BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC UEH
TRƯỜNG CÔNG NGHỆ và THIẾT KẾ
---o0o---
KHOA CÔNG NGHỆ THÔNG TIN KINH DOANH

ĐỒ ÁN KẾT THÚC HỌC PHẦN
MÔN MÁY HỌC

Chủ đề:

Random Forest

Giảng viên hướng dẫn: Ts. Nguyễn An Tế
Thực hiện: Nhóm 7
Danh sách nhóm: Lâm Thy Nhã
Võ Yến Nhi
Nguyễn Lê Thanh Oanh
Vương Kiến Phát
Lê Đình Phong

Lớp học phần: 4C1INF50904401
TP. Hồ Chí Minh, tháng 11 năm 2024

BẢNG ĐÁNH GIÁ

Họ và tên	Mã số sinh viên	Mức độ đóng góp
Nguyễn Lê Thanh Oanh	31221020761	100%
Lâm Thy Nhã	31211020484	100%
Vương Kiến Phát	31221022681	100%
Lê Đình Phong	31221025185	100%
Võ Yến Nhi	31221026992	100%

1. GIỚI THIỆU THUẬT TOÁN RANDOM FOREST:

1.1. Cây quyết định (Decision Tree):

Decision Tree là thành phần cơ bản và là đơn vị cấu thành chính trong Random Forest. Tuy nhiên, Decision Tree có những nhược điểm mà việc phát triển Random Forest sinh ra để khắc phục chúng:

1.1.1. Overfitting

Hạn chế lớn nhất của Decision Tree chính là dễ bị Overfitting. Hiện tượng này xảy ra khi mô hình xây dựng được và dữ liệu training quá khớp nhau dẫn tới dự đoán bị nhầm lẫn. Đặc biệt trong trường hợp lượng dữ liệu training quá nhỏ nhưng độ phức tạp của mô hình lại quá cao.

Ví dụ ta có bộ dữ liệu ghi nhận lịch sử đặt phòng *"Hotel Booking Cancellation Prediction"* mà nhóm sử dụng, giả sử tồn tại một mẫu đặt phòng đặc biệt xảy ra trong mùa cao điểm với giá $200 nhưng lại bị hủy, trong khi hầu hết các đặt phòng trong mùa cao điểm (với giá cao như vậy) thường không bị hủy. Decision Tree có thể tạo ra một nhánh mới chỉ để xử lý mẫu này, làm cho cây hoạt động rất tốt trên tập train, nhưng khi tới tập test nó sẽ gặp khó khăn vì không tổng quát hóa được. Decision Tree hoạt động dựa trên nguyên tắt liên tục chia dữ liệu thành các nhóm nhỏ dựa trên các đặc trưng (như *"giá trung bình"*, *"mùa cao điểm"*,...). Trên thực tế, mẫu này có thể do sơ suất từ phía nhân viên hoặc do lý do cá nhân của khách. Khi áp dụng mô hình này lên dữ liệu mới, các mẫu tương tự (*"mùa cao điểm"*, *"giá $200"*) nhưng không bị hủy sẽ có thể bị dự đoán sai.

Với trường hợp này, sử dụng Random Forest sẽ giảm thiểu sự phụ thuộc vào các mẫu cụ thể, giúp cải thiện độ chính xác và ổn định của mô hình. Trong Random Forest, mỗi cây chỉ sử dụng một phần training data (bootstrapping) và một tập con các đặc trưng; sau đó đưa ra dự đoán dựa trên trung bình hoặc bỏ phiếu từ nhiều cây (giảm ảnh hưởng của các cây bị overfitting)

1.1.2. Dễ bị ảnh hưởng bởi Outliers

Một giá trị outlier có thể ảnh hưởng lớn tới cách một cây phân chia dữ liệu. Giả sử tồn tại 1 mẫu đặt phòng có giá bất thường là $10000, trong khi giá trị trung bình của giá phòng chỉ tầm $100; giống như ở trường hợp overfitting, Decision Tree sẽ tạo ra một nhánh riêng để xử lý, làm giảm hiệu quả của các dự đoán khác.

Trong khi đó Random Forest tạo ra nhiều cây huấn luyện trên các traning set khác nhau, từ đó làm giảm đáng kể tác động của các giá trị ngoại lai.

1.1.3. Mất cân bằng dữ liệu (Imbalanced Data)

Trong trường hợp dữ liệu có tỷ lệ không đồng đều giữa 2 nhóm mục tiêu, giả sử tỷ lệ nhãn *"Canceled"* và *"Not_Canceled"* chênh lệch khá lớn (30% và 70%), Decision Tree thường có xu hướng thiên vị nhóm có tỷ lệ lớn hơn, dẫn tới dễ bỏ qua các yếu tố đặc biệt của nhóm thiểu số.

Đặt ra trường hợp *"giá trung bình dưới $50"* có tỷ lệ *"Canceled"* rất cao. Decision Tree quyết định phân nhánh như sau:

- *"Giá trung bình dưới $50"* → Dự đoán hủy phòng (*Canceled*).
- *"Giá trung bìn*h trên $50" → Dự đoán không hủy phòng (*Not Canceled*).

Có thể thấy trong trường hợp này Decision Tree đã bỏ qua các yếu tố quan trọng khác (thời gian đặt trước, số người đặt phòng,...). Vấn đề xảy ra ở đây chính là nếu trong testing set tồn tại mẫu *"giá trung bình dưới $50"* và có yêu cầu đặc biệt hoặc thời gian đặt phòng dài, Decision Tree vẫn sẽ dự đoán là *"Canceled"* dù cho thực tế khách hàng đó không hề hủy phòng.

Random Forest khắc phục hạn chế trên của Decision Tree bằng cách xây dựng "rừng cây", mỗi cây học từ một phần khác nhau và dựa vào các đặc trưng khác nhau của dữ liệu. Dự đoán của Random Forest được tổng hợp trung bình từ nhiều cây hoặc bỏ phiếu đa số.

1.2. Cách hoạt động của thuật toán Random Forest:

Ta có giả định như sau: $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ là một tập dữ liệu bao gồm N quan sát. Sử dụng phương pháp chọn mẫu có hoàn lại để tạo thành B tập dữ liệu con. Ta thực hiện M lượt nhặt các mẫu từ tổng thể và bỏ vào túi để tạo thành tập B_i, $= \{(x_1^{(i)}, y_1^{(i)}), (x_2^{(i)}, y_2^{(i)}), \ldots, (x_M^{(i)}, x_M^{(i)})\}$. Trong tập B_i, các phần tử được phép lặp lại.

Với mỗi tập B_i ta sẽ xây dựng được một cây quyết định tương ứng, và trả về kết quả dự báo là $\hat{y}_j^{(i)} = f_i(x_j)$. Trong đó, $\hat{y}_j^{(i)}$ là dự báo của quan sát thứ j từ mô hình thứ (i), x_j là giá trị vector đầu vào, $f_i(x_j)$ là hàm dự báo của mô hình thứ i.

Khi xây dựng một cây quyết định từ mẫu bootstrap, ta chọn ngẫu nhiên $m \leq p$ thuộc tính, với mục tiêu, ví dụ: tối đa hóa giá trị của Information Gain,... thuộc tính nào trong nhóm m thuộc tính này có chất lượng phân chia tốt nhất sẽ được chọn làm node ở bước đó. Loại thuộc tính đã được chọn ra khỏi tập thuộc tính p ban đầu, tiếp tục chọn ngẫu nhiên m thuộc tính để xem xét. Cứ lặp lại như vậy cho đến khi xây dựng cây quyết định.

Kết quả dự đoán của thuật toán:

- **Đối với mô hình hồi quy:** Giá trị dự đoán cuối cùng là trung bình của tất cả các dự đoán từ các cây, như công thức sau:

$$\hat{y}_j = \frac{1}{B}\Sigma_{i=1}^{B}\hat{y}_j^{(i)}$$

- **Đối với mô hình phân loại:** Mỗi cây trong Random Forest dự đoán một nhãn cho điểm x, và nhãn cuối cùng được chọn dựa trên nhãn có tần suất xuất hiện nhiều nhất.

$$\hat{y}_j = argmax_c \Sigma_{i=1}^{B} p\left(\hat{y}_j^{(i)} = c\right)$$

1.3. Ví dụ minh họa:

Ví dụ ta có tập dữ liệu ban đầu có số phần tử N = 4, số thuộc tính đầu vào p = 4 như sau:

Id	Chest Pain	Good Blood Circulation	Blocked Arteries	Weight	Heart Disease
1	No	No	No	125	No
2	Yes	Yes	Yes	180	Yes
3	Yes	Yes	No	210	No
4	Yes	No	Yes	167	Yes

Bảng 1.1. Tập dữ liệu ban đầu

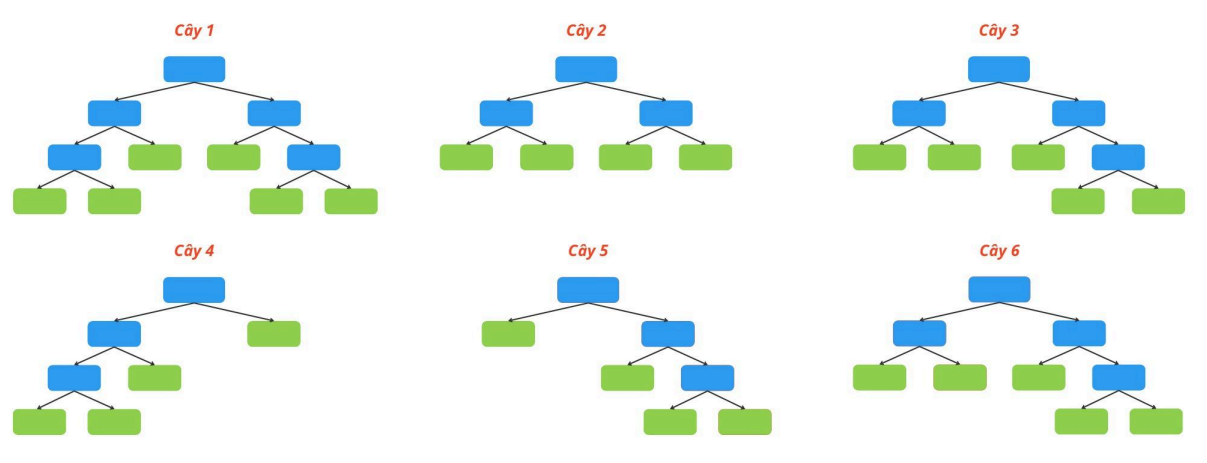
Chọn ngẫu nhiên 4 phần tử từ tập dữ liệu ban đầu theo phương pháp chọn mẫu có hoàn lại để tạo tập bootstrap, ta lần lượt chọn được các phần tử 2, 1, 4, 4. Lưu ý rằng các phần tử trong tập bootstrap được phép lặp lại.

Id	Chest Pain	Good Blood Circulation	Blocked Arteries	Weight	Heart Disease
2	Yes	Yes	Yes	180	Yes
1	No	No	No	125	No
4	Yes	No	Yes	167	Yes
4	Yes	No	Yes	167	Yes

Bảng 1.2. Tập bootstrap

Chọn ngẫu nhiên m = 2 thuộc tính từ tập dữ liệu ban đầu, ở đây ta chọn được Good Blood Circulation và Blocked Arteries là ứng viên cho node gốc. Giả sử Good Blood Circulation là thuộc tính có chất lượng phân chia tốt hơn nên ta chọn nó làm node gốc. Vì thuộc tính Good Blood Circulation đã được dùng làm node gốc, ta sẽ loại nó ra khỏi tập thuộc tính xem xét. Sau đó với ba thuộc tính còn lại (Chest Pain, Blocked Arteries, Weight), ta tiếp tục chọn ngẫu nhiên 2 thuộc tính. Giả sử ta chọn được hai thuộc tính là Chest Pain và Weight. Ta tiếp tục so sánh chất lượng phân chia của hai thuộc tính nhằm xây dựng cây quyết định như thông thường, chỉ khác là ta chỉ có một số lượng m thuộc tính ngẫu nhiên ở mỗi bước để xét.

