TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TIỂU LUẬN GIỮA KỲ

NHẬP MÔN HỌC MÁY

Người hướng dẫn: **PGS.TS. LÊ ANH CƯỜNG**
Người thực hiện: **LÊ HẢI TIẾN – 52101002**
HUỲNH THỊ TRÀ MY - 52100704
LÊ PHÚC ANH - 52300091
Khoá **: 25 & 27**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2025

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TIỂU LUẬN GIỮA KỲ

NHẬP MÔN HỌC MÁY

Người hướng dẫn: **PGS.TS. LÊ ANH CƯỜNG**
Người thực hiện: **LÊ HẢI TIẾN – 52101002**
HUỲNH THỊ TRÀ MY - 52100704
LÊ PHÚC ANH - 52300091
Khoá : **25 & 27**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2025

LỜI CẢM ƠN

Chúng tôi xin gửi lời cảm ơn chân thành đến Trường Đại học Tôn Đức Thắng và Khoa Công nghệ Thông tin đã tạo điều kiện cho chúng tôi được học tập và nghiên cứu trong một môi trường chuyên nghiệp, hiện đại với đầy đủ các nguồn tài nguyên phục vụ cho việc học tập và phát triển chuyên môn.

Chúng tôi đặc biệt bày tỏ lòng biết ơn sâu sắc đến PGS.TS. Lê Anh Cường, người đã tận tình hướng dẫn, cung cấp kiến thức và định hướng cho chúng tôi trong suốt quá trình thực hiện bài báo cáo này. Sự hướng dẫn của thầy không chỉ giúp chúng tôi hiểu rõ hơn về các mô hình học máy mà còn giúp chúng tôi rèn luyện tư duy phân tích, kỹ năng nghiên cứu và khả năng ứng dụng lý thuyết vào thực tiễn. Những góp ý và phản hồi của thầy trong từng giai đoạn nghiên cứu đã giúp chúng tôi nhận ra những điểm cần cải thiện, từ đó hoàn thiện bài tiểu luận một cách tốt nhất.

Ngoài ra, chúng tôi cũng xin gửi lời cảm ơn đến các giảng viên trong Khoa Công nghệ Thông tin đã truyền đạt những kiến thức nền tảng vững chắc, giúp chúng tôi có đủ khả năng tiếp cận với các lĩnh vực chuyên sâu như học máy. Các môn học và bài giảng của khoa đã cung cấp cho chúng tôi tư duy hệ thống, phương pháp nghiên cứu khoa học và cách tiếp cận các công nghệ hiện đại.

Bài báo cáo này là kết quả của quá trình học hỏi và nỗ lực không ngừng. Chúng tôi vô cùng trân trọng những cơ hội mà nhà trường, khoa và thầy cô đã mang lại. Một lần nữa, chúng tôi xin chân thành cảm ơn!

ĐỒ ÁN ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi chúng tôi và được sự hướng dẫn của PGS.TS. Lê Anh Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 23 tháng 3 năm 2025

Tác giả

(ký tên và ghi rõ họ tên)

Lê Hải Tiến

Huỳnh Thị Trà My

Lê Phúc Anh

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

TÓM TẮT

Báo cáo này trình bày về các biến thể của thuật toán Gradient Descent (Batch GD, SGD, Mini-Batch GD, Momentum, RMSprop, Adam) và ứng dụng vào dự đoán giá nhà, so sánh hiệu suất dựa trên RMSE. Ngoài ra, báo cáo cũng thực hiện một bài toán phân loại trên tập dữ liệu Adult Dataset từ UCI, sử dụng 5 mô hình Machine Learning khác nhau, đánh giá kết quả, xử lý overfitting và so sánh hiệu suất mô hình.

Bài báo cáo gồm có 2 chương:

Chương 1: So sánh và ứng dụng các biến thể của thuật toán Gradient Descent trong dự đoán giá nhà

Chương 2: So sánh các thuật toán machine learning trong bài toán phân loại và phân tích ảnh hưởng của overfitting

MỤC LỤC

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

DANH MỤC HÌNH

DANH MỤC BẢNG

CHƯƠNG 1 – So sánh và ứng dụng các biến thể của thuật toán Gradient Descent trong dự đoán giá nhà

1. Giới thiệu thuật toán

- Gradient Descent (GD) là một thuật toán tối ưu phổ biến, được sử dụng để tìm giá trị nhỏ nhất (local minimum hoặc global minimum) của một hàm số, đặc biệt trong các bài toán Machine Learning như tối ưu hóa hàm mất mát. Ý nghĩa chính của GD nằm ở việc mô phỏng quá trình "xuống dốc" trên đồ thị hàm số: từ một điểm khởi tạo ngẫu nhiên, thuật toán di chuyển dần về hướng mà độ dốc (gradient) nhỏ nhất, tức là nơi hàm số đạt giá trị thấp nhất. Phương pháp này đặc biệt hữu ích khi giải phương trình đạo hàm bằng 0 (để tìm cực trị) là bất khả thi do sự phức tạp của hàm hoặc số lượng dữ liệu lớn.

- Cơ chế của GD dựa trên việc lặp lại các bước sau:

1. **Khởi tạo**: Bắt đầu từ một điểm ngẫu nhiên x_0 trên đồ thị hàm số.
2. **Tính gradient**: Tính đạo hàm f′(x) tại điểm hiện tại, biểu thị hướng và độ dốc của hàm tại điểm đó. Gradient cho biết hướng tăng nhanh nhất của hàm, do đó di chuyển ngược gradient sẽ giảm giá trị hàm.
3. **Cập nhật**: Di chuyển điểm hiện tại theo hướng ngược gradient, với bước di chuyển được điều chỉnh bởi một tham số gọi là learning rate (η). Quá trình này lặp lại cho đến khi gradient tiến gần về 0 (tức là đến gần cực tiểu).
4. **Dừng**: Thuật toán dừng khi đạo hàm đủ nhỏ (thỏa mãn ngưỡng) hoặc sau một số lần lặp cố định.

   - Việc chọn learning rate (η) rất quan trọng: nếu quá nhỏ, quá trình hội tụ chậm; nếu quá lớn, có thể vượt qua cực tiểu hoặc không hội tụ.

2. Công thức toán học

- Đối với hàm một biến f(x), công thức cập nhật của Gradient Descent được biểu diễn như sau:

$$x_{t+1} = x_t - \eta f'(x_t)$$

trong đó:

- $\boldsymbol{x_t}$: Giá trị của biến tại bước lặp thứ t
- $\boldsymbol{x_{t+1}}$: Giá trị cập nhật tại bước lặp tiếp theo.
- $\boldsymbol{\eta}$: Learning rate (hằng số dương nhỏ, thường từ 0.01 đến 0.1).
- $\boldsymbol{f'(x_t)}$: Đạo hàm của hàm f(x) tại điểm x_t, thể hiện độ dốc.

- Quá trình này lặp lại cho đến khi $\left|f'(x_{t+1})\right| < \varepsilon$ (ϵ là ngưỡng nhỏ, ví dụ 1e-6) hoặc đạt số lần lặp tối đa.

3. Các biến thể của GDA

3.1. Batch GD

- Batch GD là phiên bản cơ bản của Gradient Descent, sử dụng toàn bộ tập dữ liệu huấn luyện để tính gradient tại mỗi bước lặp. Điều này đảm bảo hướng di chuyển chính xác, nhưng tốn nhiều tài nguyên tính toán khi tập dữ liệu lớn, và tốc độ hội tụ có thể chậm.

- Cơ chế của Batch GD:

+ Tính gradient của hàm mất mát J(θ) dựa trên toàn bộ dữ liệu (tổng trung bình gradient của tất cả mẫu).

+ Cập nhật tham số θ theo hướng ngược gradient với learning rate η.

+ Lặp lại cho đến khi hàm mất mát hội tụ (gradient gần 0) hoặc đạt số lần lặp tối đa.

- Công thức cập nhật tham số:

$$\theta = \theta - \eta \nabla J(\theta)$$

$$\nabla J(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla J_i(\theta)$$

trong đó:

+ $\boldsymbol{\theta}$: một tập hợp tham số của mô hình

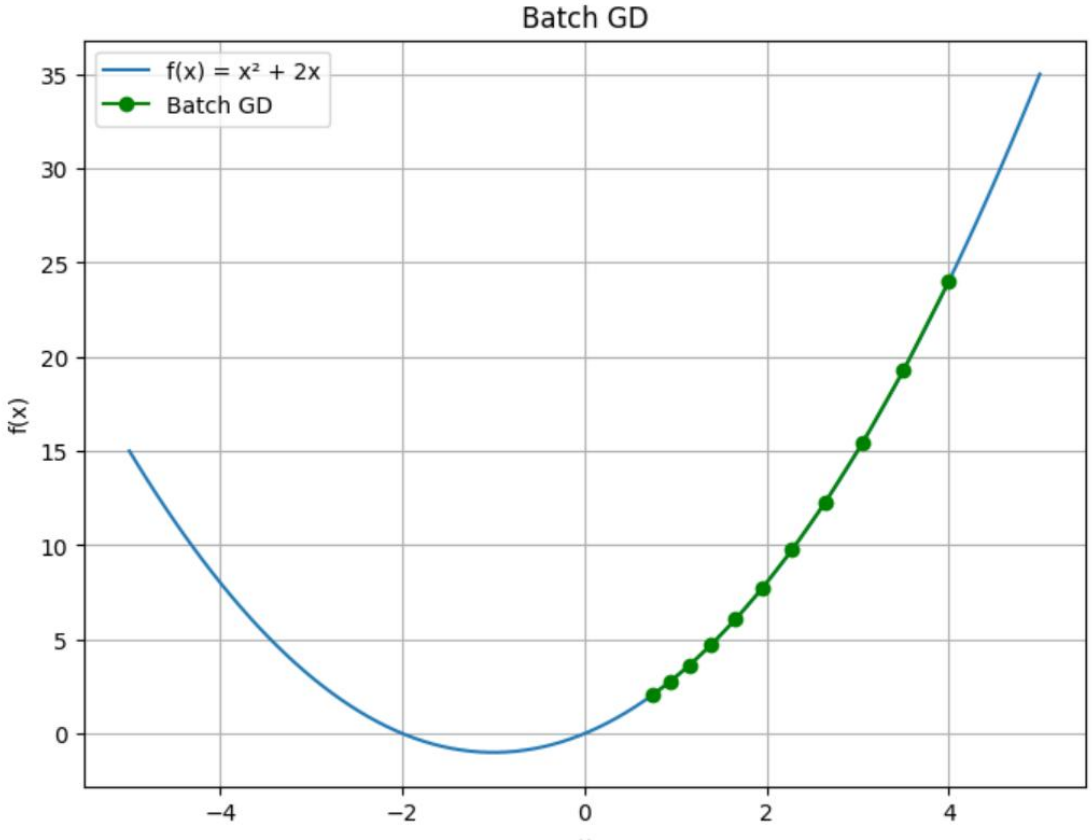
+ $\boldsymbol{\eta}$: tốc độ học

+ $\boldsymbol{\nabla J(\theta)}$: gradient trung bình

+ **n**: số mẫu dữ liệu

+ $\nabla J_i(\theta)$: gradient tính trên mẫu I

- Hình vẽ minh họa:



Hình ảnh 1. Minh họa Batch GD

3.2. Mini-Batch GD

- Mini-batch GD là một sự kết hợp giữa Batch GD và Stochastic GD, sử dụng một tập con (mini-batch) của dữ liệu tại mỗi bước lặp. Phương pháp này cân bằng giữa tốc độ tính toán và độ ổn định, thường được sử dụng trong thực tế, đặc biệt với các mô hình Deep Learning.

