra.

Báo cáo chi tiết theo từng lớp

Báo cáo chi tiết cung cấp các chỉ số quan trọng như precision (độ chính xác), recall (tỷ lệ hồi đáp), f1-score, và support (số lượng mẫu) cho từng lớp.

Bảng phân tích

| Lớp | Precision | Recall | F1-Score | Support |
| --- | --- | --- | --- | --- |
| 0 | 0.91 | 0.96 | 0.94 | 105 |
| 1 | 0.82 | 0.76 | 0.79 | 91 |
| 2 | 0.78 | 0.82 | 0.80 | 92 |
| 3 | 0.94 | 0.92 | 0.93 | 112 |

Để đánh giá chất lượng của một mô hình phân lớp ta sẽ sử dụng 4 chỉ số sau:

**Accuracy** là độ chính xác của mô hình được tính theo công thức sau

**Precision (độ chính xác theo lớp dương)**: Chỉ số này dùng để đo lượng độ chính xác trong các dự đoán là Positive. Trong tất cả các mẫu được dữ đoán là Positive, tỉ lệ phần tram nào thực sự là Positive. Mô hình mà có Positive cao nghĩa là mô hình ít có khả năng tạo ra các dự đoán dương giả (False Positive). Nghĩa là bệnh nhân đó mô hình dự đoán là Positive nhưng thực tế lại là Negative. Được tính theo công thức sau

**Recall (độ nhạy hoặc tỉ lệ phát hiện):** Recall đo lường khả năng của mô hình trong việc tìm ra các mẫu Positive. Trong tất cả các mẫu thực sự là Positive thì mô hình dự đoán được bao nhiêu. Vậy Recall cao nghĩa là mô hình ít bỏ sót các trường hợp Positive thực sự. Chỉ số được tính theo công thức sau.

**F1-Score (Trung bình của Precision và Recall):** là một chỉ số cân bằng giữa Recall và Precision, đặc biệt hữu ích khi ta cần cân bằng cả 2 chỉ số trên. Nếu F1-Score cao có nghĩa là mô hình hoạt động tốt ở cả hai khía cạnh: Ít False Positive và ít False Negative. Chỉ số được tính theo công thức sau.



KẾT LUẬN

Mô hình SVM kết hợp với chiến lược phân loại One-vs-One đã chứng minh là một phương pháp hiệu quả và mạnh mẽ để dự đoán mức giá điện thoại dựa trên các đặc trưng của điện thoại di động. Kết quả của bài nghiên cứu này không chỉ cung cấp một giải pháp hiệu quả cho việc phân loại mức giá điện thoại mà còn mở ra những hướng nghiên cứu và ứng dụng khác trong lĩnh vực phân loại và dự đoán với dữ liệu phức tạp.

Hy vọng nghiên cứu này sẽ góp phần vào việc phát triển các mô hình dự đoán và phân loại chính xác hơn trong tương lai, đồng thời cung cấp một công cụ hữu ích cho các nhà nghiên cứu và thực hành trong lĩnh vực học máy và trí tuệ nhân tạo.

**TÀI LIỆU THAM KHẢO**

[Mobile Price Classification](https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification)

[Multi-class classification using Support Vector Machines (SVM) - GeeksforGeeks](https://www.geeksforgeeks.org/multi-class-classification-using-support-vector-machines-svm/)