Sau đó, lặp lại bước tạo mẫu bootstrap và xây dựng cây quyết định cho đến khi xây dựng được B cây quyết định. Giả sử $B = 6$.



Hình 1.1. Giả định 6 cây quyết định được xây dựng từ các mẫu bootstrap

Bây giờ ta có một quan sát mới như sau:

Id	Chest Pain	Good Blood Circulation	Blocked Arteries	Weight	Heart Disease
5	Yes	No	No	168	?

Bảng 1.3. Quan sát mới

Ta chạy quan sát này qua từng cây quyết định và ghi nhận lại kết quả ở mỗi cây. Giả sử ta thu được kết quả như sau:

Heart Disease	
Yes	No
5	1

Bảng 1.4. Kết quả dự báo

Vì nhãn "Yes" có nhiều vote hơn hơn, ta quyết định gán nhãn cho quan sát mới này là "Yes".

Id	Chest Pain	Good Blood Circulation	Blocked Arteries	Weight	Heart Disease
5	Yes	No	No	168	**Yes**

Bảng 1.5. Nhãn của quan sát mới

1.4. Giảm phương sai

Ta có phương sai của kết quả dự báo từ mô hình trong trường hợp đối với bài toán hồi quy:

$$\sigma^2 = Var(\frac{1}{B}\Sigma_{i=1}^{B}\hat{y}^{(i)})$$

$$\sigma^2 = \frac{1}{B^2}[\sum_{i=1}^{B} Var(\hat{y}^{(i)}) + 2\sum_{1\le m\le n\le B} Cov(y^{(m)}, y^{(n)})]$$

Trong đó:

- $\Sigma_{i=1}^{B} Var(\hat{y}^{(i)})$: Tổng phương sai của từng cây độc lập
- $\sum_{1\le m\le n\le B} Cov(y^{(m)}, y^{(n)})$: Hiệp phương sai giữa các cặp cây

Kết quả của mô hình con A không chịu ảnh hưởng hoặc phụ thuộc vào mô hình con B nên ta có thể giả định kết quả dự báo từ các mô hình hoàn toàn độc lập nhau. Tức là ta có $Cov(y^{(m)}, y^{(n)}) = 0, \forall 1 \le m \le n \le B$. Đồng thời giả định chất lượng các mô hình là đồng đều, được thể hiện qua phương sai dự báo là đồng nhất $Var(\hat{y}^{(i)}) = \sigma^2, \forall i = 1, B$. Từ đó suy ra:

$$\sigma^2_{\hat{y}} = \frac{1}{B^2}\Sigma_{i=1}^{B} Var(\hat{y}^{(i)}) = \frac{1}{B^2} B\,\sigma^2 = \frac{1}{B}\sigma^2$$

Một trung bình của B biến ngẫu nhiên độc lập và đồng phân phối, mỗi biến có phương sai σ^2 sẽ có phương sai là $\frac{1}{B}\sigma^2$. Nếu các biến này chỉ đơn giản là đồng phân phối (i.d., nhưng không nhất thiết độc lập) với tương quan cặp dương ρ, ta có $Cov(y^{(m)}, y^{(n)}) = \rho\sigma^2$. Phương sai trung bình sẽ là:

$$\sigma^2_{\hat{y}} = \frac{1}{B^2}[\Sigma_{i=1}^{B} Var(\hat{y}^{(i)}) + 2\sum_{1\le m\le n\le B} Cov(y^{(m)}, y^{(n)})]$$

$$\sigma^2_{\hat{y}} = \frac{1}{B^2}(B\sigma^2 + B(B-1)\rho\sigma^2)$$

$$\sigma^2_{\hat{y}} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

Khi tăng số lượng cây ở B, thành phần $\frac{1-\rho}{B}\sigma^2$ sẽ giảm, và phương sai của kết quả sẽ giảm dần, nhưng chỉ giảm đến một giới hạn nhất định là $\rho\sigma^2$. Nếu ρ nhỏ, tức là các cây ít phụ thuộc vào nhau (các cây độc lập hơn), thì phương sai của kết quả sẽ giảm đáng kể khi tăng số lượng cây B. Nếu ρ lớn, tức là các cây phụ thuộc nhiều vào nhau, việc tăng số lượng cây sẽ không làm giảm phương sai nhiều.

Random Forest là một trong những đặc trưng của bagging. Bagging, hay còn gọi là bootstrap aggregation, là một kỹ thuật ensemble learning nhằm giảm phương sai của một hàm dự đoán ước lượng bằng cách trung bình nhiều mô hình có phương sai cao nhưng độ chệch thấp nhằm giảm phương sai. Cây quyết định được xem là một mô hình phù hợp với bagging vì chúng có độ chệch tương đối thấp nếu được phát triển đủ sâu, nhưng bù lại chúng lại rất nhiễu và do đó, chúng sẽ hưởng lợi rất nhiều từ việc trung bình hóa các dự đoán. Mỗi cây được tạo ra trong Bagging đều có phân phối giống nhau (i.d.), do đó, kỳ vọng của một trung bình của B cây sẽ giống như kỳ vọng của bất kỳ cây nào trong số đó. Điều này có nghĩa là độ chệch của các cây đã được bagging sẽ giống như độ chệch của các cây đơn lẻ, và cách duy nhất để cải thiện kết quả là thông qua việc làm giảm phương sai.

Random Forest làm giảm phương sai tốt hơn Bagging ở chỗ khác với Bagging sử dụng toàn bộ p thuộc tính để tạo cây, Random Forest chọn ngẫu nhiên m biến đầu vào làm ứng viên cho mỗi node. Nếu mỗi cây quyết định sử dụng tất cả các thuộc tính, các cây có thể được xây dựng giống nhau nếu một số thuộc tính có chất lượng phân chia nhánh vượt trội. Bằng cách chỉ sử dụng một tập con ngẫu nhiên các thuộc tính tại mỗi node, Random Forest đảm bảo rằng mỗi cây được xây dựng trên một không gian tìm kiếm khác nhau, giảm sự phụ thuộc giữa các cây trong rừng, giúp các cây ít giống nhau hơn. việc chọn ngẫu nhiên những biến đầu vào còn làm giảm khả năng các cây học quá kỹ từ dữ liệu huấn luyện, giảm overfitting và tăng tính đa dạng cho các cây. Điều này làm giảm tương quan ρ giữa các cây vì các cây ít có khả năng sử dụng các thuộc tính giống nhau để đưa ra dự đoán.

1.5. Siêu tham số

Khi xây dựng cây quyết định, ta cần chú ý đến các siêu tham số sau đây để tối đa hóa hiệu quả của mô hình:

- Số lượng cây quyết định được xây dựng B (n_estimators): càng lớn thì hiệu quả của mô hình Random Forest càng tăng, nhưng bù lại thì chi phí tính toán cũng tăng.
- Số lượng mẫu n (max_samples) trong tập bootstrap cần lấy để xấy dựng cây thông thường được chọn bằng với số lượng mẫu trong tập train ban đầu.
- Số lượng thuộc tính m (max_features) để tách ở mỗi bước:
  - Với bài toán phân loại, giá trị m thông thường là $m = \sqrt{p}$

- Với bài toán hồi quy, giá trị m thông thường là

Ngoài ra, còn có các thuộc tính của thuật toán Decision Tree để xây dựng cây như độ sâu tối đa, số phần tử tối thiểu trong 1 node để có thể tách.

2. MỘT VẤN ĐỀ MỞ RỘNG CỦA THUẬT TOÁN RANDOM FOREST

2.1. Mẫu ngoài túi (Out-of-Bag Samples)

2.1.1. Tìm hiểu về mẫu ngoài túi

Một đặc điểm độc đáo của Random Forest là sử dụng mẫu ngoài túi (Out-of-Bag - OOB). Khi xây dựng cây, mỗi cây sẽ được huấn luyện trên một mẫu bootstrap. Khi thực hiện bootstrap aggregation, hai tập dữ liệu độc lập sẽ được tạo ra. Một là tập dữ liệu bootstrap, bao gồm các quan sát được chọn theo phương pháp chọn mẫu có hoàn lại từ tập dữ liệu gốc, những quan sát trong tập bootstrap này còn được gọi là các dữ liệu có-trong-túi (in-the-bag). Còn lại tập dữ liệu chỉ chứa các các mẫu ngoài túi (OOB), là các quan sát không được chọn trong quá trình lấy mẫu bên trên.

2.1.2. Ví dụ minh họa

Tiếp tục với Ví dụ minh họa ở Phần 1.2, ta đã tạo ra một tập bootstrap bao gồm 4 quan sát. Các quan sát của tập bootstrap này đã lấy các phần tử thứ 2, 1 và 4 của tập dữ liệu gốc. Vậy còn phần tử thứ 3 của tập dữ liệu gốc chưa được đưa vào tập dữ liệu bootstrap khi ta thực hiện bootstrap aggregation.

Id	**Chest Pain**	**Good Blood Circulation**	**Blocked Arteries**	**Weight**	**Heart Disease**
3	Yes	Yes	No	210	No

Bảng 2.1. Ví dụ - Mẫu thuộc tập dữ liệu OOB (1)

→ Phần tử thứ 3 này sẽ mẫu thuộc tập dữ liệu OOB của cây quyết định này.

2.2. Out-of-Bag Error

2.2.1. Tìm hiểu về Out-of-Bag Error

Khi xây dựng một Random Forest, nhiều tập bootstrap và bộ OOB sẽ được tạo ra. Các bộ OOB có thể được tổng hợp thành một tập dữ liệu, nhưng mỗi mẫu chỉ được coi là

out-of-bag đối với các cây không bao gồm nó trong mẫu bootstrap của chúng. Vì mỗi mẫu OOB không được sử dụng để đào tạo các mô hình có tập bootstrap không chứa nó, nên ta có thể sử dụng mẫu OOB để kiểm tra hiệu quả mô hình cây đó. Quá trình này không chỉ giúp kiểm tra riêng lẻ từng cây mà còn cung cấp một đánh giá tổng thể về khả năng dự đoán của toàn bộ mô hình mà không cần phải chia tập dữ huấn luyện và tập kiểm tra riêng biệt.

Ta sẽ lấy mẫu OOB này chạy để kiểm tra mô hình cây không sử dụng nó sẽ để dự đoán giá trị cho mẫu OOB. Mô hình cây có thể dự đoán đúng hoặc dự đoán sai so với giá trị thực tế của mẫu OOB đó. Và ta có Out-of-Bag Error (OOB Error) là một giá trị ước tính về lỗi dự đoán của Random Forest dựa trên những mẫu không được sử dụng trong quá trình huấn luyện mỗi cây (các mẫu OOB). Bên dưới là các bước để tính toán OOB Error:

- **Bước 1:** Xác định các mẫu OOB cho mỗi cây. Mỗi mẫu dữ liệu sẽ là OOB cho một số cây nhất định trong rừng.
- **Bước 2:** Dự đoán bằng các cây không chứa mẫu OOB. Đối với mỗi mẫu dữ liệu, tìm các cây không sử dụng mẫu đó trong quá trình huấn luyện. Sử dụng những cây này để dự đoán nhãn cho mẫu OOB. Với bài toán phân loại, sử dụng phương pháp majority vote (lấy nhãn xuất hiện nhiều nhất). Với bài toán hồi quy, lấy giá trị trung bình của các dự đoán.
- **Bước 3:** So sánh dự đoán với nhãn thực tế. So sánh nhãn dự đoán từ các cây với nhãn thực tế của mẫu OOB. Nếu dự đoán sai, đánh dấu mẫu đó là lỗi.
- **Bước 4:** Tính OOB Error. OOB Error được tính bằng công thức:

  OOB Error = Số lượng mẫu dự đoán sai / Tổng số mẫu OOB

2.2.2. Ví dụ minh họa

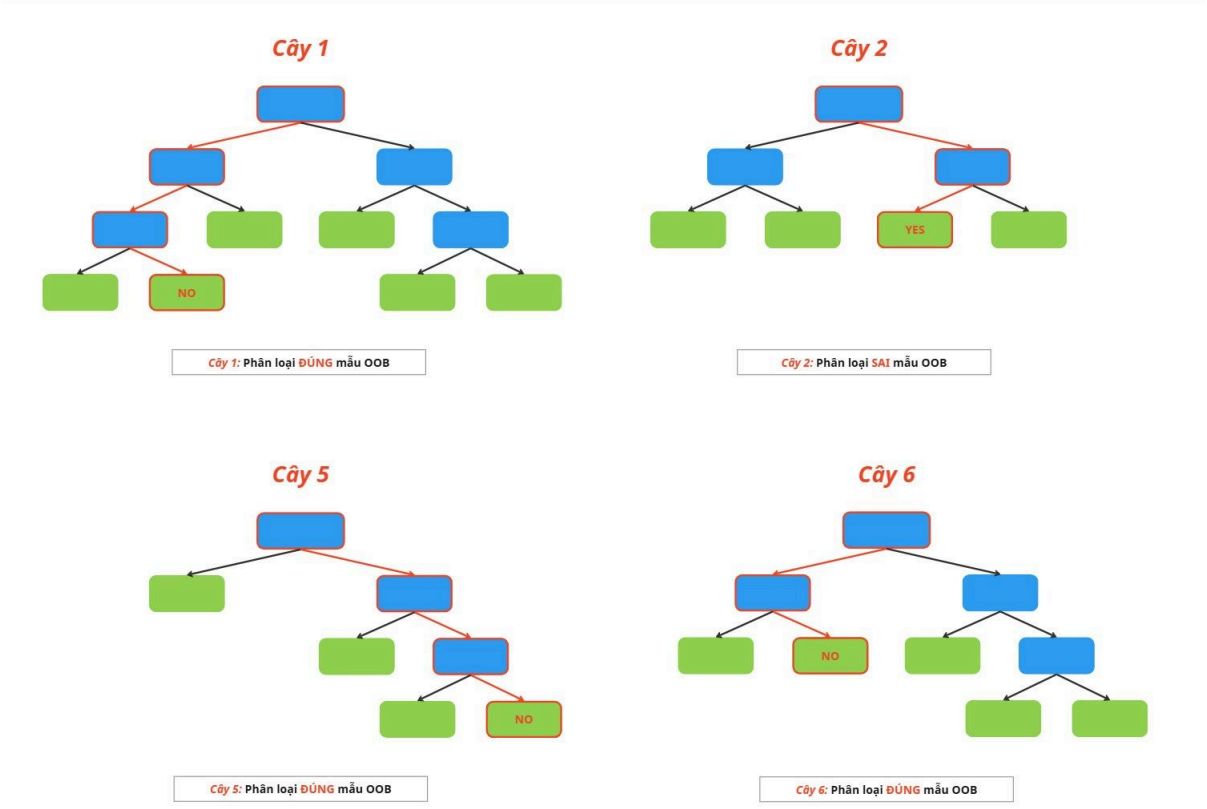
Tiếp tục với Ví dụ ở Phần 2.1.2, giả sử rằng mẫu dữ liệu OOB bên dưới không được sử dụng để huấn luyện cho các cây: Cây 1, Cây 2, Cây 5 và Cây 6 của Hình 1.1.

Id	**Chest Pain**	**Good Blood Circulation**	**Blocked Arteries**	**Weight**	**Heart Disease**
3	Yes	Yes	No	210	No

Bảng 2.2. Ví dụ - Mẫu thuộc tập dữ liệu OOB (2)

Ta sẽ sử dụng nó để kiểm tra hiệu quả phân loại của Cây 1, Cây 2, Cây 5 và Cây 6 trong Random Forest. Ta sẽ chạy qua từng cây trong số 4 vả xem các cây có phân loại đúng cột Heart Disease có giá trị là No hay không. Sau khi đã chạy mẫu qua 4 cây, giả sử ta thu được kết quả như hình bên dưới:

- Cây 1, Cây 5 và Cây 6 đã phân loại cột Heart Disease có giá giá trị là NO
- Cây 2 phân loại cột Heart Disease có giá trị là YES.



Hình 2.1. Ví dụ - Minh họa sử dụng mẫu OOB để kiểm tra mô hình Random Forest

Ta sẽ thu được bảng kết quả phân loại cho cột Heart Disease mẫu OOB trên, đặt tên là mẫu OOB 1, như sau:

Yes	No
1	3

Bảng 2.3. Ví dụ - Bảng kết quả sử dụng mẫu OOB 1 để chạy mô hình

Dựa vào tính chất bỏ phiếu đa số của Random Forest, ta sẽ phân loại Heart Disease của mẫu OOB 1 là NO. Vậy trong trường hợp này, mẫu OOB đã được phân loại đúng bởi Random Forest.

Tiếp theo, ta cứ lặp lại tương tự cho các quan sát còn lại trong tập dữ liệu OOB của tất cả các cây. Giả định rằng trong tập dữ liệu OOB sẽ bổ sung thêm 2 quan sát nữa cho các cây khác là:

Id	Chest Pain	Good Blood Circulation	Blocked Arteries	Weight	Heart Disease
1	No	No	No	125	No
2	Yes	Yes	Yes	180	Yes

Bảng 2.4. Ví dụ - Bổ sung các dòng dữ liệu cho tập dữ liệu OOB

Và sau khi chạy các mẫu OOB này cho các cây không chứa nó và sử dụng bỏ phiếu đa số, ta thu được kết quả như sau:

Id	Yes	No	Heart Disease được dự đoán	Phân loại
1	3	1	YES	Sai
2	4	0	YES	Đúng

Bảng 2.5. Ví dụ - Bảng kết quả sử dụng 2 mẫu bổ sung để chạy mô hình

Từ đó, ta có thể tính toán được độ chính xác mà Random Forest này có thể đặt được thông qua tỉ lệ số mẫu OOB được phân loại đúng bởi Random Forest. OOB Error sẽ là tỷ lệ mẫu OOB được phân loại sai bởi Random Forest và sẽ được tính như sau:

OOB Error = Số lượng mẫu dự đoán sai / Tổng số mẫu OOB = 1/3 = 0.333

2.3. Tầm quan trọng của biến

Sau khi huấn luyện một Random Forest, ta thường muốn biết biến nào có sức mạnh dự đoán tốt nhất. Các biến có tầm quan trọng cao là những yếu tố chính ảnh hưởng đến kết quả, và giá trị của chúng có tác động đáng kể đến đầu ra. Ngược lại, các biến có tầm quan trọng thấp có thể bị loại bỏ khỏi mô hình, giúp mô hình đơn giản hơn và nhanh hơn khi huấn luyện cũng như dự đoán.

Có hai phương pháp đánh giá độ quan trọng của biến trong thuật toán Random Forest:

Phương pháp 1. Đánh giá dựa trên cải thiện tiêu chí chia (split-criterion):

Ta có các chỉ số thể hiện tiêu chí chia phổ biến trong Decision Tree như sau:

- Đối với bài toán phân loại: Tiêu chí phổ biến là Gini Index và Entropy.
- Đối với bài toán hồi quy: Tiêu chí phổ biến là Mean Squared Error (MSE).

Cách thực hiện:

- Với mỗi thuộc tính x_j trong p thuộc tính đầu vào và với mỗi cây được xây dựng trong Random Forest, ta tìm tất cả các node có sử dụng thuộc tính x_j để chia nhánh. Ta tính toán sự khác biệt về chỉ số tiêu chí chia trước và sau khi chia nhánh sử dụng thuộc tính x_j và lưu lại kết quả.
- Cộng tất cả các kết quả sau khi tính toán sự sai khác ở tất cả các cây, đây chính là chỉ số thể hiện tầm quan trọng của biến x_j.

Phương pháp 2. Dựa trên độ giảm chính xác (Accuracy-based important):

Các bước để xác định được tầm quan trọng của biến dựa trên độ giảm chính xác của mô hình:

- Bước 1: Huấn luyện mô hình Random Forest với tập dữ liệu bootstrap.
- Bước 2: Sử dụng tập dữ liệu OOB để kiểm tra lại độ hiệu quả của mô hình, tính toán độ chính xác của mô hình khi phân loại tập OOB.
- Bước 3: Sau đó, xáo trộn ngẫu nhiên giá trị của của biến này trong tập OOB, trong đó vẫn giữ nguyên các biến khác.
- Bước 4: Sử dụng tập dữ liệu OOB vừa được xáo trộn giá trị trong một biến để tính toán lại độ chính xác của mô hình khi phân loại tập OOB mới.
- Bước 5: Giảm độ chính xác do hoán vị được tính toán và trung bình hóa qua tất cả các cây trong rừng. Giá trị này được sử dụng làm thước đo độ quan trọng của biến đó.

Ý nghĩa: Việc hoán vị ngẫu nhiên các giá trị của biến phá vỡ mối liên hệ giữa biến đó và biến mục tiêu. Nếu mô hình bị giảm đáng kể độ chính xác sau khi thực hiện thay đổi, điều này cho thấy biến đó có tầm quan trọng cao đối với dự đoán. Nếu không, biến có thể không quan trọng và có thể loại bỏ để đơn giản hóa mô hình.

Ví dụ bảng bên dưới là tập dữ liệu OOB ban đầu. Sau khi chạy tập dữ liệu này với mô hình Random Forest, ta thu được độ chính xác ban đầu của mô hình phân loại tập OOB là 90%.

ID	Chest Pain	Good Blood Circulation	Blocked Arteries	Weight (kg)	Heart Disease (Target)
1	No	No	No	125	No
2	Yes	Yes	Yes	180	Yes
3	Yes	Yes	No	210	No
4	Yes	No	Yes	167	Yes

Bảng 2.6. Ví dụ - Giả định Tập dữ liệu OOB ban đầu

Đến Bước 3, ta sẽ thay đổi ngẫu nhiên các giá trị của biến Chest Pain để cho ra một tập dữ liệu OOB mới, những biến còn lại sẽ giữ nguyên giá trị cho các mẫu:

ID	Chest Pain	Good Blood Circulation	Blocked Arteries	Weight (kg)	Heart Disease (Target)
1	Yes	No	No	125	No
2	No	Yes	Yes	180	Yes
3	Yes	Yes	No	210	No
4	No	No	Yes	167	Yes

Bảng 2.7. Ví dụ - Giả định Tập dữ liệu OOB có biến Chest Pain thay đổi giá trị

Ta sẽ lấy tập dữ liệu OOB để chạy lại mô hình Random Forest và tính toán ra độ chính xác của mô hình phân loại trên tập mới này, giả sử, ta thấy độ chính xác giảm xuống còn 75%.

Cuối cùng, ta so sánh độ chính xác của mô hình trước và sau khi thay đổi các giá trị của biết Chest Pain, ta thấy Độ giảm độ chính xác = 90% - 75% = 15%. Ta có thể đưa ra kết luận biến "Chest Pain" có ảnh hưởng quan trọng đến mô hình Random Forest vì việc xáo trộn các giá trị của biến này đã làm giảm đáng kể độ chính xác.

2.4. Ma trận gần gũi (Proximity Matrix)

2.4.1. Xây dựng ma trận gần gũi

Ma trận lân cận là một một ma trận vuông kích thước $N \times N$ (với N là số quan sát trong tập dữ liệu), trong đó mỗi phần tử (i, j) biểu diễn mức độ gần nhau giữa hai mẫu i và j.

Ma trận gần gũi được sử dụng để đo lường mức độ tương đồng giữa các điểm dữ liệu trong Random Forest. Nó đo lường tần suất hai mẫu dữ liệu chia sẻ cùng một nút cuối (terminal node) trong các cây của rừng ngẫu nhiên. Bên dưới là các bước để ta có thể xây dựng ma trận gần gũi:

- **Bước 1:** Khởi tạo ma trận. Ban đầu, tất cả các phần tử trong ma trận được gán giá trị 0.
- **Bước 2:** Duyệt qua từng cây trong rừng. Với mỗi cặp mẫu i và j trong tập OOB, nếu chúng cùng nằm ở một nút cuối trong cây đó, tăng giá trị của phần tử (i, j) thêm 1.
- **Bước 3:** Chuẩn hóa ma trận. Sau khi duyệt qua tất cả các cây, mỗi phần tử được chia cho tổng số cây để chuẩn hóa giá trị về khoảng từ 0 đến 1.

$$\text{Proximity}(i, j) = \text{Số lần } i, j \text{ cùng nút cuối/Tổng số cây.}$$

Ma trận gần gũi giúp đo lường mức độ giống nhau giữa các điểm dữ liệu dựa trên cách chúng được nhóm lại trong các cây của Random Forest. Hai điểm được xem là "gần nhau" nếu chúng thường xuyên xuất hiện cùng một nút cuối trong các cây.