- Cơ chế của Mini-Batch GD:

+ Chia tập dữ liệu thành các mini-batch, ví dụ mỗi batch có 32 hoặc 64 mẫu

+ Tính gradient trung bình trên mini-batch

+ Cập nhật tham số θ dựa trên gradient của mini-batch

+ Lặp lại cho đến khi duyệt qua hết dữ liệu hoặc hội tụ

- Công thức toán học:

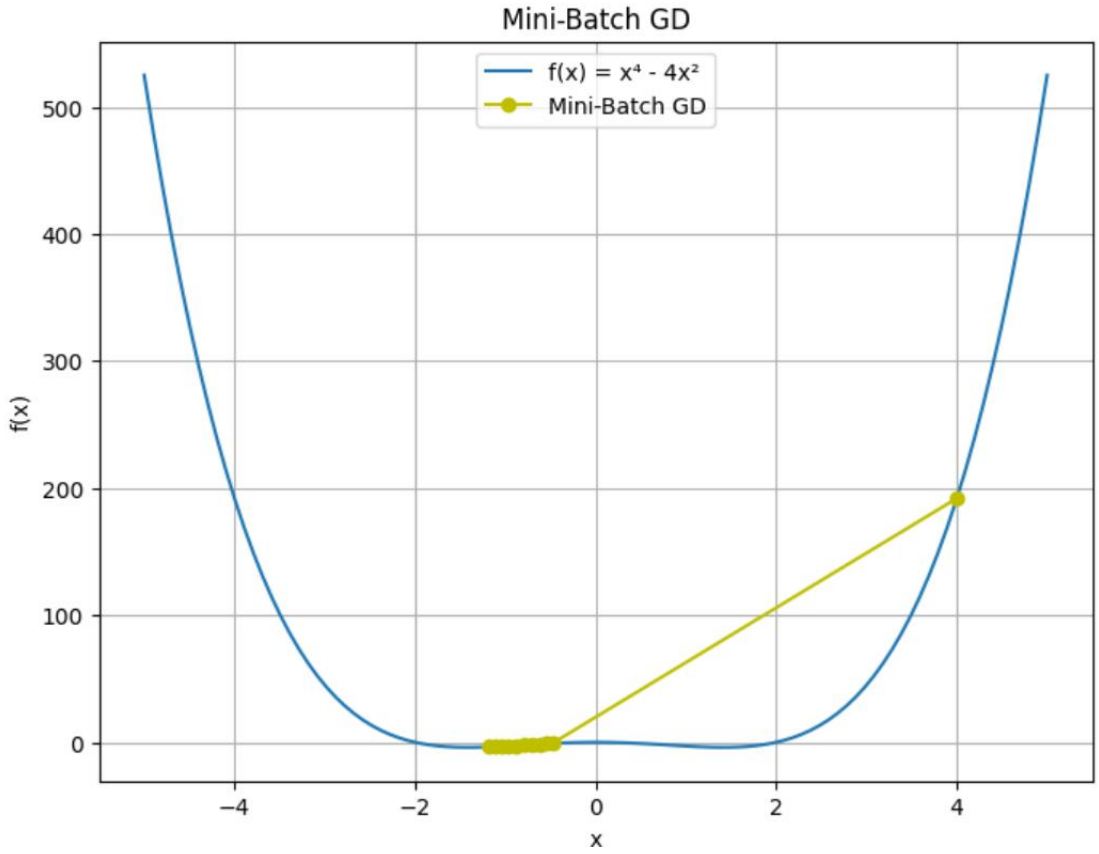
$$\theta = \theta - \eta \frac{1}{B} \sum_{i \in batch} \nabla J_i(\theta)$$

trong đó:

+ **B**: kích thước của mini-batch

+ $\sum_{i \in batch} \boldsymbol{\nabla J_i(\theta)}$: tổng gradient trên các mẫu trong mini-batch

- Hình vẽ minh họa:



Hình ảnh 2. Minh họa Mini-Batch GD

3.3. Stochastic Gradient Descent (Stochastic GD)

- Stochastic GD (SGD) là thuật toán khắc phục nhược điểm của Batch GD bằng cách chỉ sử dụng một mẫu dữ liệu ngẫu nhiên tại mỗi bước lặp để tính gradient. Điều này làm tăng tốc độ tính toán, nhưng hướng di chuyển có thể dao động mạnh, dẫn đến quá trình tối ưu không ổn định.

- Cơ chế thuật toán:

+ Chọn ngẫu nhiên một mẫu dữ liệu (x_i, y_i)

+ Tính gradient của hàm mất mát dựa trên mẫu đó

+ Cập nhật tham số θ theo hướng ngược gradient của mẫu được chọn

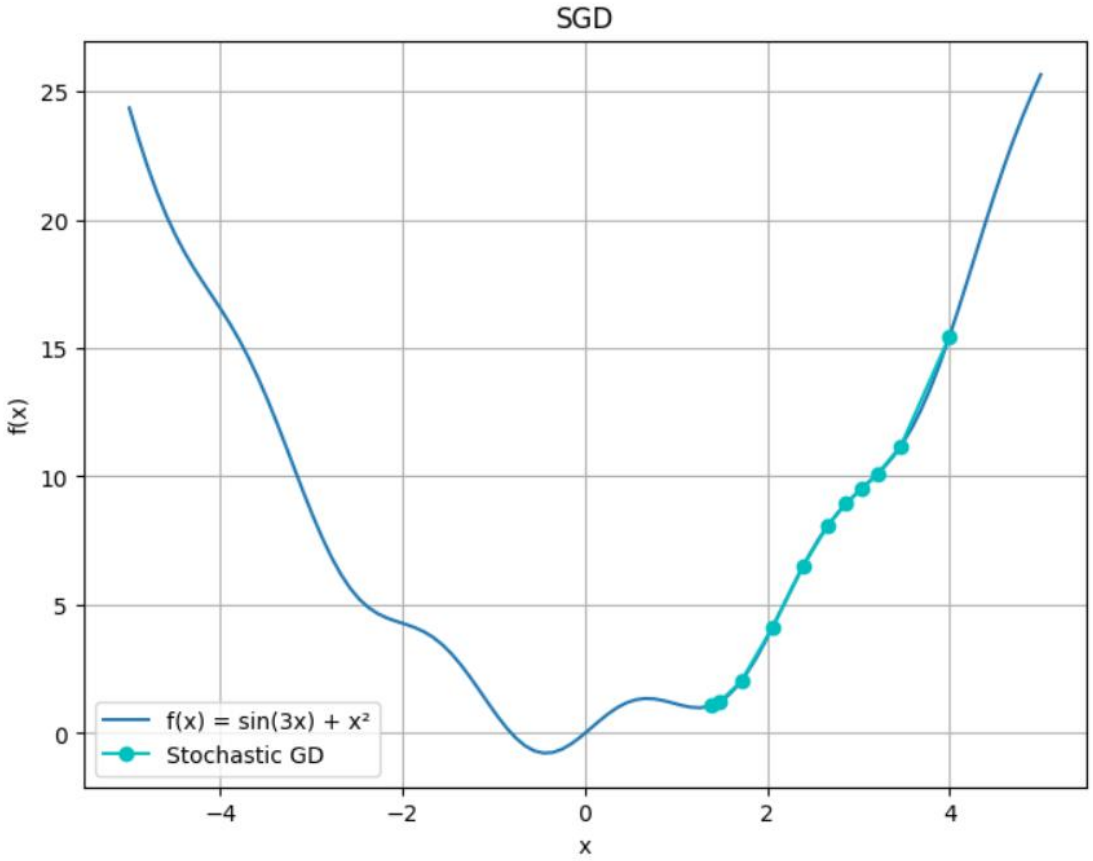
+ Lặp lại cho đến khi hội tụ hoặc đạt số lần lặp tối đa

- Công thức toán học:

$$\theta = \theta - \eta \nabla J_i(\theta)$$

với $\boldsymbol{\nabla J_i(\theta)}$ là gradient tính trên mẫu (x_i, y_i) được chọn ngẫu nhiên.

- Hình vẽ minh họa:



Hình ảnh 3. Minh họa SGD

3.4. Momentum

- Momentum cải tiến GD bằng cách thêm một thành phần "động lượng" để tăng tốc độ hội tụ, đặc biệt khi gradient dao động hoặc khi gặp các vùng dốc phức tạp. Nó mô phỏng quán tính vật lý, giúp thuật toán di chuyển nhanh hơn trong các vùng gradient ổn định và giảm dao động.

- Cơ chế hoạt động:

+ Sử dụng một vận tốc v để tích lũy gradient từ các bước trước

+ Cập nhật tham số dựa trên vận tốc này, thay vì chỉ dựa vào gradient hiện tại

+ Tham số γ (thường khoảng 0.9) điều chỉnh mức độ ảnh hưởng của gradient trước đó

- Công thức cập nhật:

$$\theta = \theta - v_t$$

$$v_t = \gamma v_{t-1} + \eta \nabla J(\theta)$$

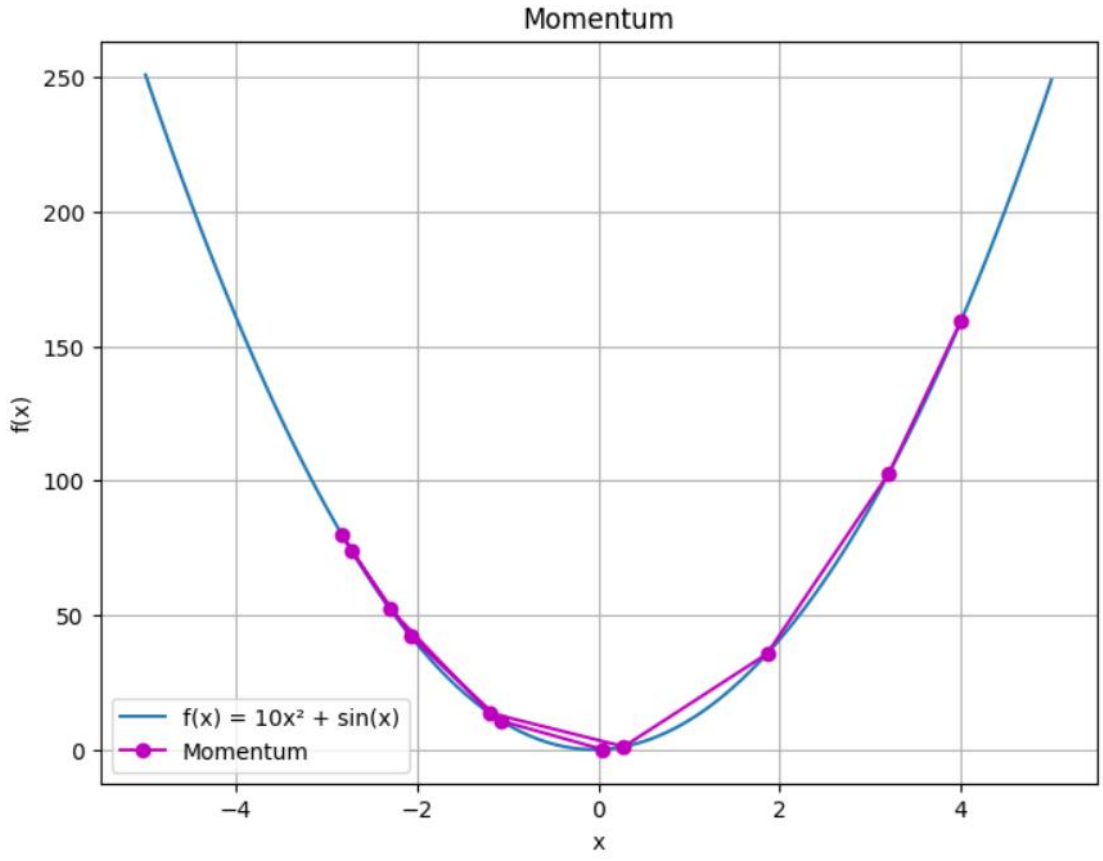
trong đó:

+ $\boldsymbol{v_t}$: vận tốc tại bước t

+ $\boldsymbol{\gamma}$: hằng số động lượng, nằm trong đoạn [0;1]

+ $\boldsymbol{\nabla J(\theta)}$: gradient tại bước hiện tại

- Hình vẽ minh họa



Hình ảnh 4. Minh họa Momentum

3.5. RMSprop (Root Mean Square Propagation)

- RMSprop là thuật toán cải thiện SGD bằng cách điều chỉnh learning rate cho từng tham số dựa trên trung bình lũy thừa bậc hai của gradient (RMS). Điều này giúp giảm dao động và tăng tốc độ hội tụ, đặc biệt hiệu quả với các bài toán có gradient thay đổi mạnh.

- Cơ chế hoạt động:

+ Tính trung bình lũy thừa bậc hai của gradient.

+ Sử dụng giá trị này để chuẩn hóa gradient, từ đó điều chỉnh bước cập nhật.

+ Thêm tham số ρ (thường khoảng 0.9) kiểm soát mức độ ảnh hưởng của gradient trước đó.