- Giá trị cao (gần 1) trong ma trận biểu thị rằng hai mẫu thường xuyên xuất hiện cùng nhau trong cùng một nút cuối, tức là chúng có đặc điểm tương đồng.
- Giá trị thấp hoặc bằng 0 cho thấy hai mẫu ít hoặc không xuất hiện cùng nhau, nghĩa là chúng có thể không giống nhau hoặc thuộc các lớp khác nhau.

Một số ứng dụng của ma trận gần gũi:

- Giúp tìm ra các cụm (clusters) trong dữ liệu bằng cách xác định nhóm điểm thường xuyên xuất hiện cùng nhau.
- Giúp đánh giá mô hình có hiệu quả tốt hay không. Nếu các điểm thuộc cùng một lớp thường xuyên chia sẻ nút cuối, điều đó cho thấy mô hình phân tách lớp hiệu quả.

- Ma trận gần gũi còn có thể giúp điền được các dữ liệu bị thiếu trong tập dữ liệu, bao gồm tối ưu dữ liệu bị thiếu trong tập dữ liệu gốc để tạo Random Forest hoặc điền dữ liệu bị thiếu cho một mẫu mới đang muốn phân loại
- Ngoài ra, ta cũng có trực quan hóa ma trận gần gũi bằng Heatmap hoặc phương pháp Multidimensional Scaling (MDS) để dễ quan sát các điểm dữ liệu hơn.

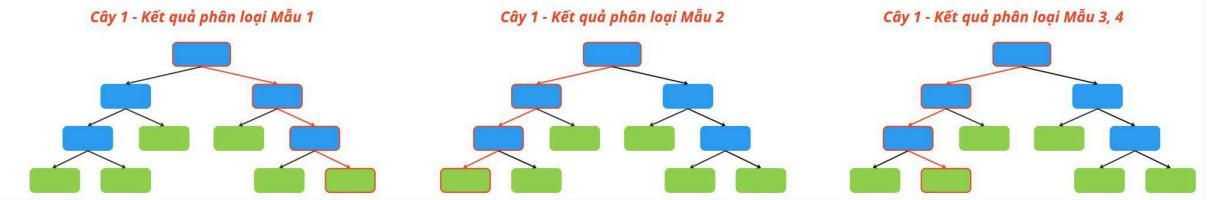
2.4.2. Ví dụ minh họa

Ta vẫn sẽ sử dụng tập dữ liệu của Phần 1.2 - Bảng 1.1 để xây dựng ma trận lân cận. Giả sử, ta sẽ xây dựng 1 rừng ngẫu nhiên có 10 cây quyết định. Tiếp theo, ta sẽ chạy lần lượt tất cả các quan sát của tập dữ liệu cho tất cả 10 cây. Ma trận gần gũi ban đầu sẽ là:

	1	2	3	4
1				
2				
3				
4				

Bảng 2.7. Ví dụ - Ma trận gần gũi khởi tạo

Đầu tiên, ta sẽ chạy 4 quan sát của tập dữ liệu cho Cây 1. Ta thu được kết quả như hình bên dưới. Ta thấy rằng mẫu 3 và mẫu 4 kết thúc phân loại ở cùng một nút lá, thế nên, mẫu 3 và mẫu 4 sẽ tương đồng nhau theo như Random Forest.



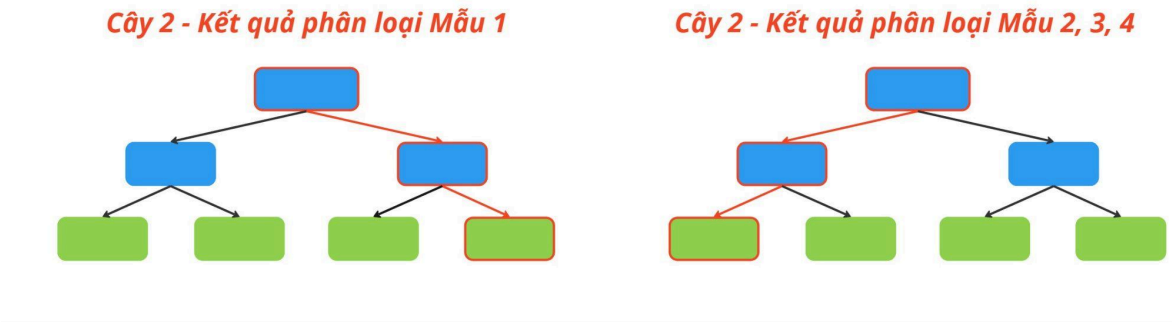
Hình 2.2. Ví dụ - Sử dụng tập dữ liệu OOB để chạy mô hình Cây 1

Sau khi đã chạy xong tất cả các quan sát cho Cây 1, ta chỉnh sửa lại ma trận gần gũi như bên dưới. Bởi vì mẫu 3 và mẫu 4 cùng chia sẻ với nhau một nút lá nên ta sẽ tăng 1 đơn vị ở ô (3, 4) và (4, 3). Ngoài ra, không còn bất kì cặp dữ liệu kết thúc đường đi tại cùng 1 nút lá, nên đây sẽ là ma trận gần gũi sau khi chạy xong cây 1.

	1	2	3	4
1				
2				
3				1
4			1	

Bảng 2.8. Ví dụ - Ma trận gần gũi sau khi chạy mô hình Cây 1

Tiếp theo, lặp lại các bước trên, ta sẽ chạy 4 quan sát của tập dữ liệu cho Cây 2. Ta cũng thu được kết quả như hình bên dưới. Ở cây 2, sẽ có mẫu 2, 3 và 4 cùng kết thúc tại cùng một nút lá.



Hình 2.3. Ví dụ - Sử dụng tập dữ liệu OOB để chạy mô hình Cây 2

Vì vậy, ma trận gần gũi sau khi chạy xong cây 2 sẽ được cộng thêm 1 đơn vị tại cái ô sau: (2, 3) và (3, 2); (2, 4) và (4, 2); (3, 4) và (4, 3).

	1	2	3	4
1				
2			1	1
3		1		2
4		1	2	

Bảng 2.9. Ví dụ - Ma trận gần gũi sau khi chạy mô hình Cây 2

Lặp lại các bước như thế cho đến cây thứ 10, ta sẽ thu được ma trận gần gũi như bên dưới này:

	1	2	3	4
1		2	1	1
2	2		1	1
3	1	1		8
4	1	1	8	

Bảng 2.10. Ví dụ - Ma trận gần gũi sau khi chạy toàn bộ các cây trong mô hình

Sau đó, ta sẽ lấy từng giá trị lân cận trong ma trận này để chia cho Tổng số cây, ta sẽ có được ma trận gần gũi cuối cùng. Nhận xét:

- 2 mẫu cùng nhau xuất hiện ở nút cuối nhiều lần nhất là mẫu 3 và mẫu 4 (giá trị = 0.8)
- Các mẫu còn lại không tương đồng nhau nhiều.

	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

Bảng 2.10. Ví dụ - Ma trận gần gũi hoàn chỉnh

2.4.2.1. Điền dữ liệu bị thiếu trong tập dữ liệu gốc sử dụng Ma trận gần gũi

Giả sử, tập dữ liệu ban đầu sẽ có mẫu 4 bị khuyết mất dữ liệu ở cột Blocked Arteries và cột Weight. Ta sẽ sử dụng ma trận gần gũi để điền vào giá trị bị thiêu đó cho mẫu 4. Tập dữ liệu ban đầu sẽ như sau:

Id	Chest Pain	Good Blood Circulation	Blocked Arteries	Weight	Heart Disease
1	No	No	No	125	No

2	Yes	Yes	Yes	180	Yes
3	Yes	Yes	No	210	No
4	Yes	No	???	???	Yes

Bảng 2.11. Ví dụ - Tập dữ liệu gốc có dữ liệu bị khuyết

Để điền dữ liệu bị khuyết ở cột Blocked Arteries của mẫu 4, ta sẽ đi tính trọng số của từng giá trị (YES và NO) trong cột dựa vào ma trận lân cận:

- Trọng số của giá trị YES = Giá trị lân cận của YES / Tổng giá trị lân cận (của mẫu 4). Giá trị lân cận của YES là ô (4, 2)

$$Trọng\ số\ "YES" = \frac{0.1}{0.1 + 0.1 + 0.8} = 0.1$$

- Trọng số của giá trị NO = Giá trị lân cận của NO / Tổng giá trị lân cận (của mẫu 4). Giá trị lân cận của NO là tổng của ô (4, 1) và (4, 3)

$$Trọng\ số\ "NO" = \frac{0.1 + 0.8}{0.1 + 0.1 + 0.8} = 0.9$$

Tiếp theo, ta tính tần suất xuất hiện có trọng của giá trị YES và giá trị NO cho cột Block Arteries. Giá trị nào có tần suất xuất hiện lớn hơn, ta sẽ điền nó vào ô dữ liệu bị khuyết:

- Tần suất xuất hiện của YES: $\frac{1}{3} \times 0.1 = 0.03$
- Tần suất xuất hiện của NO: $\frac{2}{3} \times 0.1 = 0.6$

Vậy, giá trị NO có tần suất xuất hiện có trọng số cao hơn, ta sẽ **điền giá trị NO vào cột Blocked Arteries cho mẫu 4**.

Còn đối với dữ liệu bị thiếu ở cột Weight, ta sẽ sử dụng giá trị lân cận để tính toán được cân nặng trung bình. Đầu tiên, ta vẫn cần tính trọng số từ mẫu 1 đến mẫu 3:

- Trọng số của mẫu 1: $\frac{0.1}{0.1 + 0.1 + 0.8} = 0.1$
- Trọng số của mẫu 2: $\frac{0.1}{0.1 + 0.1 + 0.8} = 0.1$
- Trọng số của mẫu 3: $\frac{0.8}{0.1 + 0.1 + 0.8} = 0.8$

Vậy trung bình cân nặng là
$(125 \times 0.1) + (180 \times 0.1) + (210 \times 0.8) = 198.5$. Ta sẽ **điền giá trị 198.5 vào cột Weight cho mẫu 4.**

2.4.2.2. Điền dữ liệu bị thiếu cho mẫu mới muốn phân loại

Bây giờ, ta có một mẫu mới muốn phân loại nó có bị Heart Disease hay không. Tuy nhiên, cột Blocked Arteries lại bị thiếu mất dữ liệu:

Chest Pain	Good Blood Circulation	Blocked Arteries	Weight	Heart Disease

No	No	???	168	

Bảng 2.12. Ví dụ - Mẫu mới cần phân loại có dữ liệu bị khuyết

Đầu tiên, ta sẽ copy ra 2 dòng dữ liệu, một dòng sẽ có giá trị của Heart Disease là YES, dòng còn lại sẽ có giá trị của Heart Disease là NO.