- Công thức toán học:

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)[\nabla J(\theta)]^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot \nabla J(\theta)$$

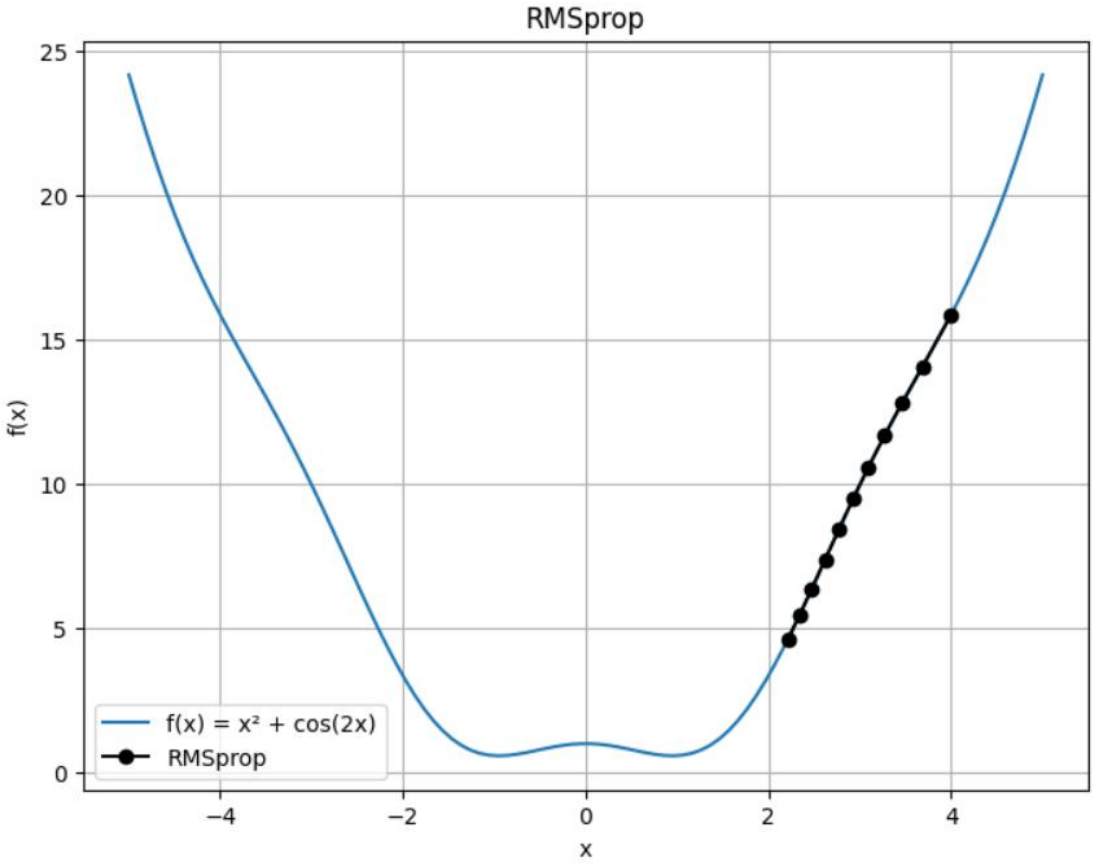
trong đó:

+ $\boldsymbol{E[g^2]_t}$**:** trung bình lũy thừa bậc hai của gradient tại bước t.

+ $\boldsymbol{\rho}$: hằng số làm mượt, thường là 0.9.

+ $\boldsymbol{\epsilon}$: một hằng số rất nhỏ, để tránh trường hợp $E[g^2]_t = 0$.

- Hình vẽ minh họa



Hình ảnh 5. Minh họa RMSprop

3.6. Adam (Adaptive Moment Estimation)

- Adam kết hợp ưu điểm của Momentum và RMSprop, sử dụng cả trung bình động của gradient (moment 1) và trung bình động của lũy thừa bậc hai gradient. Đây là một trong những thuật toán tối ưu phổ biến nhất trong Deep Learning nhờ khả năng hội tụ nhanh và ổn định.

- Cơ chế:

+ Tính trung bình động của gradient (moment 1) và lũy thừa bậc hai của gradient (moment 2).

+ Hiệu chỉnh các giá trị trung bình để giảm sai lệch ban đầu.

+ Cập nhật tham số dựa trên cả hai moment, với learning rate được chuẩn hóa.

- Công thức toán học:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla J(\theta)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)[\nabla J(\theta)]^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta = \theta - \hat{m}_t \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon}$$

trong đó:

+ $\boldsymbol{m_t}$: trung bình động của gradient (moment 1)

+ $\boldsymbol{v_t}$: trung bình động của lũy thừa bậc hai gradient (moment 2)

+ $\boldsymbol{\beta_1, \beta_2}$: hằng số làm mượt, thường $\beta_1 = 0.9; \beta_2 = 0.999$

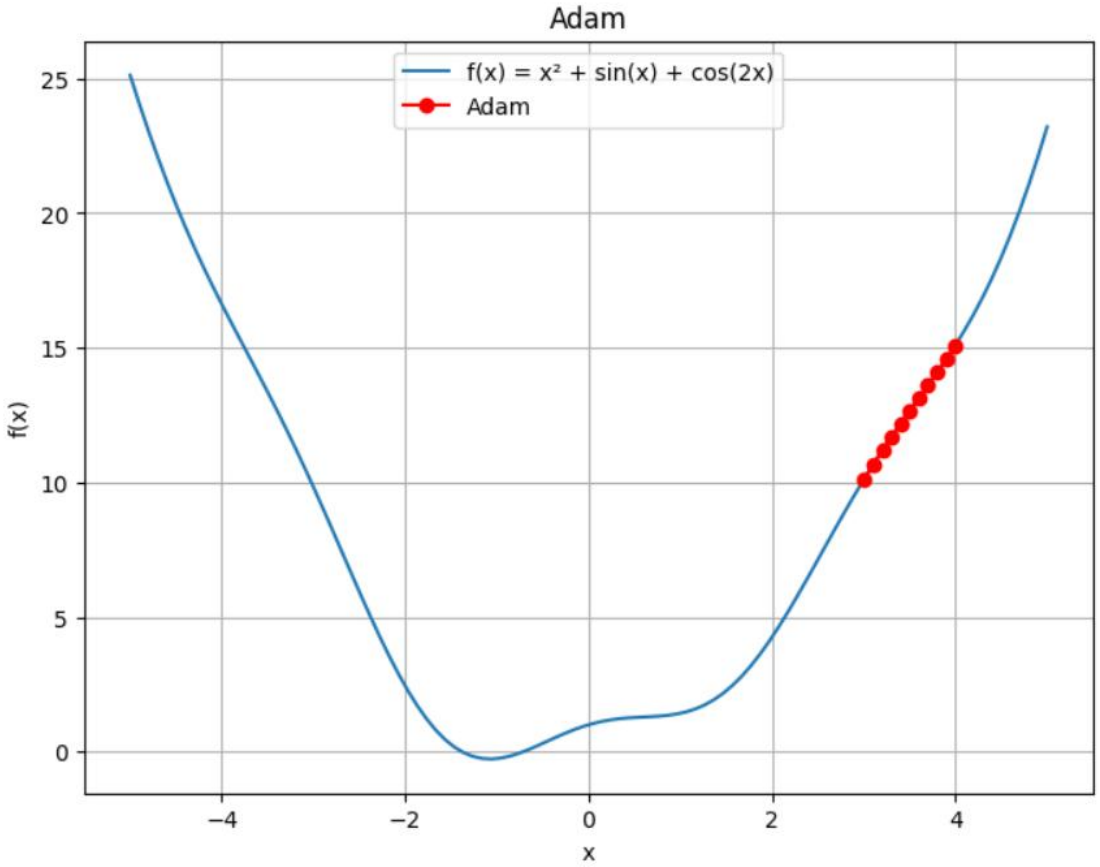
` + $\boldsymbol{\varepsilon}$: một hằng số rất nhỏ

+ $\boldsymbol{\hat{m}_t}$: moment 1 đã hiệu chỉnh bias, đảm bảo ước lượng chính xác của gradient trung bình.

+ $\boldsymbol{\hat{v}_t}$: moment 2 đã hiệu chỉnh bias, cung cấp ước lượng chính xác của biến thiên gradient.

+ $\boldsymbol{\beta_1^t, \beta_2^t}$: lũy thừa của β_1, β_2 tại bước t, biểu thị mức độ suy giảm của bias ban đầu theo thời gian, dùng trong hiệu chỉnh bias để làm cho m_t và v_t chính xác hơn khi t tăng.

- Hình vẽ minh họa:



Hình ảnh 6. Minh họa Adam

4. So sánh hiệu suất

Từ các biểu đồ minh họa ở trên, dưới đây là bảng so sánh hiệu suất của các biến thể Gradient Descent

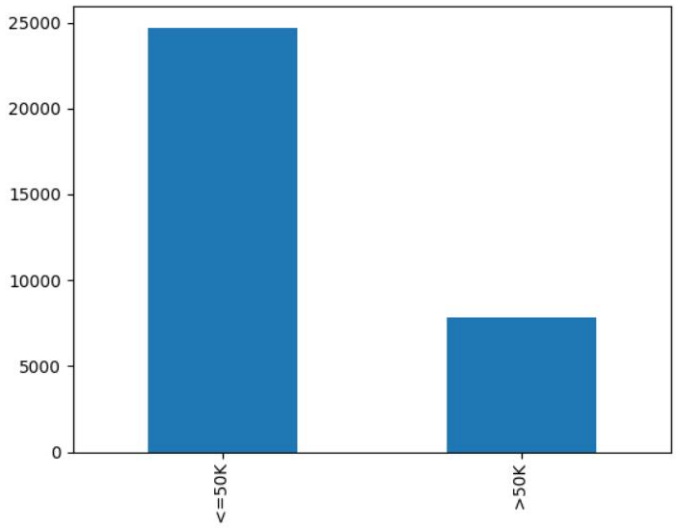
Biến thể	Đặc điểm chính	Tốc độ hội tụ	Độ chính xác	Ổn định	Ứng dụng phổ biến
Batch Gradient Descent (BGD)	Cập nhật sau khi duyệt toàn bộ dữ liệu	Chậm	Cao	Ổn định	Bài toán nhỏ, mô hình tuyến tính

	mỗi lần				
Mini-Batch Gradient Descent (MBGD)	Cập nhật sau mỗi batch nhỏ (ví dụ: 32, 64 mẫu)	Trung bình	Cao	Ít dao động	Deep Learning, dữ liệu lớn
Stochastic Gradient Descent (SGD)	Cập nhật sau mỗi mẫu dữ liệu	Nhanh		Dao động mạnh	Dữ liệu online, thời gian thực
Momentum GD	Thêm động lượng để giảm dao động	Nhanh hơn SGD	Cao	Ổn định hơn SGD	Mạng nơ-ron sâu
RMSprop	Điều chỉnh learning rate tự động	Rất nhanh	Cao	Ổn định	Deep Learning
Adam	Kết hợp Momentum & RMSprop, tối ưu nhất	Rất nhanh	Cao	Rất ổn định	Deep Learning, NLP

Bảng 1. So sánh hiệu suất của các biến thể

5. Minh họa trên tập dữ liệu adult

5.1. Giới thiệu về tập dữ liệu Adult



Hình ảnh 7. Tập dữ liệu Adult

Tập dữ liệu Adult là một bộ dữ liệu phổ biến trong lĩnh vực Machine Learning, thường được sử dụng cho bài toán phân loại nhị phân: dự đoán mức thu nhập của một cá nhân (">50K" hoặc "<=50K") dựa trên các thông tin nhân khẩu học và kinh tế. Dữ liệu được lấy từ cuộc điều tra dân số Hoa Kỳ và chứa các đặc trưng về độ tuổi, trình độ học vấn, nghề nghiệp, giới tính, tình trạng hôn nhân, v.v.

Tập dữ liệu Adult bao gồm 14 đặc trưng đầu vào và 1 nhãn đầu ra, với tổng cộng khoảng 48,842 mẫu dữ liệu.

Dữ liệu được đọc từ tệp adult.csv bằng thư viện pandas. Kích thước của tập dữ liệu sau khi tải lên được kiểm tra bằng lệnh df.shape, và danh sách các cột được lấy bằng df.columns. Bảng sau thể hiện một số đặc trưng quan trọng:

Bảng 2. Feature của dữ liệu Adult

Đặc trưng	Mô tả
age	Độ tuổi
workclass	Loại công việc (Chính phủ, Tư nhân, Tự kinh doanh, v.v.)
education	Trình độ học vấn
marital-status	Tình trạng hôn nhân
occupation	Nghề nghiệp
relationship	Mối quan hệ trong gia đình
race	Chủng tộc
sex	Giới tính
hours-per-week	Số giờ làm việc mỗi tuần
native-country	Quốc gia xuất thân
income	Nhãn đầu ra ("<=50K" hoặc ">50K")

Sau khi tải dữ liệu, phân bố của biến mục tiêu (income) được vẽ biểu đồ cột để quan sát sự chênh lệch giữa hai lớp.

5.2. Tiền xử lý dữ liệu

Do tập dữ liệu Adult chứa cả đặc trưng dạng số và phân loại, cần thực hiện một số bước tiền xử lý:

- Kiểm tra giá trị thiếu: Nếu có giá trị thiếu, các dòng có giá trị thiếu sẽ bị loại bỏ hoặc điền bằng giá trị trung bình/giá trị phổ biến.