No.	Chest Pain	Good Blood Circulation	Blocked Arteries	Weight	Heart Disease
1	No	No	???	168	**YES**
2	No	No	???	168	**NO**

Bảng 2.13. Ví dụ - 2 dòng sao chép từ mẫu mới đã điền biến target

Dựa vào Bảng 1.1, ta thấy rằng nếu bệnh nhân có Heart Disease được phân loại là YES thì thường Blocked Arteries của họ sẽ là YES, còn Heart Disease được phân lại là NO thì thường Blocked Arteries của họ sẽ là NO. Thế ta được dự đoán cột Blocked Arteries của 2 dòng dữ liệu trên sẽ là:

No.	Chest Pain	Good Blood Circulation	Blocked Arteries	Weight	Heart Disease
1	No	No	**YES**	168	**YES**
2	No	No	**NO**	168	**NO**

Bảng 2.14. Ví dụ - 2 dòng sao chép từ mẫu mới đã điền đầy đủ dữ liệu

Sau đó, ta sẽ tiếp tục lấy 2 dòng dữ liệu này để chạy qua tất cả các cây trong Random Forest của ta. Giả sử đối với dòng dữ liệu 1, cột Heart Disease được gán nhãn đúng là YES với 5/6 cây. Còn đối với dòng dữ liệu 2, cột Heart Disease chỉ được gán nhãn đúng là NO với 2/6 cây. Vậy ta sẽ **chọn dòng dữ liệu 1** để điền vào dữ liệu bị khuyết cho mẫu mới.

2.4.2.3. Trực quan hóa bằng Heatmap hoặc phương pháp MDS

Để có được ma trận khoảng cách (Distance Matrix) từ ma trận lân cận, ta chỉ cần đơn giản lấy 1 - các giá trị lân cận, ta sẽ thu được ma trận khoảng cách. Ta có thể sử dụng ma trận khoảng cách này để trực quan hóa bằng Heatmap hoặc MDS.

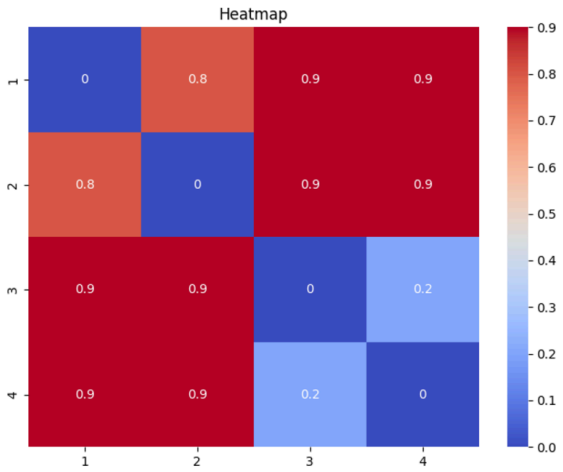
	1	2	3	4
1		0.8	0.9	0.9

2	0.8		0.9	0.9
3	0.9	0.9		0.2
4	0.9	0.9	0.2	

Bảng 2.15. Ví dụ - Ma trận khoảng cách

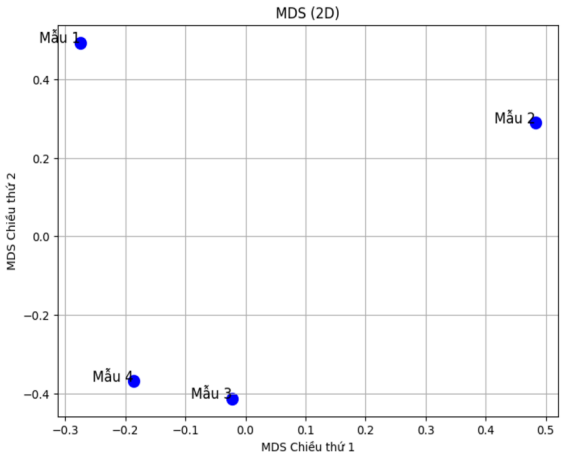
Bên dưới là hình ảnh minh họa

Heatmap



Hình 2.4. Biểu đồ Heatmap

Phương pháp MDS



Hình 2.5. Biểu đồ điểm - PP MDS

2.5. Random Forest và hiện tượng Overfitting

Random Forest ít bị overfitting khi tăng số lượng cây B vì mỗi cây độc lập với nhau và trung bình kết quả giúp giảm sai số. Lý do khiến Random Forest hạn chế overfitting là:

- Mỗi cây trong Random Forest được xây dựng dựa trên một mẫu bootstrap từ dữ liệu gốc. Điều này tạo ra sự đa dạng trong các cây. Khi các cây đưa ra các dự đoán độc lập, việc lấy trung bình (đối với hồi quy) hoặc bỏ phiếu đa số (đối với phân loại) giúp giảm phương sai của mô hình tổng thể.
- Thay vì sử dụng toàn bộ các biến để tìm ra điều kiện chia tốt nhất tại mỗi nút, Random Forest chỉ chọn ngẫu nhiên một tập con các biến. Giá trị m nhỏ (số biến được chọn ngẫu nhiên tại mỗi lần chia) giúp giảm khả năng các cây đều

tập trung vào các biến mạnh nhất, từ đó làm tăng sự đa dạng giữa các cây và hạn chế overfitting.

Mặc dù Random Forest thường chống overfitting tốt, vẫn có một số tình huống có thể xảy ra overfitting:

- Nếu dữ liệu huấn luyện không đủ đa dạng hoặc có quá nhiều nhiễu, các cây vẫn có thể học những đặc điểm không hữu ích.
- Nếu mỗi cây trong rừng được phát triển đến độ sâu tối đa (fully grown), chúng có thể học hết nhiễu trong dữ liệu. Tuy nhiên, như đã nói ở trên, việc trung bình các cây sẽ giúp giảm tác động này. Kiểm soát độ sâu của cây có thể giảm nhẹ overfitting.

Rừng ngẫu nhiên là một mô hình mạnh mẽ chống overfitting tốt nhờ việc kết hợp nhiều cây độc lập. Tuy nhiên, việc kiểm soát các tham số như độ sâu cây và số biến được chọn ngẫu nhiên là cần thiết để đảm bảo mô hình đạt hiệu suất tốt nhất trên dữ liệu mới.

3. THỰC NGHIỆM

Nhóm tiến hành sử dụng thuật toán Random Forest để giải quyết bài toán gán nhãn cho bộ dữ liệu *"Hotel Booking Cancellation Prediction"* thu thập từ Kaggle, sử dụng ngôn ngữ lập trình Python và môi trường Colab.

Tổng quan: Bộ dữ liệu Hotel Booking Cancellation Prediction trên Kaggle chứa thông tin về việc đặt phòng khách sạn với biến mục tiêu là tình trạng đặt phòng (*"Cancel"* và *"Not_Canceled"*).

Kích thước: Bộ dữ liệu bao gồm 36284 dòng dữ liệu và 17 thuộc tính.

STT	Tên biến	Mô tả	Loại biến	Kiểu dữ liệu
1	**Booking_ID**	Mã định danh cho mỗi lượt đặt phòng	Biến độc lập	Định danh
2	**number of adults**	Số lượng người trưởng thành trong một lượt đặt phòng	Biến độc lập	Định lượng

3	**number of children**	Số lượng trẻ em trong một lượt đặt phòng	Biến độc lập	Định lượng
4	**number of weekend nights**	Số đêm cuối tuần trong một lượt đặt phòng	Biến độc lập	Định lượng
5	**number of week nights**	Số đêm trong tuần (không tính cuối tuần) trong một lượt đặt phòng	Biến độc lập	Định lượng
6	**type of meal**	Loại bữa ăn đi kèm lượt đặt phòng đó	Biến độc lập	Định danh
7	**car parking space**	Không gian đỗ xe (có hoặc không)	Biến độc lập	Định danh
8	**room type**	Loại phòng đã đặt	Biến độc lập	Định danh
9	**lead time**	Thời gian đặt trước	Biến độc lập	Định lượng
10	**market segment type**	Kênh đặt phòng	Biến độc lập	Định danh
11	**repeated**	Khách hàng đã từng đặt trước đó hay chưa (có hoặc không)	Biến độc lập	Định danh
12	**P-C**	Trước lần đặt phòng hiện tại, khách hàng này đã từng hủy phòng bao nhiêu lần	Biến độc lập	Định lượng
13	**P-not-C**	Trước lần đặt phòng hiện tại, khách hàng này đã từng đặt phòng (và không hủy) bao nhiêu lần	Biến độc lập	Định lượng
14	**average price**	Giá tiền trung bình mỗi đêm mà khách hàng phải trả cho một lượt đặt phòng	Biến độc lập	Định lượng
15	**special requests**	Số yêu cầu đặc biệt khác mà khách hàng đưa ra	Biến độc lập	Định lượng
16	**date of reservation**	Ngày khách hàng đặt phòng	Biến độc lập	Định tính
17	**booking status**	Tình trạng đặt phòng (đã hủy hoặc chưa hủy)	Biến phụ thuộc	Định danh

3.1. Tiền xử lý dữ liệu

3.1.1. Xử lý các cột dữ liệu:

Dễ thấy được rằng có một biến có tên "Booking_ID" chứa những mã định danh cho mỗi lượt đặt phòng. Tuy nhiên, thông qua kiểm tra, những giá trị này có vẻ như là không có ý nghĩa về mặt thống kê.

```
print(data["Booking_ID"].duplicated().sum())
print(data["Booking_ID"].nunique())
```

Kết quả:

```
0
36285
```

Có thể thấy được rằng những giá trị“"Booking_ID"”chỉ xuất hiện một lần và không hề xuất hiện lại. Vậy nên những giá trị mà nó biểu diễn không có ý nghĩa gì ngoài liệt kê thứ tự các đơn đặt phòng và điều này thì không mang nhiều mục đích nghiên cứu. Vì thế về cột "Booking_ID" thì sẽ loại bỏ ra khỏi mô hình không mang ý nghĩa thống kê.

```
data.drop(["Booking_ID"], axis=1, inplace=True)
data.index = data.index + 1
print(data.shape)
data.head()
```

(36285, 16)

	number of adults	number of children	number of weekend nights	number of week nights	type of meal	car parking space	room type	lead time	market segment type	repeated	P-C	P-not-C	average price	special requests	date of reservation	booking status
1	1	1	2	5	Meal Plan 1	0	Room_Type 1	224	Offline	0	0	0	88.00	0	10/2/2015	Not_Canceled
2	1	0	1	3	Not Selected	0	Room_Type 1	5	Online	0	0	0	106.68	1	11/6/2018	Not_Canceled
3	2	1	1	3	Meal Plan 1	0	Room_Type 1	1	Online	0	0	0	50.00	0	2/28/2018	Canceled
4	1	0	0	2	Meal Plan 1	0	Room_Type 1	211	Online	0	0	0	100.00	1	5/20/2017	Canceled
5	1	0	1	2	Not Selected	0	Room_Type 1	48	Online	0	0	0	77.00	0	4/11/2018	Canceled

3.1.2. Xử lý Missing Data:

```
_column_totals = data.notnull().sum()
values_missing = _column_totals.loc[_column_totals < data.shape[0]]
```

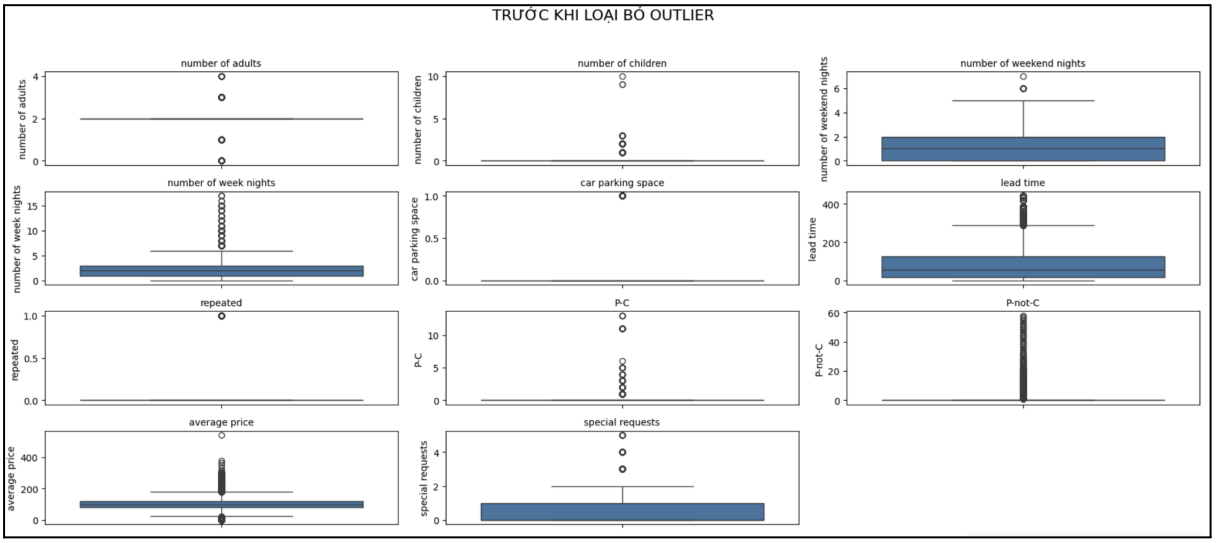
```
print(values_missing)
```

Kết quả:

```
Series([], dtype: int64)
```

Có thể thấy kết quả trả về không có thuộc tính nào có missing values. Do đó, chúng ta sẽ bỏ qua phần xử lý này.