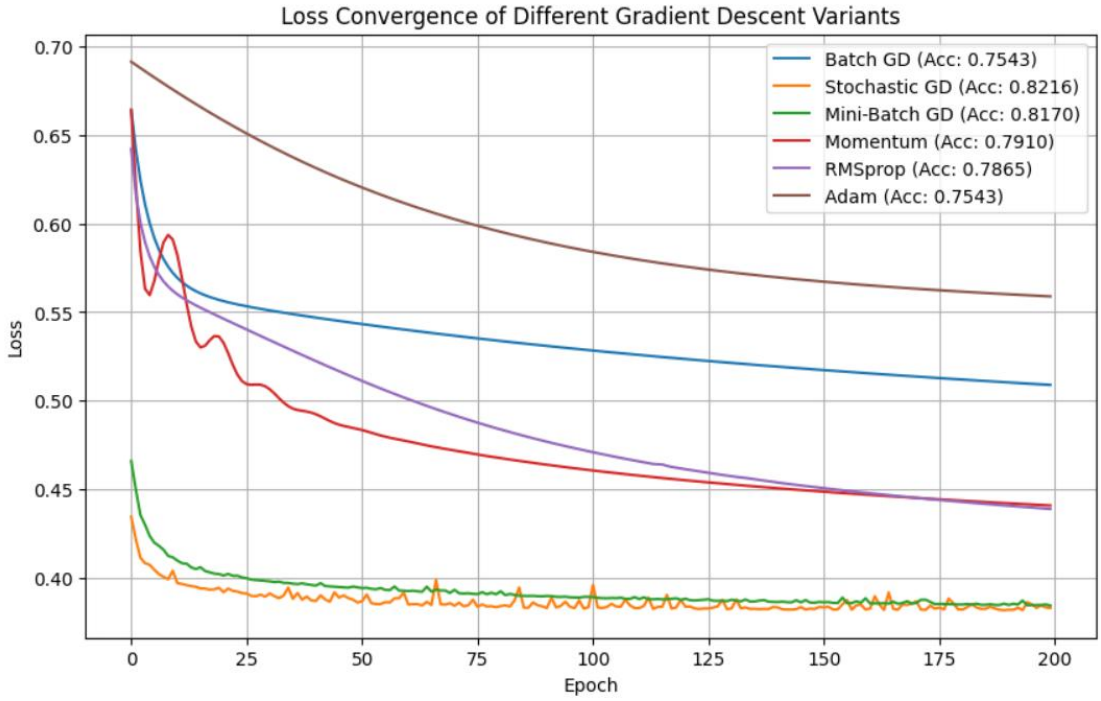
- Mã hóa dữ liệu phân loại: Các cột không phải số (ví dụ: *workclass, education, occupation*, v.v.) được chuyển thành số bằng *LabelEncoder.*

- Chuẩn hóa dữ liệu: Sử dụng *MinMaxScaler* để đưa tất cả giá trị về khoảng [0,1], giúp tăng tốc độ huấn luyện và cải thiện độ ổn định của mô hình.

- Chia dữ liệu: Dữ liệu được chia thành 80% huấn luyện và 20% kiểm tra bằng *train_test_split()* để đảm bảo tính tổng quát của mô hình.

5.3. Huấn luyện mô hình

Các mô hình được sử dụng để phân loại: Batch Gradient Descent, Stochastic Gradient Descent, Mini-Batch Gradient Descent, Momentum, RMSprop, Adam Optimizer.



Hình ảnh 8. So sánh các biến thể trên tập Adult

SGD và Mini-Batch GD có tốc độ hội tụ nhanh nhất, với độ chính xác lần lượt là 82.16% và 81.70%.

Momentum và RMSprop có sự hội tụ ổn định nhưng vẫn còn dao động nhẹ.

Adam, dù phổ biến, lại không đạt hiệu suất cao trong trường hợp này.

Batch GD có xu hướng hội tụ chậm hơn, phù hợp với các mô hình cần hội tụ ổn định nhưng không quá lớn về dữ liệu.

5.4. Kết luận

Kết quả này cho thấy rằng việc lựa chọn thuật toán tối ưu hóa phù hợp có thể ảnh hưởng lớn đến hiệu suất của mô hình Machine Learning. Trong bài toán này, SGD và Mini-Batch GD là hai phương pháp tốt nhất do tốc độ hội tụ nhanh và accuracy cao.

6. Minh họa với bài toán dự đoán giá nhà

6.1. Mô tả bài toán

Bài toán dự đoán giá nhà là một bài toán hồi quy (regression), trong đó mục tiêu là xây dựng mô hình có thể ước lượng giá nhà, giá trị của căn nhà (Y_house_price) dựa trên các đặc trưng đầu vào như:

- Tuổi của căn nhà (X2_house_age).
- Khoảng cách đến trạm MRT (X3_distance_to_MRT).
- Số cửa hàng tiện lợi gần đó (X4_convenience_stores).
- Vĩ độ (X5_latitude).
- Kinh độ (X6_longitude).

Mục tiêu của mô hình là học từ dữ liệu quá khứ để dự đoán giá nhà cho những căn nhà mới chưa biết giá.

Dữ liệu được sử dụng trong bài toán này là tập dữ liệu **Real Estate Valuation Data Set**, một tập dữ liệu phổ biến trong học máy để dự đoán giá nhà. Tập dữ liệu bao gồm 414 mẫu dữ liệu, mỗi mẫu chứa 5 đặc trưng đầu vào và 1 giá trị đầu ra (giá nhà).

Mục tiêu của bài toán là xây dựng một mô hình hồi quy tuyến tính để dự đoán giá nhà một cách chính xác, sử dụng các phương pháp tối ưu hóa Gradient Descent. Các biến thể của thuật toán Gradient Descent sẽ được áp dụng để tối ưu hóa trọng số của mô hình hồi quy tuyến tính.

6.2. Mục tiêu của bài toán

Mục tiêu chính của bài toán là:

- Xây dựng một mô hình hồi quy tuyến tính để dự đoán giá nhà dựa trên các đặc trưng đầu vào.
- Sử dụng các phương pháp tối ưu hóa Gradient Descent, bao gồm các biến thể như Batch Gradient Descent, Mini-Batch Gradient Descent, Stochastic Gradient Descent, Momentum, RMSprop, và Adam, để huấn luyện mô hình.
- Đánh giá hiệu suất của mô hình bằng chỉ số RMSE (Root Mean Squared Error), một chỉ số phổ biến trong bài toán hồi quy để đo lường độ lệch giữa giá trị dự đoán và giá trị thực tế.

- Minh họa quá trình hội tụ của hàm mất mát (loss) thông qua biểu đồ, giúp quan sát hiệu quả của phương pháp tối ưu hóa.

6.3. Các bước thực hiện

- Bước 1: Chuẩn bị dữ liệu

Đầu tiên, chúng ta cần chuẩn bị dữ liệu để huấn luyện mô hình. Các bước bao gồm đọc dữ liệu, tiền xử lý, chuẩn hóa, và chia dữ liệu thành tập huấn luyện và tập kiểm tra.

```
df = pd.read_excel('Real estate valuation data set.xlsx')  # Adjust
path to your downloaded file

# Rename columns for easier handling
df.columns = ['No', 'X1_transaction_date', 'X2_house_age',
'X3_distance_to_MRT',
              'X4_convenience_stores', 'X5_latitude', 'X6_longitude',
'Y_house_price']

# Drop unnecessary column 'No' and 'X1_transaction_date' (not useful
for prediction)
df = df.drop(['No', 'X1_transaction_date'], axis=1)

# Separate features (X) and target (y)
X = df.drop('Y_house_price', axis=1).values
y = df['Y_house_price'].values

# Standardize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Print shapes of training and test sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
```

+ **Đọc dữ liệu:** Dữ liệu được đọc từ file Real estate valuation data set.xlsx bằng thư viện pandas.

+ **Tiền xử lý dữ liệu:**

- Đổi tên các cột để dễ hiểu hơn, ví dụ: X2_house_age (tuổi nhà), X3_distance_to_MRT (khoảng cách đến trạm MRT), Y_house_price (giá nhà).

- Loại bỏ cột No (số thứ tự) và X1_transaction_date (ngày giao dịch) vì không cần thiết cho dự đoán.

- Tách dữ liệu thành đặc trưng (X) và nhãn (y).

+ **Chuẩn hóa dữ liệu**: Sử dụng StandardScaler để chuẩn hóa các đặc trưng, đưa chúng về trung bình 0 và độ lệch chuẩn 1, giúp Gradient Descent hội tụ nhanh hơn.

+ **Chia dữ liệu**: Chia dữ liệu thành tập huấn luyện (80%, 331 mẫu) và tập kiểm tra (20%, 83 mẫu) với random_state=42 để đảm bảo tính tái lập.

+ **Kết quả**:

X_train: 331 mẫu, 5 đặc trưng.

X_test: 83 mẫu, 5 đặc trưng.

y_train: 331 nhãn.

- Bước 2: Xây dựng mô hình hồi quy tuyến tính

Mô hình hồi quy tuyến tính được xây dựng để dự đoán giá nhà dựa trên các đặc trưng đầu vào. Chúng ta cũng định nghĩa hàm mất mát và các hàm hỗ trợ để dự đoán và đánh giá.

```
# Define Mean Squared Error (MSE) loss function for regression
def compute_mse_loss(X, y, weights, bias):
    m = len(y)
    predictions = X @ weights + bias
    loss = (1/(2*m)) * np.sum((predictions - y) ** 2)  # MSE
    return loss

# Prediction and RMSE calculation functions
def predict(X, weights, bias):
    return X @ weights + bias

def rmse(y_true, y_pred):
    return np.sqrt(np.mean((y_true - y_pred) ** 2))
```

+ **Mô hình hồi quy tuyến tính:** Mô hình có dạng:

predictions=X · weights+bias

Trong đó:

X là ma trận đặc trưng.

weights là vector trọng số.

bias là độ lệch.

+ **Hàm mất mát:** Hàm compute_mse_loss tính Mean Squared Error (MSE):

$$MSE = \frac{1}{2m}\sum_{l=1}^{m}(predictions_i - y_i)^2$$

Hàm này đo lường độ lệch trung bình giữa giá trị dự đoán và giá trị thực tế.

+ **Hàm dự đoán**: Hàm predict thực hiện dự đoán giá nhà dựa trên mô hình hồi quy tuyến tính.

+ **Hàm đánh giá:** Hàm rmse tính chỉ số RMSE để đánh giá độ lỗi:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y\_true_i - y\_pred_i)^2}$$

- Bước 3: Tối ưu hóa mô hình bằng Gradient Descent

Chúng ta triển khai các phương pháp tối ưu hóa Gradient Descent để huấn luyện mô hình, bao gồm 6 thuật toán: Batch Gradient Descent, Mini-Batch Gradient Descent, Stochastic Gradient Descent, Momentum, RMSprop, và Adam.

+ **Phương pháp tối ưu hóa:** Lớp GradientDescentOptimizer triển khai 6 phương pháp tối ưu hóa Gradient Descent để huấn luyện mô hình hồi quy tuyến tính:

1. Batch Gradient Descent:

Cập nhật trọng số và độ lệch dựa trên toàn bộ dữ liệu huấn luyện trong mỗi vòng lặp.

Ưu điểm: Hội tụ ổn định.

Nhược điểm: Tốn tài nguyên tính toán khi dữ liệu lớn.

Quá trình: Tính gradient trung bình trên toàn bộ dữ liệu, sau đó cập nhật trọng số và độ lệch theo công thức:

$$parameters = parameters - step\_rate \cdot grad$$

$$bias = bias - step\_rate \cdot bias\_grad\backslash$$

```
def batch(self, X_data, y_labels):
        num_samples, num_features = X_data.shape
        parameters = np.zeros(num_features)
        bias = 0

        for _ in range(self.num_iterations):
            predictions = X_data @ parameters + bias  # Linear
prediction
            grad = (1/num_samples) * X_data.T @ (predictions - y_labels)
            bias_grad = (1/num_samples) * np.sum(predictions - y_labels)

            parameters -= self.step_rate * grad
            bias -= self.step_rate * bias_grad

            loss = compute_mse_loss(X_data, y_labels, parameters, bias)
            self.loss_history.append(loss)

        return parameters, bias
```

2. Mini-Batch Gradient Descent:

Chia dữ liệu thành các batch nhỏ (kích thước batch = 32), sau đó cập nhật trọng số trên từng batch.

Ưu điểm: Cân bằng giữa tốc độ và độ ổn định, phù hợp với dữ liệu lớn.

Quá trình: Xáo trộn dữ liệu, chia thành các batch, tính gradient trên từng batch, và cập nhật trọng số:

$$weights = weights - step\_rate \cdot weight\_update$$

$$bias = bias - step\_rate \cdot bias\_update$$

```
def mini_batch(self, features, targets, batch_size=32):
        sample_count, feature_count = features.shape
        weights = np.zeros(feature_count)
        bias = 0

        for _ in range(self.num_iterations):
            shuffled_indices = np.random.permutation(sample_count)
            X_perm = features[shuffled_indices]
            y_perm = targets[shuffled_indices]

            for start_idx in range(0, sample_count, batch_size):
                X_batch = X_perm[start_idx:start_idx + batch_size]
```

```
                y_batch = y_perm[start_idx:start_idx + batch_size]

                predictions = X_batch @ weights + bias  # Linear
prediction
                weight_update = (1/len(y_batch)) * X_batch.T @
(predictions - y_batch)
                bias_update = (1/len(y_batch)) * np.sum(predictions -
y_batch)

                weights -= self.step_rate * weight_update
                bias -= self.step_rate * bias_update

            loss = compute_mse_loss(features, targets, weights, bias)
            self.loss_history.append(loss)

        return weights, bias
```

3. Stochastic Gradient Descent (SGD):

Cập nhật trọng số trên từng mẫu dữ liệu ngẫu nhiên.

Ưu điểm: Tốc độ cập nhật nhanh, phù hợp với dữ liệu lớn.

Nhược điểm: Hội tụ không ổn định, dễ dao động quanh điểm tối ưu.