3.1.3. Xử lý Outliers:

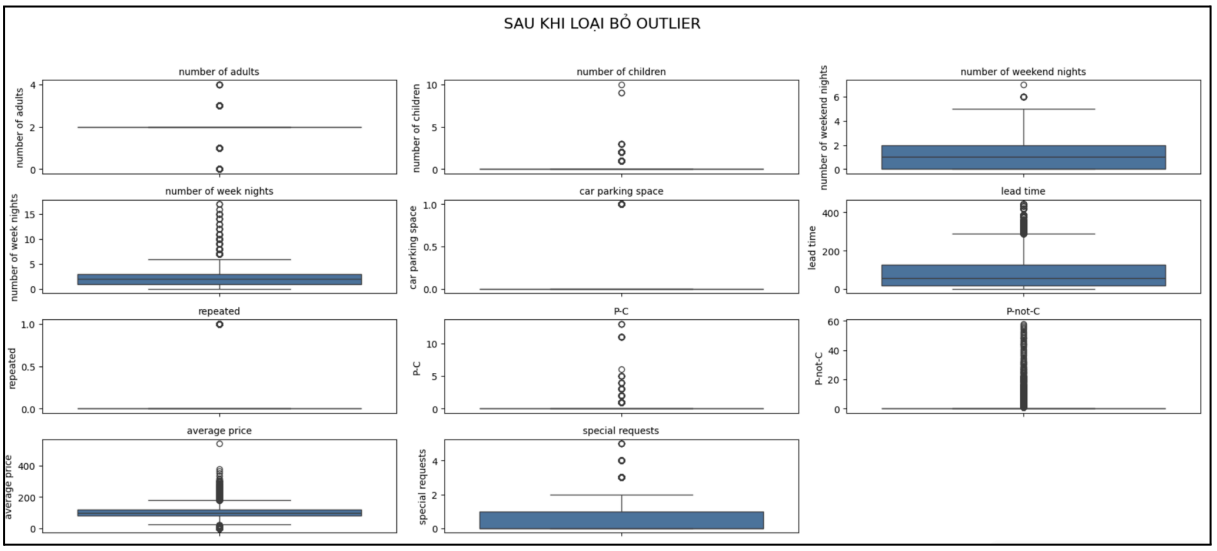


Nhìn chung, giá trị ngoại lai xuất hiện ở khá nhiều thuộc tính. Tuy nhiên, xét trên tập dữ liệu, có nhiều cột là biến nhị phân (‘repeated’, ‘car parking space’), biến có ít điểm dữ liệu (‘number of adults’, ‘number of children’), biến có độ lệch mạnh(‘P-C’, ‘P-not-C’).Vậy nên chúng ta sẽ phải cân nhắc khi xử lý những giá trị quá thưa thớt và giữ lại những giá trị khác để đảm bảo sự nguyên vẹn cho bộ dữ liệu. Trong bối cảnh của bộ dữ liệu, các biến ‘lead time’ và ‘average price’ sẽ được thực hiện bỏ lược bỏ giá trị ngoại lai.

```
outliers_cols = ["lead time", "average price"]
for column in outliers_cols:
    if data[column].dtype in ["int64", "float64"]:
        q1 = data[column].quantile(0.25)
        q3 = data[column].quantile(0.75)
        iqr = q3 - q1
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr
```

```
        data = data[
            (data[column] >= lower_bound) & (data[column] <= upper_bound)
        ]
numeric_columns = data.select_dtypes(include=['float64', 'int64']).columns
plt.figure(figsize=(18, 12))
plt.suptitle('SAU KHI LOẠI BỎ OUTLIER', fontsize=16, y=0.95)

# Vẽ từng boxplot
for i, var in enumerate(numeric_columns):
    plt.subplot(6, 3, i + 1)
    sbn.boxplot(y=data[var])
    plt.title(f'{var}', fontsize=10)

plt.tight_layout(rect=[0, 0, 1, 0.93])
plt.show()
```

Kết quả:



3.2. Chuẩn hóa dữ liệu

Đầu tiên, nhóm xử lý thuộc tính "date of reservation" bằng cách chuyển thành 3 thuộc tính mới là "day", "month", "year".

```
data = data[~data["date of reservation"].str.contains("-")]
```

```
data["date of reservation"] = pd.to_datetime(data["date of reservation"])

data["day"] = data["date of reservation"].dt.day
data["month"] = data["date of reservation"].dt.month
data["year"] = data["date of reservation"].dt.year

# Drop the original datetime column
data = data.drop(columns=["date of reservation"])
```

Tiếp theo, nhóm xử lý cột biến phụ thuộc "booking status" bằng cách chuyển các giá trị về 0 ; 1.

```
data["booking status"] = data["booking status"].replace("Canceled", 1)
data["booking status"] = data["booking status"].replace("Not_Canceled", 0)
```

Cuối cùng, nhóm thực hiện Encoding cho các biến categorical bằng phương thức get_dummies

```
object_columns = data.select_dtypes(include=["object"]).columns
data = pd.get_dummies(data, columns=object_columns)
data = data.replace({True: 1, False: 0})
```

3.3. Giảm chiều dữ liệu:

3.3.1. Xác định các cặp thuộc tính tương quan mạnh với nhau

Đầu tiên, thực hiện tính toán hệ số tương quan giữa các đặc trưng trong bộ dữ liệu (f_df) bằng cách sử dụng "Spearman correlation". Từ đó, tìm các cặp đặc trưng có hệ số tương quan lớn hơn 0.7 (hoặc nhỏ hơn -0.7). Các cặp đặc trưng có hệ số tương quan mạnh sẽ được lưu vào danh sách correlation_pairs. Việc phát hiện và loại bỏ các đặc trưng tương quan mạnh giúp giảm độ phức tạp của mô hình và tránh hiện tượng overfitting.

```
mx = f_df.corr(method='spearman')
correlation_pairs = []
for i in range(len(mx.columns)):
    for j in range(i + 1, len(mx.columns)):
        col1 = mx.columns[i]
        col2 = mx.columns[j]
        correlation_value = mx.iloc[i, j]
        if abs(correlation_value) > 0.7:
            correlation_pairs.append((col1, col2, correlation_value))

# Hiển thị các cặp cột có tương quan lớn hơn 0.7
for col1, col2, value in correlation_pairs:
    print(f"Cặp features: {col1} và {col2}, Hệ số tương quan: {value:.2f}")
```

Kết quả:

```
Cặp features: repeated và P-not-C, Hệ số tương quan: 0.93
Cặp features: type of meal_Meal Plan 1 và type of meal_Not Selected, Hệ số tương quan: -0.78
Cặp features: room type_Room_Type 1 và room type_Room_Type 4, Hệ số tương quan: -0.88
Cặp features: market segment type_Offline và market segment type_Online, Hệ số tương quan: -0.86
```

3.3.2. Lựa chọn các đặc trưng quan trọng nhất

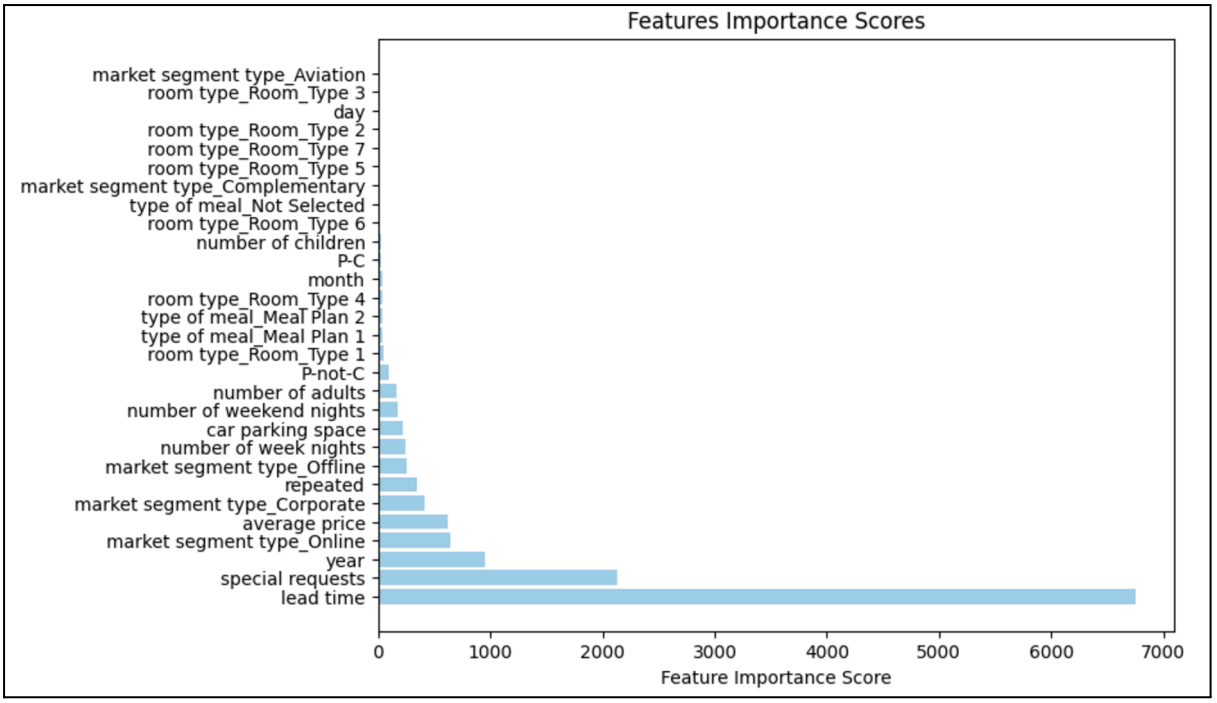
Tiếp theo, nhóm sẽ chọn ra các thuộc tính có ảnh hưởng lớn nhất đến mục tiêu. Phương pháp F-test được sử dụng để bỏ các thuộc tính ít quan trọng và chỉ giữ lại những thuộc tính có sự ảnh hưởng mạnh đến dự đoán. Trong tập dữ liệu này, F-test kiểm tra mức độ quan trọng của từng thuộc tính đối với mục tiêu, từ đó chọn ra 10 thuộc tính có mối quan hệ mạnh mẽ nhất đối với biến mục tiêu. Do đó, số lượng thuộc tính giảm đi nhiều mà không làm mất đi thông tin quan trọng, giúp mô hình học nhanh hơn và giảm độ phức tạp.

```
k_best = SelectKBest(score_func=f_classif, k=10) #dùng F-test đánh giá mức độ tương quan của mỗi feature với target
```

```
X_ = k_best.fit_transform(X, y)
y_ = y

# Get the indices of the selected features
selected_features_indices = k_best.get_support(indices=True)

# Get the scores associated with each feature
feature_scores = k_best.scores_

# Create a list of tuples containing feature names and scores
feature_info = list(zip(X.columns, feature_scores))

# Sort the feature info in descending order based on scores
sorted_feature_info = sorted(feature_info, key=lambda x: x[1], reverse=True)

for feature_name, feature_score in sorted_feature_info[:10]:
    print(f"{feature_name}: {feature_score:.2f}")
```

Kết quả trả về:

```
lead time: 6755.25
special requests: 2136.14
year: 952.07
market segment type_Online: 646.78
average price: 617.23
market segment type_Corporate: 414.32
repeated: 343.90
market segment type_Offline: 250.21
number of week nights: 248.88
car parking space: 216.60
```



3.3.3. Xử lý các thuộc tính bị loại bỏ

Sau khi chọn được các đặc trưng quan trọng, nhóm tiến hành trích xuất dữ liệu các thuộc tính từ chỉ số của chúng (selected_features_indices). Thuộc tính market segment type_Online bị loại khỏi bộ dữ liệu vì nó có sự tương quan cao với market segment type_Offline, điều này có thể dẫn đến vấn đề đa cộng tuyến và ảnh hưởng xấu đến mô hình.

```
selected_features_df = X.iloc[:, selected_features_indices]
selected_features_df = selected_features_df.drop(['market segment type_Online'],
axis=1) #do 2 features market segment type_Offline và market segment type_Online
cso sự tương quan cao
selected_features_df.head(10)
```

3.4. Xây dựng và đánh giá mô hình

Danh sách ensemble_clfs lưu trữ các mô hình RandomForestClassifier với các giá trị khác nhau của tham số max_features, lần lượt là 2, 3, 4, 5, 6, 7, 8, None. Mỗi phần tử trong danh sách là một tuple gồm:

- (label): mô tả giá trị của tham số max_features.