Quá trình: Chọn ngẫu nhiên một mẫu dữ liệu, tính gradient, và cập nhật trọng số:

$$coeffs = coeffs - step\_rate \cdot coeff\_grad$$

$$bias = bias - step\_rate \cdot bias\_grad$$

```
def stochastic(self, input_data, output_data):
        total_samples, total_features = input_data.shape
        coeffs = np.zeros(total_features)
        bias = 0

        for _ in range(self.num_iterations):
            for _ in range(total_samples):
                rand_sample = np.random.randint(total_samples)
                X_single = input_data[rand_sample:rand_sample+1]
                y_single = output_data[rand_sample:rand_sample+1]

                prediction = X_single @ coeffs + bias  # Linear
prediction
                coeff_grad = X_single.T @ (prediction - y_single)
                bias_grad = np.sum(prediction - y_single)

                coeffs -= self.step_rate * coeff_grad
```

```
                bias -= self.step_rate * bias_grad

            loss = compute_mse_loss(input_data, output_data, coeffs,
bias)
            self.loss_history.append(loss)

        return coeffs, bias
```

4. Momentum Gradient Descent:

Sử dụng động lượng để tăng tốc hội tụ, giúp vượt qua các điểm tối ưu cục bộ.

Tham số momentum_factor=0.9 điều chỉnh mức độ ảnh hưởng của gradient trước đó.

Quá trình: Tính vận tốc (velocity) dựa trên gradient hiện tại và vận tốc trước đó, sau đó cập nhật trọng số:

$velocity\_vector = momentum\_factor \cdot velocity\_vector - step\_rate \cdot param\_grad$ $model\_params = model\_params + velocity\_vector$

```
def momentum(self, X_matrix, y_vector, momentum_factor=0.9):
        n_samples, n_features = X_matrix.shape
        model_params = np.zeros(n_features)
        bias = 0
        velocity_vector = np.zeros(n_features)
        bias_velocity = 0

        for _ in range(self.num_iterations):
            predictions = X_matrix @ model_params + bias  # Linear
prediction
            param_grad = (1/n_samples) * X_matrix.T @ (predictions -
y_vector)
            bias_grad = (1/n_samples) * np.sum(predictions - y_vector)

            velocity_vector = momentum_factor * velocity_vector -
self.step_rate * param_grad
            bias_velocity = momentum_factor * bias_velocity -
self.step_rate * bias_grad

            model_params += velocity_vector
            bias += bias_velocity

            loss = compute_mse_loss(X_matrix, y_vector, model_params,
bias)
            self.loss_history.append(loss)
```

```
        return model_params, bias
```

5. RMSprop Gradient Descent:

Điều chỉnh tốc độ học dựa trên trung bình động của gradient bình phương, giúp hội tụ nhanh hơn.

Tham số: decay_rate=0.9 (hệ số trung bình động), tiny_value=1e-8 (tránh chia cho 0).

Quá trình: Tính trung bình động của gradient bình phương, sau đó cập nhật trọng số:

$$moving\_avg = decay\_rate \cdot moving\_avg + (1 - decay\_rate)(grad\_vector^2)$$

$$coefficients = coefficients - step\_rate \cdot \frac{grad\_vector}{\sqrt{moving\_avg} + tiny\_value}$$

```
def rmsprop(self, X_input, y_target, decay_rate=0.9, tiny_value=1e-8):
        num_rows, num_cols = X_input.shape
        coefficients = np.zeros(num_cols)
        bias = 0
        moving_avg = np.zeros(num_cols)
        bias_avg = 0

        for _ in range(self.num_iterations):
            predictions = X_input @ coefficients + bias  # Linear
prediction
            grad_vector = (1/num_rows) * X_input.T @ (predictions -
y_target)
            bias_grad = (1/num_rows) * np.sum(predictions - y_target)

            moving_avg = decay_rate * moving_avg + (1 - decay_rate) *
(grad_vector ** 2)
            bias_avg = decay_rate * bias_avg + (1 - decay_rate) *
(bias_grad ** 2)

            coefficients -= self.step_rate * grad_vector /
(np.sqrt(moving_avg) + tiny_value)
            bias -= self.step_rate * bias_grad / (np.sqrt(bias_avg) +
tiny_value)

            loss = compute_mse_loss(X_input, y_target, coefficients,
bias)
            self.loss_history.append(loss)

        return coefficients, bias
```

6. Adam Gradient Descent:

Kết hợp ưu điểm của Momentum và RMSprop, điều chỉnh tốc độ học một cách thích nghi.

Tham số: b1=0.9 (động lượng), b2=0.999 (trung bình động của gradient bình phương), small_eps=1e-8 (tránh chia cho 0).

Quá trình: Tính động lượng và trung bình động của gradient, hiệu chỉnh các giá trị này, sau đó cập nhật trọng số:

$$momentum = b1.momentum + (1 - b1).grad\_vals$$

$$velocity = b2.velocity + (1 - b2).(grad\_vals2)$$

$$params = params - step\_rate \cdot \frac{momentum\_corr}{\sqrt{velocity\_corr} + small\_eps}$$

```
def adam(self, X_features, y_labels, b1=0.9, b2=0.999, small_eps=1e-8):
        sample_size, feature_size = X_features.shape
        params = np.zeros(feature_size)
        bias = 0
        momentum = np.zeros(feature_size)
        velocity = np.zeros(feature_size)
        bias_momentum = 0
        bias_velocity = 0
        iter_count = 0

        for _ in range(self.num_iterations):
            iter_count += 1
            predictions = X_features @ params + bias  # Linear
prediction
            grad_vals = (1/sample_size) * X_features.T @ (predictions -
y_labels)
            bias_grad = (1/sample_size) * np.sum(predictions - y_labels)

            momentum = b1 * momentum + (1 - b1) * grad_vals
            velocity = b2 * velocity + (1 - b2) * (grad_vals ** 2)
            bias_momentum = b1 * bias_momentum + (1 - b1) * bias_grad
            bias_velocity = b2 * bias_velocity + (1 - b2) * (bias_grad
** 2)

            momentum_corr = momentum / (1 - b1 ** iter_count)
            velocity_corr = velocity / (1 - b2 ** iter_count)
            bias_mom_corr = bias_momentum / (1 - b1 ** iter_count)
            bias_vel_corr = bias_velocity / (1 - b2 ** iter_count)
```

```
            params -= self.step_rate * momentum_corr /
(np.sqrt(velocity_corr) + small_eps)
            bias -= self.step_rate * bias_mom_corr /
(np.sqrt(bias_vel_corr) + small_eps)

            loss = compute_mse_loss(X_features, y_labels, params, bias)
            self.loss_history.append(loss)

        return params, bias
```

Mỗi phương pháp đều lưu giá trị hàm mất mát vào loss_history để vẽ biểu đồ hội tụ.

- Bước 4: Huấn luyện và đánh giá mô hình

Ta huấn luyện mô hình bằng cả 6 phương pháp tối ưu hóa Gradient Descent, dự đoán giá nhà trên tập kiểm tra, tính chỉ số RMSE cho từng phương pháp, và vẽ biểu đồ hội tụ của hàm mất mát để so sánh hiệu quả.

+ Khởi tạo bộ tối ưu hóa:

Khởi tạo 6 phương pháp tối ưu hóa với các tham số:

Batch GD, Mini-Batch GD, Momentum: step_rate = 0.1.

RMSprop, Adam: step_rate = 0.9.

Stochastic GD: step_rate = 0.0001

Số vòng lặp: 200.

```
optimizers = {
    'Batch GD': GradientDescentOptimizer(step_rate=0.1,
num_iterations=200),
    'Mini-Batch GD': GradientDescentOptimizer(step_rate=0.1,
num_iterations=200),
    'Stochastic GD': GradientDescentOptimizer(step_rate=0.0001,
num_iterations=200),
    'Momentum': GradientDescentOptimizer(step_rate=0.1,
num_iterations=200),
    'RMSprop': GradientDescentOptimizer(step_rate=0.9,
num_iterations=200),
    'Adam': GradientDescentOptimizer(step_rate=0.9, num_iterations=200)
```

+ Huấn luyện mô hình:

Với mỗi phương pháp, gọi hàm tương để tối ưu hóa trọng số và độ lệch trên tập huấn luyện (X_train, y_train).

Lưu lịch sử mất mát (loss_history) để vẽ biểu đồ hội tụ.

+ Dự đoán và đánh giá:

Dự đoán giá nhà trên tập kiểm tra (X_test) bằng hàm predict.

Tính chỉ số RMSE cho từng phương pháp để đánh giá độ lỗi.

+ Vẽ biểu đồ hội tụ: Vẽ biểu đồ của hàm mất mát (MSE) qua 200 vòng lặp cho tất cả 6 phương pháp trên cùng một biểu đồ để so sánh hiệu quả hội tụ.

+ In kết quả RMSE: In giá trị RMSE của từng phương pháp để so sánh hiệu suất.

4. Kết quả và đánh giá

- Kết quả huấn luyện

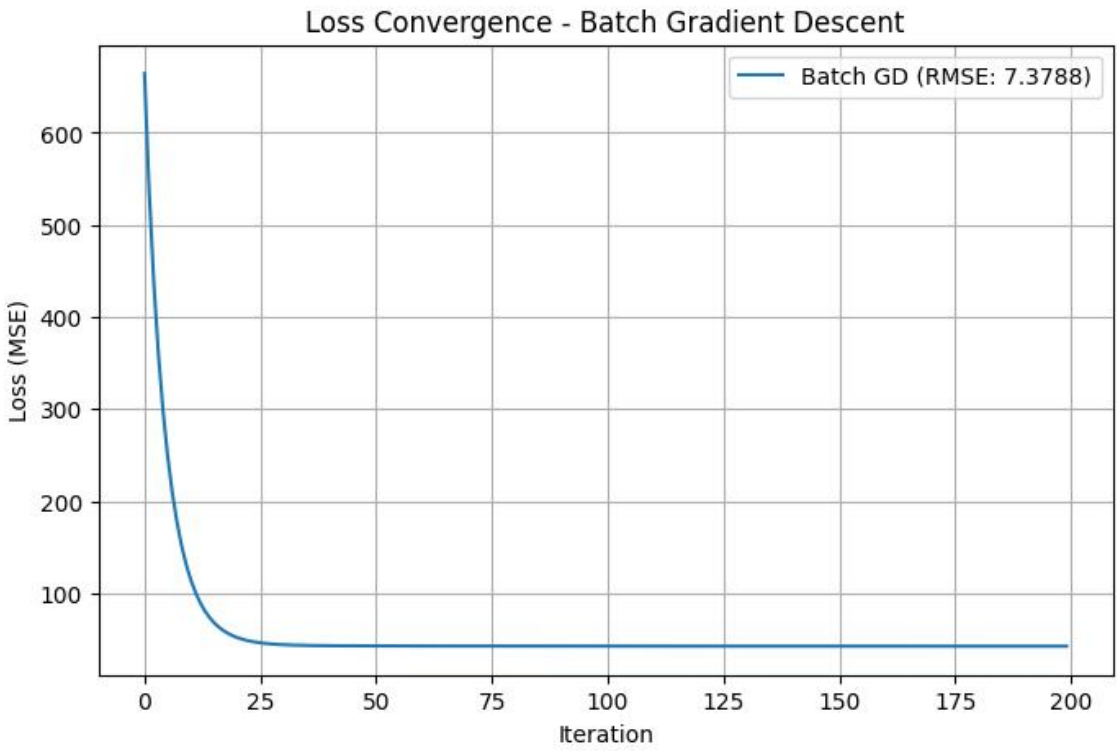
+ **RMSE**: Tất cả 6 phương pháp đều đạt chỉ số RMSE xấp xỉ 7.3 đến 7.5 .

+ Trong tập dữ liệu Real Estate Valuation, đơn vị giá là 10,000 TWD, do đó độ lệch trung bình giữa giá trị dự đoán và thực tế là khoảng 73,788 TWD.

+ Đây là một kết quả khá tốt, cho thấy mô hình có khả năng dự đoán giá nhà với độ chính xác tương đối cao. Tuy nhiên, vẫn có thể cải thiện thêm bằng cách điều chỉnh các tham số.

- Nhận xét biểu đồ hội tụ của Batch Gradient Descent

+ Batch GD



Hình ảnh 9. Biểu đồ hội tụ của Batch GD

Biểu đồ "Loss Convergence - Batch Gradient Descent" minh họa sự giảm dần của hàm mất mát (MSE) qua 200 vòng lặp:

Trục x: Số vòng lặp (iteration).

Trục y: Giá trị hàm mất mát (MSE).

Quan sát:

+ Hàm mất mát giảm rất nhanh trong khoảng 25 vòng lặp đầu tiên, từ giá trị ban đầu khoảng 600 xuống dưới 100.

+ Sau đó, đường cong giảm dần đều đặn và ổn định, đạt giá trị gần 0 sau khoảng 100 vòng lặp.