- (clf): mô hình RandomForestClassifier được xây dựng với các tham số cụ thể. Tham số oob_score = True để trả về điểm OOB của mô hình sau khi huấn luyện. Tham số max_features xác định số lượng đặc trưng được chọn ngẫu nhiên tại mỗi node để phân chia của các cây trong rừng ngẫu nhiên.

```
ensemble_clfs = [
    (
        "RandmForestClassifier, max_features= 2",
        RandomForestClassifier(
            oob_score=True,
            max_features=2,
            random_state=42,
        ),
    ),
    (
        "RandomForestClassifier, max_features= 3",
        RandomForestClassifier(
            max_features=3,
            oob_score=True,
            random_state=42,
        ),
    ),
    (
        "RandomForestClassifier, max_features= 4",
        RandomForestClassifier(
            max_features=4,
            oob_score=True,
            random_state=42,
        ),
    ),
    (
        "RandomForestClassifier, max_features= 5",
        RandomForestClassifier(
            max_features=5,
```

```
        oob_score=True,
        random_state=42,
    ),
),
(
    "RandomForestClassifier, max_features=6",
    RandomForestClassifier(
        max_features=6,
        oob_score=True,
        random_state=42,
    ),
),
(
    "RandomForestClassifier, max_features=7",
    RandomForestClassifier(
        max_features=7,
        oob_score=True,
        random_state=42,
    ),
),
(
    "RandomForestClassifier, max_features=8",
    RandomForestClassifier(
        max_features=8,
        oob_score=True,
        random_state=42,
    ),
),
 (
    "RandomForestClassifier, max_features= None",
    RandomForestClassifier(
        max_features=None,
        oob_score=True,
```

```
            random_state=42,
        ),
    ),
]
```

param_space: định nghĩa không gian tìm kiếm cho siêu tham số khi tạo cây quyết định trong quá trình xây dựng rừng ngẫu nhiên.

```
param_space = {
    'max_depth': (1, 20),
    'criterion': ['gini', 'entropy', 'log_loss']
}
```

Khởi tạo một danh sách OrderedDict: error_rate để lưu trữ các OOB_error cho từng mô hình RandomForestClassifier trong ensemble_clfs với khóa là tên các mô hình (label). Danh sách này sẽ lưu các OOB_error cho từng giá trị của tham số n_estimators cho mỗi mô hình.

```
error_rate = OrderedDict((label, []) for label, _ in ensemble_clfs)
```

Tạo giá trị bắt đầu và kết thúc cho vòng lặp tối ưu tham số n_estimaters

```
min_estimators = 10
max_estimators = 150
```

Tối ưu hóa các siêu tham số của mỗi cây quyết định trong từng mô hình random forest bằng Bayesian Optimization (sử dụng BayesSearchCV). Quá trình này sử dụng không gian tham số (param_space) đã được định nghĩa ở , với n_iter=32 cho số lần tối ưu hóa, và cv=3: sử dụng 3-fold cross-validation để kiểm tra.

Sau khi quá trình tối ưu hóa hoàn tất, các tham số tốt nhất sẽ được in ra và lưu trữ cho mô hình đó.

```
for label, clf in ensemble_clfs:
```

```
    opt = BayesSearchCV(clf, param_space, n_iter=32, cv=3, random_state=42)

    opt.fit(X_train, y_train)

    best_params = opt.best_params_
    print(f"Best parameters for {label}: {best_params}")
```

Sau khi tối ưu hóa các siêu tham số cho việc xây dựng cây quyết định trong từng mô hình rừng ngẫu nhiên, mỗi mô hình vừa được tối ưu sẽ được lưu trữ tại biến optimized_clf và huấn luyện lại với các giá trị khác nhau của tham số n_estimators. Với mỗi giá trị n_estimators, mô hình được huấn luyện lại và OOB_error được tính toán bằng cách:

OOB_error = 1 - optimized_clf.oob_score_

Với oob_score_ là giá trị trả về khi tham số oob_score=True.

```
error_rate = OrderedDict((label, []) for label, _ in ensemble_clfs)

for label, clf in ensemble_clfs:

    opt = BayesSearchCV(clf, param_space, n_iter=32, cv=3, random_state=42)

    opt.fit(X_train, y_train)

    best_params = opt.best_params_
    print(f"Best parameters for {label}: {best_params}")

    optimized_clf = opt.best_estimator_

    for i in range(min_estimators, max_estimators + 1, 10):
        optimized_clf.set_params(n_estimators=i, **best_params)
        optimized_clf.fit(X_train, y_train)

        oob_error = 1 - optimized_clf.oob_score_
```

```
        error_rate[label].append((i, oob_error))
```

Sau khi huấn luyện xong cho tất cả các mô hình trong ensemble_clfs với các giá trị tối ưu trong param_space, OOB_error được in ra cho mỗi mô hình trong ensemble. Mỗi mô hình được biểu diễn bằng nhãn label và các OOB_error trong danh sách.

```
for label, errors in error_rate.items():
    print(f"{label}: {errors}")
```

Sau đó thực hiện việc chọn ra mô hình rừng ngẫu nhiên với các tham số max_feature, n_estimaters tối ưu nhất thông qua OOB_error thấp nhất và khi tăng n_estimaters, OOB_error không tăng .

```
lowest_error_label = None
lowest_error_rate = float('inf')
lowest_n_estimators = None

for label, errors in error_rate.items():

    min_error = min(errors, key=lambda x: x[1])
    min_n_estimators, min_error_rate = min_error

    if min_error_rate < lowest_error_rate:
        lowest_error_rate = min_error_rate
        lowest_error_label = label
        lowest_n_estimators = min_n_estimators

print(f"Label with the lowest error rate: {lowest_error_label}")
print(f"Lowest error rate: {lowest_error_rate:.4f} at n_estimators =
{lowest_n_estimators}")
```

Xây dựng mô hình RandomForestClassifier() với các tham số đã tối ưu.

```
clf_f  = RandomForestClassifier(n_estimators = 120, max_features = 3, criterion = 'gini', max_depth = 17, oob_score=True, random_state=42)
model  = clf_f.fit(X_train, y_train)
```

Sử dụng mô hình đã tạo để dự đoán cho tập X_test.

```
y_pred= model.predict(X_test)
```

Hàm classification_eval trả về các chỉ số đánh giá mô hình dựa trên giá trị thực tế (y_test) và giá trị dự đoán (y_pred).

```
def classification_eval(y_test, y_pred):
    accuracy  = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall    = recall_score(y_test, y_pred)
    f1        = f1_score(y_test, y_pred)
    return accuracy, precision, recall, f1
```

Đánh giá mô hình random forest bằng các chỉ số và hiển thị giá trị chỉ số.

```
scores = classification_eval(y_test, y_pred)

print(f'accuray   = {(scores[0] * 100):.1f}%')
print(f'precision = {(scores[1] * 100):.1f}%')
print(f'recall    = {(scores[2] * 100):.1f}%')
print(f'f1        = {(scores[3] * 100):.1f}%')
```

In ra ma trận nhầm lẫn dựa trên kết quả thực tế và kết quả dự đoán và vẽ biểu đồ nhiệt cho kết quả đó.

```
print(confusion_matrix(y_test, y_pred))
plt.figure(figsize=(8, 6))
sbn.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues", fmt=".0f", linewidths=0.5)
plt.title("Correlation Heatmap")
```

```
plt.show()
```

3.5. Kết quả thu được

Tập dữ liệu Booking.csv sau khi được xử lí để huấn luyện mô hình.

10 dòng dữ liệu đầu tiên:

	number of week nights	car parking space	lead time	repeated	average price	special requests	year	market segment type_Corporate	market segment type_Offline	booking status
1	5	0	224	0	88.00	0	2015	0	1	0
2	3	0	5	0	106.68	1	2018	0	0	0
3	3	0	1	0	50.00	0	2018	0	0	1
4	2	0	211	0	100.00	1	2017	0	0	1
5	2	0	48	0	77.00	0	2018	0	0	1
7	4	0	34	0	107.55	1	2017	0	0	0
8	3	0	83	0	105.61	1	2018	0	0	0
9	4	0	121	0	96.90	1	2018	0	1	0
10	5	0	44	0	133.44	3	2018	0	0	0

Chia dữ liệu thành tập huấn luyện (training set) và tập kiểm tra (test set) theo tỉ lệ 20% dữ liệu sẽ được sử dụng làm tập kiểm tra, còn lại 80% làm tập huấn luyện.

Huấn luyện mô hình từ tập X_train và y_train.

Kết quả trả về sau khi thực hiện tối ưu hóa các siêu tham số của mỗi cây quyết định trong từng mô hình random forest bằng Bayesian Optimization (sử dụng BayesSearchCV).

```
Best parameters for RandomForestClassifier, max_features= 2:
OrderedDict([('criterion', 'gini'), ('max_depth', 17)])
```

```
Best parameters for RandomForestClassifier, max_features= 3:
OrderedDict([('criterion', 'gini'), ('max_depth', 17)])
```

```
Best parameters for RandomForestClassifier, max_features= 4:
OrderedDict([('criterion', 'entropy'), ('max_depth', 16)])
```

```
Best parameters for RandomForestClassifier, max_features= 5:
OrderedDict([('criterion', 'gini'), ('max_depth', 15)])
```

```
Best parameters for RandomForestClassifier, max_features=6:
OrderedDict([('criterion', 'entropy'), ('max_depth', 16)])
```

Best parameters for RandomForestClassifier, max_features=7: OrderedDict([('criterion', 'entropy'), ('max_depth', 15)])
Best parameters for RandomForestClassifier, max_features=8: OrderedDict([('criterion', 'gini'), ('max_depth', 15)])
Best parameters for RandomForestClassifier, max_features= None: OrderedDict([('criterion', 'gini'), ('max_depth', 15)])

Mô hình RandomForestClassifier được lưu trữ lại với mỗi giá trị của tham số max_features và các giá trị tối ưu của tham số 'criterion' và 'max_depth' sau khi khi sử dụng BayesSearchCV. Sau đó, tiếp tục được huấn luyện lại với các giá trị của tham số n_estimaters.