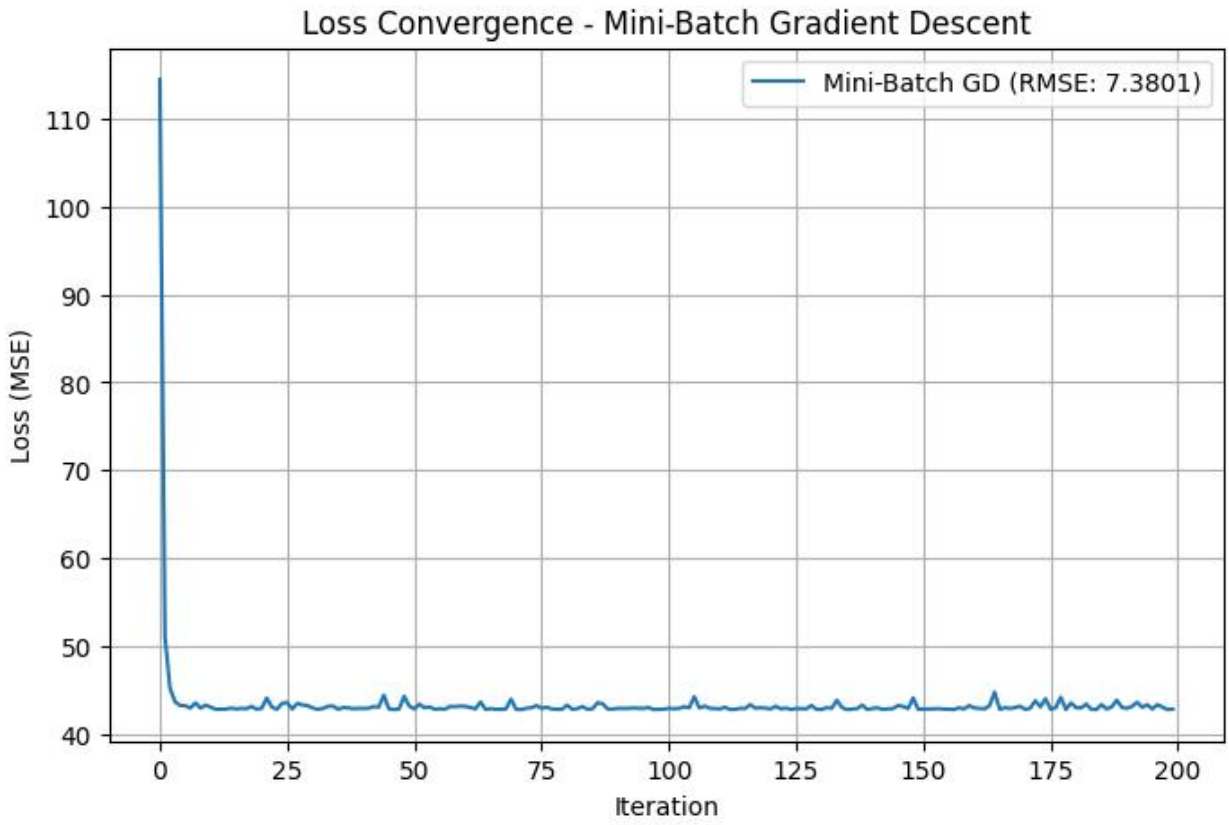
+ Từ vòng lặp 100 trở đi, hàm mất mát gần như không thay đổi, cho thấy mô hình đã hội tụ hoàn toàn.

Nhận xét:

+ Batch Gradient Descent hội tụ rất ổn định và mượt mà, đúng với đặc điểm của phương pháp này khi sử dụng toàn bộ dữ liệu để tính gradient.

+ RMSE đạt được là **7.3788**, tương đương với các phương pháp khác, cho thấy Batch GD có hiệu suất tốt nhưng có thể tốn nhiều thời gian tính toán hơn trên tập dữ liệu lớn.

+ Mini-Batch GD



Hình ảnh 10. Biểu đồ hội tụ của Mini-Batch GD

Biểu đồ "Loss Convergence - Mini-Batch Gradient Descent" minh họa sự giảm dần của hàm mất mát (MSE) qua 200 vòng lặp:

Trục x: Số vòng lặp (iteration).

Trục y: Giá trị hàm mất mát (MSE).

Quan sát:

+ Hàm mất mát giảm rất nhanh trong khoảng 10 vòng lặp đầu tiên, từ giá trị ban đầu khoảng 110 xuống dưới 50.

+ Sau đó, đường cong tiếp tục giảm nhưng có hiện tượng dao động nhẹ quanh giá trị 40-50 trong suốt 200 vòng lặp.

+ Mặc dù có dao động, giá trị hàm mất mát không tăng trở lại mà duy trì ở mức thấp, cho thấy mô hình vẫn hội tụ.

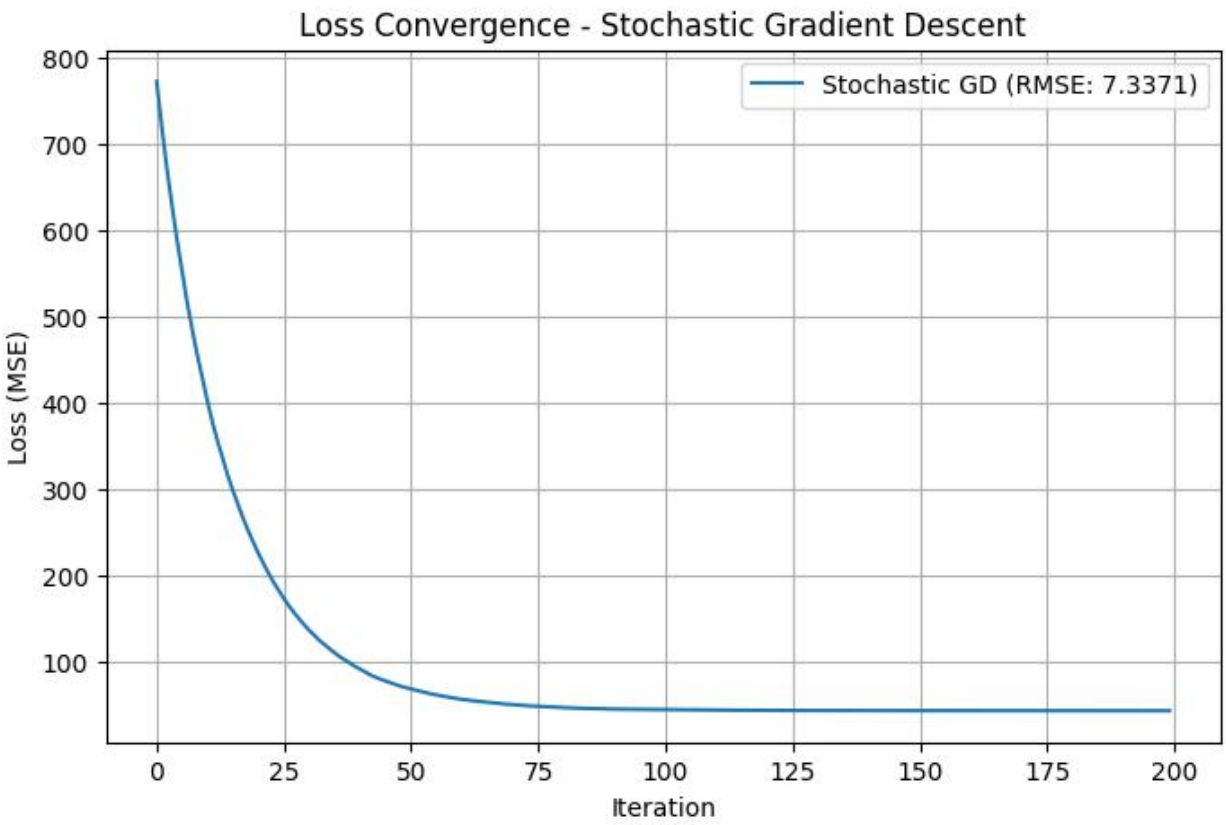
Nhận xét:

+ Mini-Batch Gradient Descent hội tụ nhanh hơn Batch Gradient Descent trong giai đoạn đầu, nhờ việc cập nhật trọng số trên từng batch nhỏ (kích thước batch = 32) thay vì toàn bộ dữ liệu.

+ Tuy nhiên, do tính ngẫu nhiên trong việc chọn batch, đường cong mất mát có dao động nhẹ, điều này là đặc trưng của phương pháp Mini-Batch GD.

+ RMSE đạt được là **7.3801**, cao hơn một chút so với Batch GD (7.3788), nhưng sự khác biệt không đáng kể, cho thấy hiệu suất của Mini-Batch GD vẫn rất tốt.

+ Stochastic GD:



Hình ảnh 11. Biểu đồ hội tụ của Stochastic GD

Biểu đồ "Loss Convergence - Stochastic Gradient Descent" minh họa sự giảm dần của hàm mất mát (MSE) qua 200 vòng lặp với step_rate=0.0001:

Trục x: Số vòng lặp (iteration).

Trục y: Giá trị hàm mất mát (MSE).

Quan sát:

+ Hàm mất mát giảm rất nhanh trong khoảng 25 vòng lặp đầu tiên, từ giá trị ban đầu khoảng 800 xuống dưới 100.

+ Sau đó, đường cong giảm dần đều đặn và ổn định, đạt giá trị gần 0 sau khoảng 100 vòng lặp.

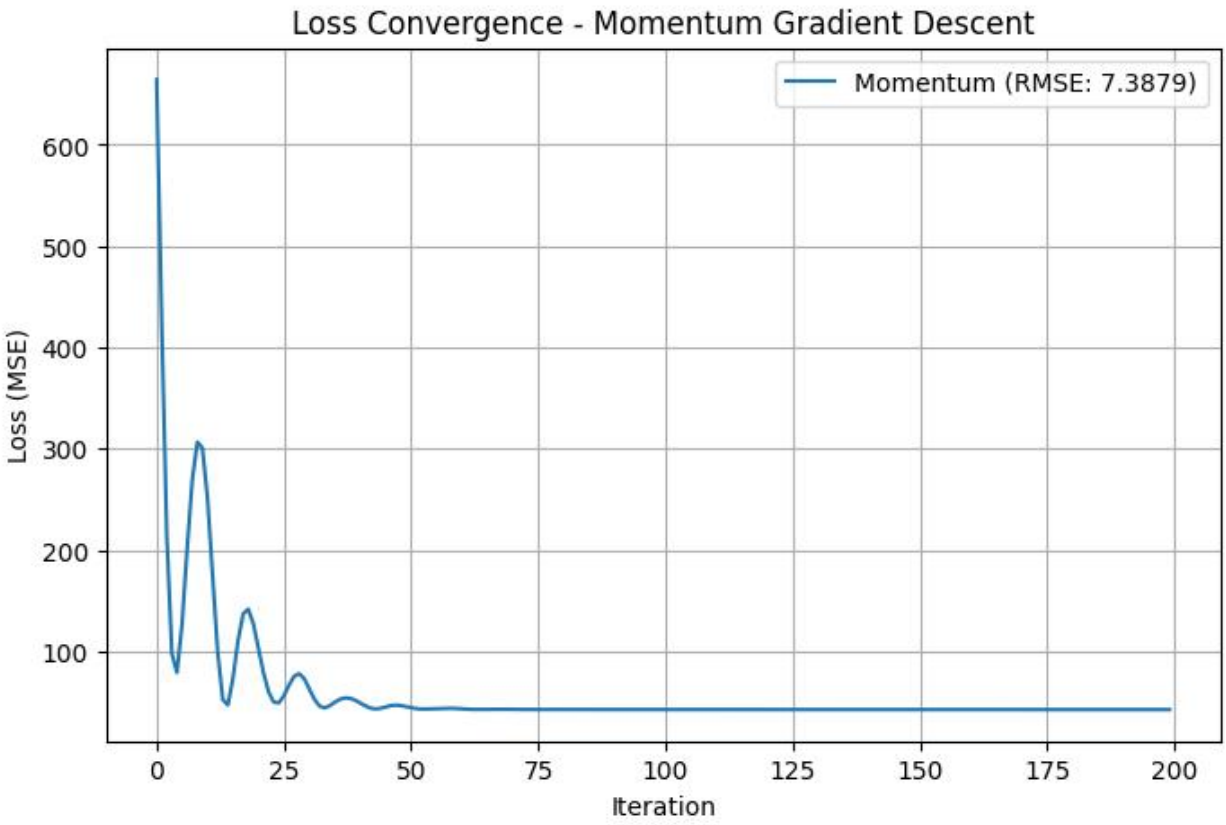
+ Từ vòng lặp 100 trở đi, hàm mất mát gần như không thay đổi, cho thấy mô hình đã hội tụ hoàn toàn.

Nhận xét:

+ Với step_rate=0.0001, Stochastic Gradient Descent (SGD) hội tụ ổn định và mượt mà

+ RMSE đạt được là **7.3371**, thấp hơn đáng kể so với Batch GD (7.3788) và Mini-Batch GD (7.3801), cho thấy hiệu suất của SGD với step_rate=0.0001 là tốt nhất trong các phương pháp đã xét cho đến thời điểm này.

+ Momentum GD:



Hình ảnh 12. Biểu đồ hội tụ của Momentum GD

Biểu đồ "Loss Convergence - Momentum Gradient Descent" minh họa sự giảm dần của hàm mất mát (MSE) qua 200 vòng lặp:

Trục x: Số vòng lặp (iteration).

Trục y: Giá trị hàm mất mát (MSE).