Kết quả được in ra dưới dạng:

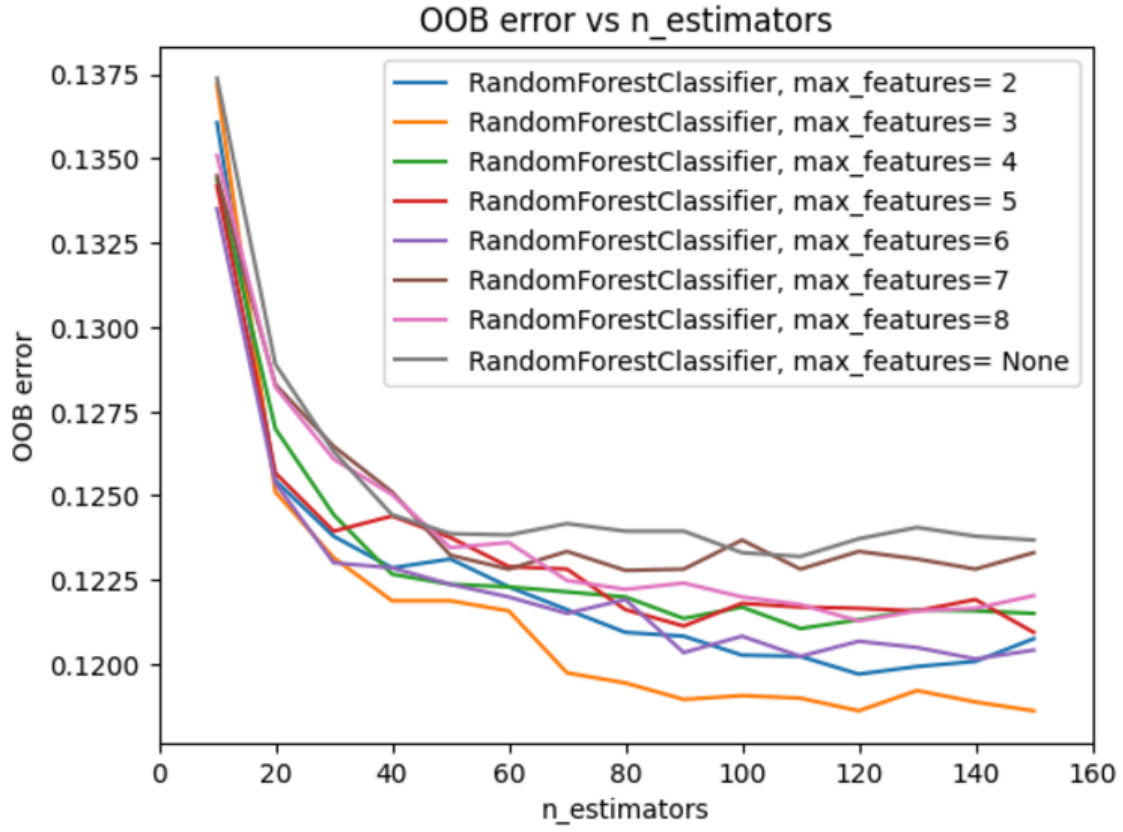
RandomForestClassifier, max_features= (giá tri cụ thể): [(N_estimaters, OOB_error).

RandomForestClassifier, max_features= 2: [(10, 0.13606514315734175), (20, 0.12544560771511126), (30, 0.12379451386543583), (40, 0.1228563923599385), (50, 0.12311906638147774), (60, 0.12229351945664002), (70, 0.12161807197268193), (80, 0.12094262448872373), (90, 0.12083004990806412), (100, 0.12026717700476564), (110, 0.12022965214454573), (120, 0.11970430410146726), (130, 0.11992945326278659), (140, 0.12007955270366621), (150, 0.1207550001876243)]
RandomForestClassifier, max_features= 3: [(10, 0.1371908889639386), (20, 0.12510788397313222), (30, 0.12315659124169764), (40, 0.12188074599422116), (50, 0.12188074599422116), (60, 0.12158054711246202), (70, 0.11974182896168717), (80, 0.11944163007992792), (90, 0.11895380689706936), (100, 0.11906638147772897), (110, 0.11899133175728915), (120, 0.1186160831550902), (130, 0.11921648091860859), (140, 0.11887875717662955), (150, 0.1186160831550902)]
RandomForestClassifier, max_features= 4: [(10, 0.1343765244474464), (20, 0.12698412698412698), (30, 0.12443243648917413), (40, 0.12266876805883897), (50, 0.12236856917707983), (60, 0.12229351945664002), (70, 0.1221434200157604),

(80, 0.12199332057488088), (90, 0.12135539795114259), (100, 0.12169312169312174), (110, 0.12105519906938345), (120, 0.12131787309092279), (130, 0.12161807197268193), (140, 0.12158054711246202), (150, 0.12150549739202221)]
RandomForestClassifier, max_features= 5: [(10, 0.134188900146347), (20, 0.1256707568764306), (30, 0.12394461330631545), (40, 0.12439491162895422), (50, 0.12375698900521592), (60, 0.1228939172201583), (70, 0.1228188674997186), (80, 0.12161807197268193), (90, 0.12113024878982326), (100, 0.12180569627378135), (110, 0.12169312169312174), (120, 0.12165559683290184), (130, 0.12158054711246202), (140, 0.12191827085444107), (150, 0.12094262448872373)]
RandomForestClassifier, max_features=6: [(10, 0.1335134526623888), (20, 0.12537055799467145), (30, 0.12300649180081802), (40, 0.1228563923599385), (50, 0.12236856917707983), (60, 0.12199332057488088), (70, 0.12150549739202221), (80, 0.12191827085444107), (90, 0.12034222672520545), (100, 0.12083004990806412), (110, 0.12022965214454573), (120, 0.1206799504671845), (130, 0.12049232616608507), (140, 0.12015460242410592), (150, 0.12041727644564526)]
RandomForestClassifier, max_features=7: [(10, 0.13448909902810613), (20, 0.12829749709182336), (30, 0.1264587789410484), (40, 0.12510788397313222), (50, 0.12323164096213746), (60, 0.1228188674997186), (70, 0.12334421554279706), (80, 0.12278134263949869), (90, 0.1228188674997186), (100, 0.12368193928477611), (110, 0.1228188674997186), (120, 0.12334421554279706), (130, 0.12311906638147774), (140, 0.1228188674997186), (150, 0.12330669068257716)]
RandomForestClassifier, max_features=8: [(10, 0.1350894967916244), (20, 0.12822244737138355), (30, 0.12608353033884945), (40, 0.1250328342526924), (50, 0.12345679012345678), (60, 0.12360688956433641), (70, 0.12248114375773955), (80, 0.12221846973620021), (90, 0.12240609403729974), (100, 0.12199332057488088), (110, 0.12176817141356144), (120, 0.12128034823070288), (130, 0.12158054711246202), (140, 0.12165559683290184), (150, 0.12203084543510079)]
RandomForestClassifier, max_features= None: [

(10, 0.13737851326503814), (20, 0.12889789485534164),
(30, 0.12630867950016889), (40, 0.12443243648917413),
(50, 0.12386956358587564), (60, 0.12383203872565574),
(70, 0.12416976246763478), (80, 0.12394461330631545),
(90, 0.12394461330631545), (100, 0.12330669068257716),
(110, 0.12319411610191755), (120, 0.12371946414499602),
(130, 0.12405718788697517), (140, 0.12379451386543583),
(150, 0.12368193928477611)]

So sánh các OOB_error để chọn được mô hình tối ưu nhất: Nếu OOB_Error giảm dần nhưng bão hòa (không giảm đáng kể khi tăng thêm cây), số lượng cây tại thời điểm đó được coi là đủ.



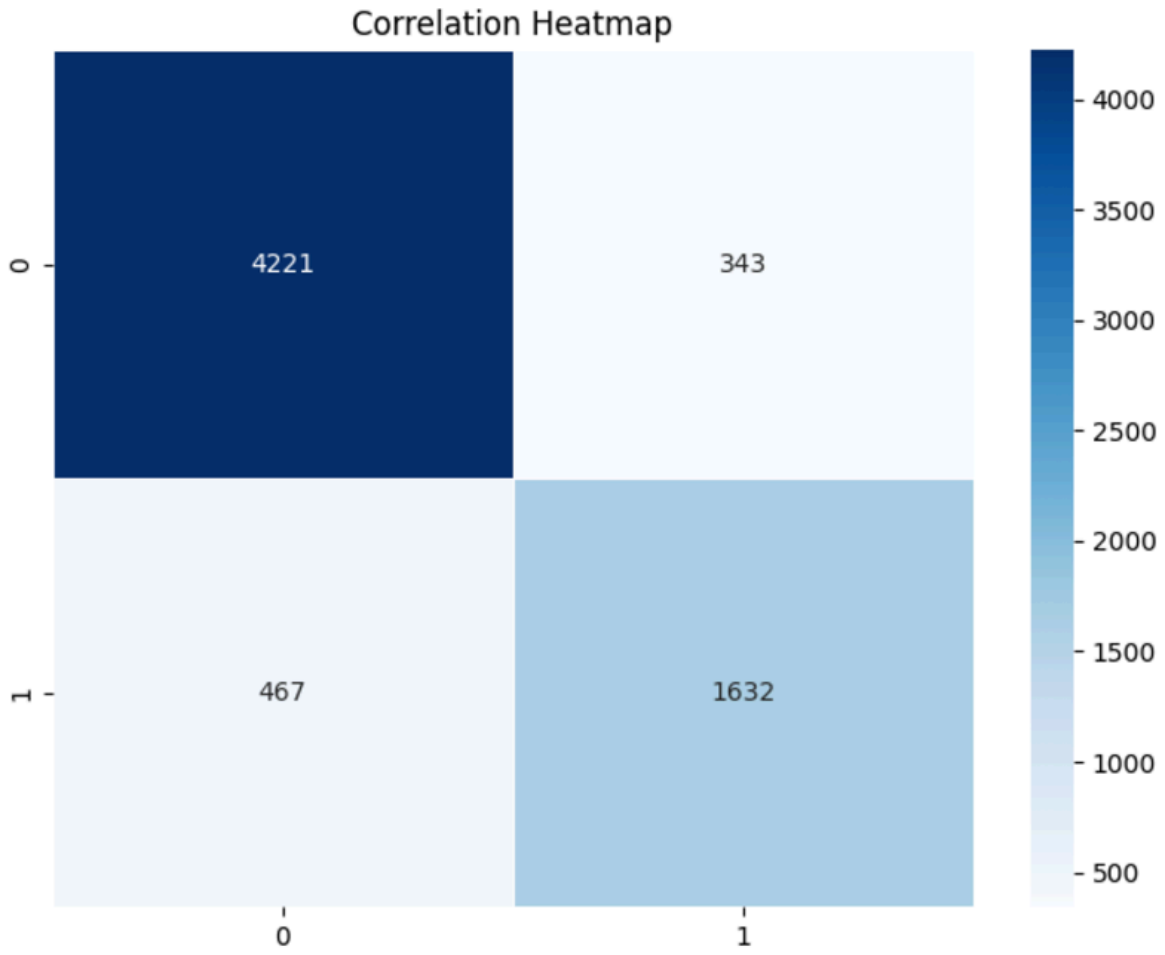
Dựa vào biểu đồ thể hiện tương quan giữa n_estimaters và OOB_error, mô hình RandomForestClassifier tối ưu nhất sẽ được xây dựng với các tham số và giá trị tương ứng sau:

- n_estimaters = 120
- max_features = 3

- criterion = 'gini'
- max_depth = 17

Sau khi huấn luyện mô hình tối ưu với tập X_test, tiến hành đánh giá mô hình dựa trên các chỉ số sau: accuray, precision, recall, f1-score và trực quan ma trận nhầm lẫn.

```
accuray   = 87.8%
precision = 82.6%
recall    = 77.8%
f1        = 80.1%
```



Trong ma trận nhầm lẫn trên, có 4421 dự đoán chính xác là Not Cancel và 1632 dự đoán chính xác là Cancel. Có 343 mẫu có nhãn là Not Cancel và 467 mẫu có nhãn là Cancel bị dự đoán sai.

4. TÀI LIỆU THAM KHẢO:

1. Random Forest, https://www.math.mcgill.ca/yyang/resources/doc/randomforest.pdf, truy cập lần cuối ngày 30/11/2024
2. Scikit-learn, OOB Errors fhttps://scikit-learn.org/1.5/auto_examples/ensemble/plot_ensemble_oob.html, truy cập lần cuối ngày 30/11/2024
3. https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html, truy cập lần cuối ngày 30/11/2024
4. https://www.kaggle.com/code/youssefaboelwafa/hotel-booking-cancellation-multiple-models, truy cập lần cuối ngày 30/11/2024