Quan sát:

+ Hàm mất mát giảm rất nhanh trong khoảng 10 vòng lặp đầu tiên, từ giá trị ban đầu khoảng 600 xuống dưới 100.

+ Tuy nhiên, có một dao động lớn tại vòng lặp thứ 10, khi hàm mất mát tăng đột biến lên khoảng 300 trước khi tiếp tục giảm.

+ Sau dao động này, đường cong giảm dần đều đặn và ổn định, đạt giá trị gần 0 sau khoảng 50 vòng lặp.

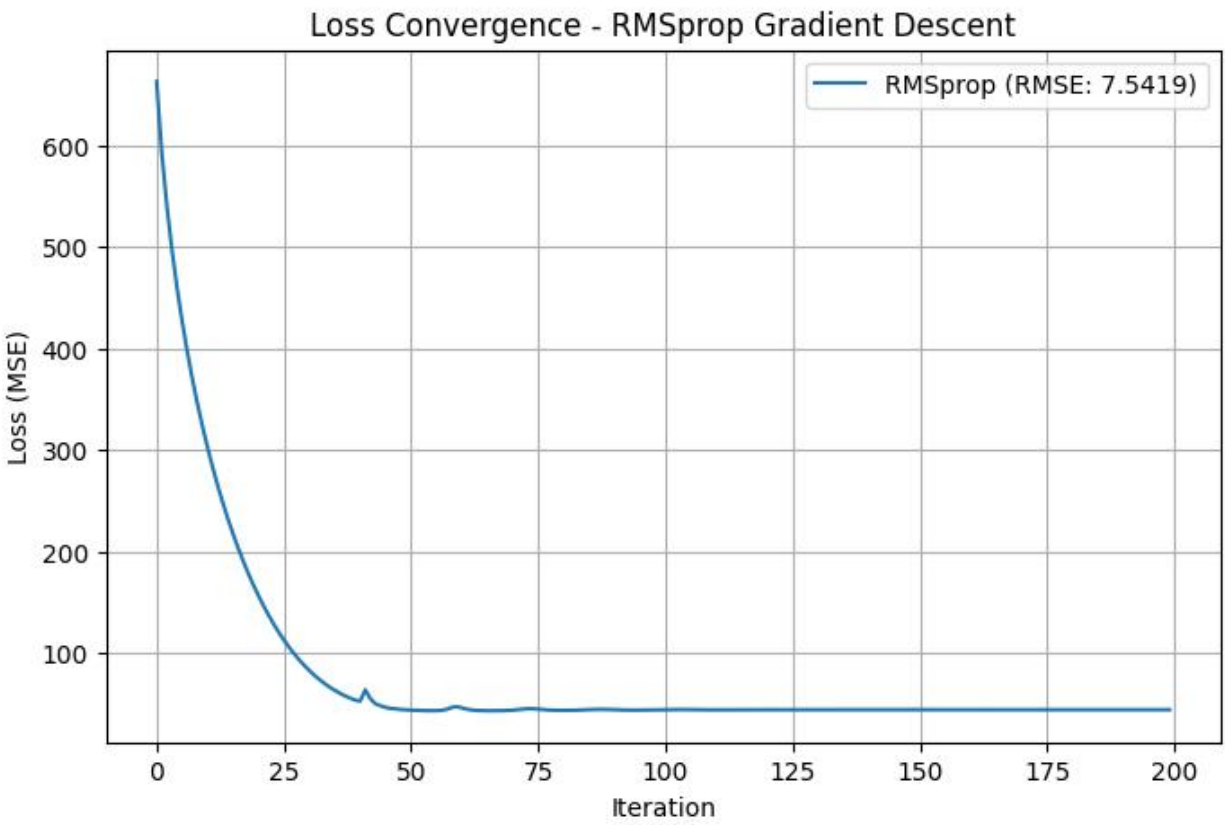
+ Từ vòng lặp 50 trở đi, hàm mất mát gần như không thay đổi, cho thấy mô hình đã hội tụ hoàn toàn.

Nhận xét:

+ Momentum Gradient Descent hội tụ nhanh hơn Batch Gradient Descent trong giai đoạn đầu, nhờ cơ chế động lượng (momentum_factor=0.9) giúp tăng tốc quá trình cập nhật trọng số.

+ RMSE đạt được là **7.3879**, cao hơn một chút so với Batch GD (7.3788) và SGD (7.3371), nhưng vẫn cho thấy hiệu suất tốt.

+ RMSprop GD:



Hình ảnh 13. Biểu đồ hội tụ của RMSprop GD

Biểu đồ "Loss Convergence - RMSprop Gradient Descent" minh họa sự giảm dần của hàm mất mát (MSE) qua 200 vòng lặp:

- **Trục x**: Số vòng lặp (iteration).
- **Trục y**: Giá trị hàm mất mát (MSE).
- **Quan sát**:

  + Hàm mất mát giảm rất nhanh trong khoảng 25 vòng lặp đầu tiên, từ giá trị ban đầu khoảng 600 xuống dưới 100.

  + Sau đó, đường cong giảm dần đều đặn và ổn định, đạt giá trị gần 0 sau khoảng 100 vòng lặp.

  + Từ vòng lặp 100 trở đi, hàm mất mát gần như không thay đổi, cho thấy mô hình đã hội tụ hoàn toàn.

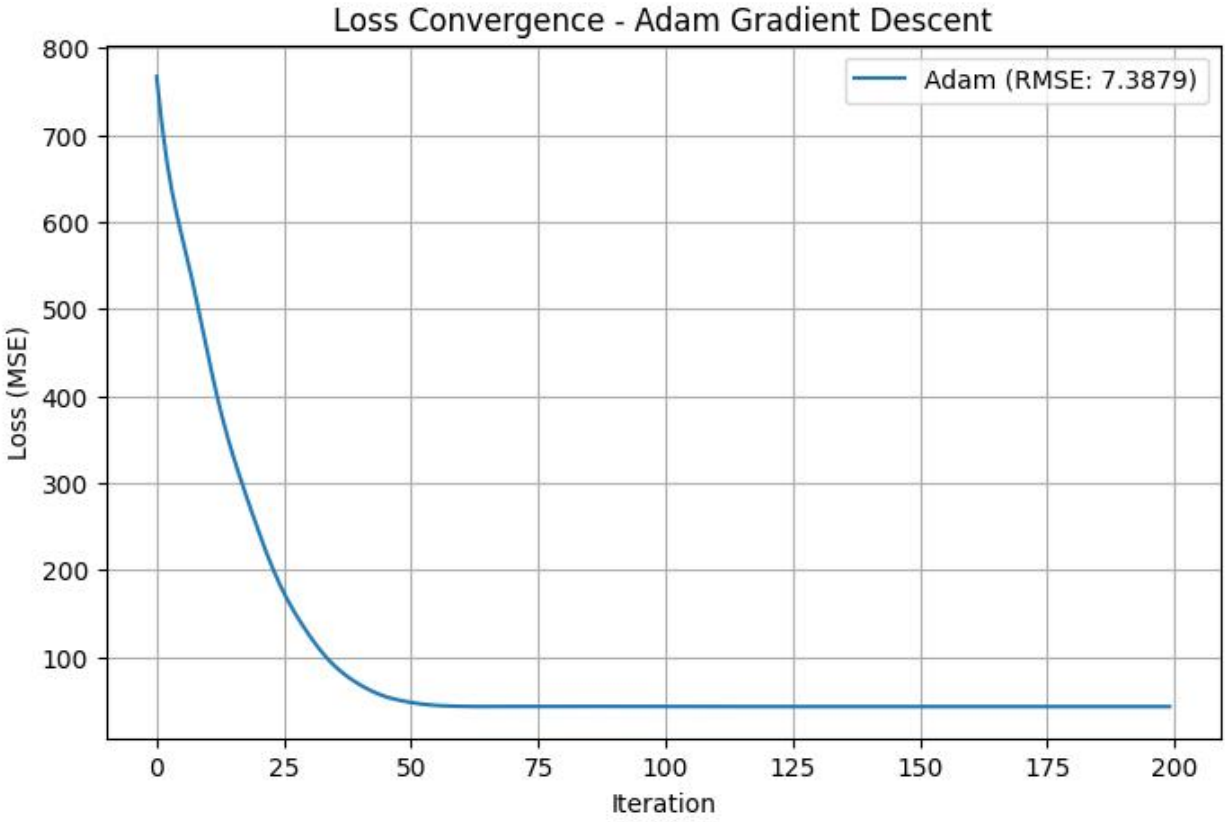
- **Nhận xét**:

  + Với step_rate=0.9, RMSprop Gradient Descent hội tụ nhanh và ổn định, nhờ cơ chế điều chỉnh tốc độ học

+ Đường cong mất mát mượt mà, không có dao động lớn, cho thấy RMSprop hoạt động hiệu quả với tốc độ học này trong bài toán này.

+ RMSE đạt được là **7.5419**, cho thấy hiệu suất của RMSprop khá tốt, dù vẫn chưa phải là tối ưu nhất trong các phương pháp đã xét.

+ Adam GD:



Hình ảnh 14. Biểu đồ hội tụ của Adam GD

Biểu đồ "Loss Convergence - Adam Gradient Descent" minh họa sự giảm dần của hàm mất mát (MSE) qua 200 vòng lặp:

- **Trục x**: Số vòng lặp (iteration).
- **Trục y**: Giá trị hàm mất mát (MSE).
- **Quan sát**:

  + Hàm mất mát giảm rất nhanh trong khoảng 25 vòng lặp đầu tiên, từ giá trị ban đầu khoảng 800 xuống dưới 100.

+ Sau đó, đường cong giảm dần đều đặn và ổn định, đạt giá trị gần 0 sau khoảng 100 vòng lặp.

+ Từ vòng lặp 100 trở đi, hàm mất mát gần như không thay đổi, cho thấy mô hình đã hội tụ hoàn toàn.

- **Nhận xét**:

+ Với step_rate=0.9, Adam Gradient Descent hội tụ rất nhanh và ổn định, nhờ cơ chế kết hợp động lượng và RMSprop để điều chỉnh tốc độ học một cách thích nghi.

+ Đường cong mất mát mượt mà, không có dao động lớn, cho thấy Adam hoạt động hiệu quả với tốc độ học này trong bài toán này.

+ RMSE đạt được là **7.3879**, cho thấy hiệu suất của Adam rất tốt, tương đương với Momentum và chỉ kém một chút so với SGD với step_rate=0.0001 (7.3371).

7. Kết luận

Trong chương này, chúng tôi đã tiến hành so sánh hiệu suất của các biến thể thuật toán Gradient Descent trong bài toán dự đoán giá nhà. Các thuật toán được đánh giá dựa trên khả năng hội tụ và độ chính xác đạt được sau một số lượng epoch nhất định.

Dựa trên kết quả thực nghiệm:

Stochastic Gradient Descent (SGD) đạt RMSE thấp nhất (7.3371), cho thấy hiệu suất dự đoán tốt nhất. Tuy nhiên, do tính chất cập nhật trọng số sau từng mẫu dữ liệu, SGD có độ dao động lớn trong quá trình tối ưu.

Mini-Batch Gradient Descent (RMSE: 7.3801) có tốc độ hội tụ nhanh hơn Batch GD nhưng dao động nhẹ do tính chất ngẫu nhiên của batch nhỏ.

Batch Gradient Descent (RMSE: 7.3788) có đường hội tụ mượt mà và ổn định, nhưng tốc độ hội tụ chậm hơn so với các phương pháp khác.

Momentum GD (RMSE: 7.3879) giúp tăng tốc hội tụ nhờ cơ chế động lượng, nhưng có dao động lớn trong những vòng lặp đầu tiên.

RMSprop GD (RMSE: 7.5419) hội tụ nhanh nhưng độ chính xác chưa tối ưu bằng các phương pháp khác.

Adam GD (RMSE: 7.3879) có cơ chế kết hợp động lượng và RMSprop để điều chỉnh tốc độ học, giúp hội tụ ổn định nhưng chưa đạt hiệu suất tốt nhất.

Như vậy, trong bài toán dự đoán giá nhà này, SGD là phương pháp có hiệu suất tốt nhất, trong khi Mini-Batch GD cũng là một lựa chọn cân bằng giữa tốc độ hội tụ và độ chính xác. Tuy nhiên, việc lựa chọn thuật toán còn phụ thuộc vào yêu cầu cụ thể của bài toán, dung lượng dữ liệu và tài nguyên tính toán sẵn có.

CHƯƠNG 2 - So sánh các thuật toán machine learning trong bài toán phân loại và phân tích ảnh hưởng của overfitting.

1. Giới thiệu bài toán phân loại

1.1. Tổng quan về bài toán phân loại

Bài toán phân loại là một trong những bài toán quan trọng trong lĩnh vực học máy, nơi mô hình được huấn luyện để phân chia dữ liệu đầu vào thành các nhóm (lớp) khác nhau dựa trên các đặc trưng của chúng. Một ứng dụng phổ biến của bài toán phân loại là phát hiện các trang web lừa đảo (phishing), giúp người dùng tránh truy cập vào các trang nguy hiểm.

1.2. Giới thiệu bộ dữ liệu PHI-UNIIL Phishing URL Dataset

Dữ liệu PHI-UNIIL Phishing URL là một tập hợp phong phú gồm gần 236 ngàn mẫu URL, với mục tiêu chính là giúp nhận diện và phân tích các trang web lừa đảo (phishing). Mỗi bản ghi trong bộ dữ liệu bao gồm nhiều đặc trưng khác nhau: từ độ dài URL, thông tin miền, các ký tự đặc biệt, chỉ số tương đồng với miền chính thống, cho đến các chỉ báo liên quan đến nội dung trang (như tiêu đề, dòng mô tả, favicon, hay các đoạn mã HTML/Javascript). Nhờ vào lượng lớn thuộc tính được thu thập, bộ dữ liệu không chỉ hỗ trợ phát triển các mô hình học máy với khả năng phân loại URL chính xác, mà còn giúp các nhà nghiên cứu hiểu rõ hơn về đặc trưng kỹ thuật và hành vi của các trang phishing. Trong bối cảnh các cuộc tấn công lừa đảo trực tuyến ngày càng trở nên tinh vi, PHI-UNIIL chính là một nguồn dữ liệu hữu ích, cho phép đánh giá và nâng cao hiệu quả của những giải pháp bảo mật trên không gian mạng.

1.3. Tóm tắt quy trình

Bộ dữ liệu được xây dựng với mục tiêu nhận diện các trang web lừa đảo (phishing), chứa nhiều đặc trưng về cấu trúc URL, thông tin bảo mật, và meta tags (như tiêu đề, mô tả, favicon). Sau quá trình làm sạch và lựa chọn đặc trưng, dữ liệu được chia thành hai phần (train/test) và đưa vào huấn luyện với nhiều

mô hình, gồm Random Forest, Gradient Boosting, Logistic Regression, SVM, KNN và Decision Tree. Kết quả cho thấy tất cả mô hình đều hoạt động rất tốt, phần lớn có độ chính xác trên 97–98%, riêng Random Forest và KNN đạt trên 98.8% cùng Recall xấp xỉ 99.2–99.3%. Các chỉ số Precision, Recall, F1-Score, và ROC AUC đều cao, chứng minh khả năng phát hiện phishing hiệu quả của bộ dữ liệu. Đáng chú ý, hiện tượng overfitting hầu như không xảy ra khi so sánh giữa tập huấn luyện và tập kiểm thử, đồng thời các phân tích về độ quan trọng của đặc trưng cho thấy URL, bảo mật (HTTPS), và yếu tố meta (copyright, description) là những tín hiệu nổi bật giúp mô hình nhận diện trang lừa đảo.

2. Chuẩn bị và tiền xử lý dữ liệu

2.1. Lựa chọn tập dữ liệu

Bộ dữ liệu được chọn từ UCI Machine Learning Repository, có tổng cộng 56 đặc trưng, được chia thành 2 nhóm:

Dữ liệu số (Numerical Features): các đặc trưng liên quan đến độ dài URL, số lượng ký tự đặc biệt, chỉ số tương đồng, tỷ lệ chữ và số trong URL.

Dữ liệu phân loại (Categorical Features): Các cờ đánh dấu như trang có favicon, trang sử dụng HTTPS, có chứa thẻ tiêu đề hay không.

2.2. Nhóm kiểu dữ liệu

- Categorical Identifiers (5 đặc trưng): FILENAME, URL, Domain, TLD, Title.
- Categorical Binary (17 đặc trưng): IsDomainIP, HasObfuscation, IsHTTPS, HasTitle, HasFavicon, Robots, IsResponsive, HasDescription, HasExternalFormSubmit, HasSocialNet, HasSubmitButton, HasHiddenFields, HasPasswordField, Bank, Pay, Crypto, HasCopyrightInfo.
- Length of Categorical Ordinal (5 đặc trưng): URLSimilarityIndex, DomainTitleMatchScore, URLTitleMatchScore, TLDLegitimateProb, URLCharProb.

- Length of Numerical Discrete(17 đặc trưng): URLLength, DomainLength, TLDLength, NoOfSubDomain, NoOfObfuscatedChar, NoOfLettersInURL, NoOfDegitsInURL, NoOfEqualsInURL, NoOfQMarkInURL, NoOfAmpersandInURL, NoOfOtherSpecialCharsInURL, NoOfURLRedirect, NoOfSelfRedirect, NoOfPopup, NoOfiFrame, NoOfSelfRef, NoOfEmptyRef.
- Length of Numerical Continuous (10 đặc trưng): CharContinuationRate, ObfuscationRatio, LetterRatioInURL, DegitRatioInURL, SpacialCharRatioInURL, LineOfCode, LargestLineLength, NoOfImage, NoOfCSS, NoOfJS.

Trong quá trình phân tích và chuẩn bị dữ liệu PHI-UNIIL, ta có thể nhóm các đặc trưng (features) dựa trên kiểu dữ liệu và mục đích sử dụng. Trước hết, một số biến mang tính chất nhận diện (Categorical Identifiers) được tách riêng để dùng cho mục tiêu tra cứu hoặc tham chiếu, thay vì đưa trực tiếp vào mô hình. Kế đến, các biến phân loại dạng nhị phân (Categorical Binary) biểu diễn thông tin "có" hoặc "không", "đúng" hoặc "sai," phù hợp cho việc phân loại dựa trên các cờ (flags). Nhóm tiếp theo là các biến phân loại mang tính thứ bậc (Categorical Ordinal), nơi giá trị của chúng phản ánh sự sắp xếp hoặc trật tự nào đó (thay vì chỉ có hai trạng thái đơn thuần). Bên cạnh đó, các biến số rời rạc (Numerical Discrete) thường liên quan đến số lượng hoặc đếm các thành phần cụ thể (chẳng hạn số lượng ký tự đặc biệt, số lượng dòng mã...). Cuối cùng, nhóm biến số liên tục (Numerical Continuous) phản ánh dữ liệu ở dạng giá trị số có thể thay đổi liên tục (vd. tỷ lệ, xác suất...). Sự phân chia này không chỉ giúp việc quản lý dữ liệu hiệu quả hơn, mà còn giúp lựa chọn kỹ thuật tiền xử lý và mô hình phù hợp khi triển khai phân tích hoặc huấn luyện thuật toán học máy.

2.3. Làm sạch và xử lý dữ liệu

2.3.1. Lọc đặc trưng theo kiểu categorical

Index	Feature	Chi2 Score	P-Value
9	HasSocialNet	78811.93404227332	0.0
16	HasCopyrightInfo	66871.06858286759	0.0

7	HasDescription	62888.38168151028	0.0
10	HasSubmitButton	46228.2673970278	0.0
11	HasHiddenFields	37821.06821214098	0.0
4	HasFavicon	36682.457685038855	0.0
5	Robots	26659.69711137214	0.0
6	IsResponsive	26647.303232701226	0.0
14	Pay	23283.517865750575	0.0

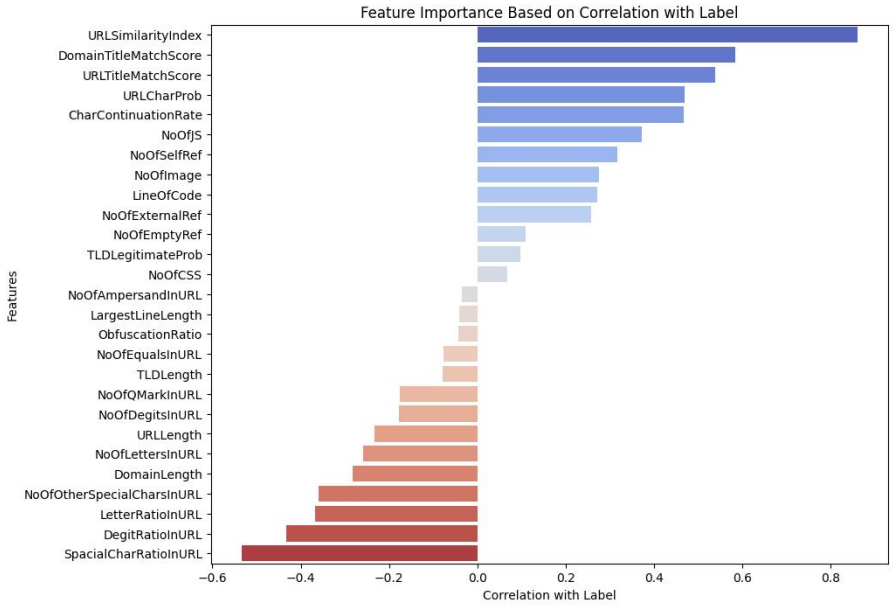
Bảng 3. Lọc đặc trưng theo Categorical

Bảng trên thể hiện kết quả kiểm định Chi-square (Chi2) giữa từng đặc trưng (feature) nhị phân và nhãn (label), trong đó **Chi2 Score** càng cao và **p-value** càng nhỏ (đặc biệt là bằng 0 hoặc rất gần 0) thì đặc trưng đó càng có mối liên hệ thống kê đáng kể với nhãn. Ngược lại, những đặc trưng có **Chi2 Score** nhỏ và p-value lớn (thường > 0.05) có khả năng không đóng góp nhiều cho việc phân loại hoặc dự đoán. Dựa trên bảng, ta có thể:

1. Chọn các đặc trưng quan trọng: Ưu tiên giữ lại các đặc trưng có Chi2 Score lớn và p-value rất nhỏ (như HasSocialNet, HasCopyrightInfo, HasDescription,...); chúng thể hiện tương quan mạnh với nhãn.

2. Loại bỏ hoặc cân nhắc loại bỏ các đặc trưng ít quan trọng: Nếu Chi2 Score thấp và p-value không đủ nhỏ (ví dụ NoOfSubDomain có p-value = 0.61), khả năng đóng góp của đặc trưng đó vào mô hình là không đáng kể.

2.3.2. Lọc đặc trưng theo numerical



Hình ảnh 15. Lọc đặc trưng theo Numerical

Biểu đồ trên cho thấy độ tương quan (Correlation) giữa từng đặc trưng (Feature) với nhãn (Label) trong tập dữ liệu. Mỗi thanh biểu diễn giá trị tương quan, có thể dương hoặc âm:

- Tương quan dương (thanh màu xanh, giá trị > 0): Khi đặc trưng tăng thì khả năng (hoặc xác suất) trang rơi vào một nhãn nhất định cũng tăng.

- Tương quan âm (thanh màu đỏ, giá trị < 0): Khi đặc trưng tăng thì xác suất (hoặc khả năng) trang thuộc nhãn tương ứng giảm đi.

- Độ lớn (tuyệt đối) của giá trị tương quan phản ánh mức độ ảnh hưởng của đặc trưng đó đến nhãn: giá trị càng lớn thì mối quan hệ càng mạnh.

Căn cứ vào biểu đồ, ta có thể:

1. Xác định các đặc trưng quan trọng

- Giữ lại những đặc trưng có |tương quan| lớn (dù dương hay âm), ví dụ như `URLSimilarityIndex` (tương quan dương mạnh) hoặc `SpacialCharRatioInURL` (tương quan âm mạnh). Chúng phản ánh quan hệ chặt chẽ nhất với nhãn, do đó thường hữu ích cho mô hình.

2. Loại bỏ hoặc giảm trọng số các đặc trưng kém quan trọng

- Các đặc trưng có giá trị tương quan gần 0 (khu vực giữa trục tung và hoành) có thể ít ảnh hưởng đến quá trình phân loại/phân cụm, dễ gây "nhiễu" hoặc làm phức tạp mô hình.

- Cân nhắc loại bỏ chúng khỏi tập huấn luyện hoặc sử dụng kỹ thuật Regularization (L1/Lasso) để mô hình tự điều chỉnh trọng số xuống gần 0.

Việc chọn và loại bỏ dựa trên ngưỡng tương quan (chẳng hạn |corr| < 0.05 hoặc 0.1...) là bước đầu để thu gọn số chiều và loại trừ tính năng dư thừa, giúp giảm nguy cơ overfitting và cải thiện hiệu suất mô hình. Sau khi loại bớt, nên đánh giá lại mô hình (ví dụ qua Accuracy, F1-score, AUC...) để đảm bảo chất lượng phân loại không bị giảm (thậm chí có thể tăng).

2.4. Các đặc trưng sau khi lọc

Dưới đây là danh sách các đặc trưng còn lại sau quá trình lọc (feature selection), kèm theo mô tả ngắn gọn về ý nghĩa của chúng:

1. HasSocialNet (int64)

Cờ cho biết trang có chứa liên kết đến mạng xã hội (Facebook, Twitter, v.v.) hay không.

2. HasCopyrightInfo (int64)

Cờ cho biết trang có thông tin bản quyền (copyright) không.

3. HasDescription (int64)

Cờ cho biết trang có thẻ meta "description" hay không.

4. HasSubmitButton (int64)

Cờ cho biết trang có nút Submit (gửi biểu mẫu) hay không.

5. HasHiddenFields (int64)

Cờ cho biết trang có các trường ẩn trong biểu mẫu hay không.

6. HasFavicon (int64)

Cờ cho biết trang có favicon hay không.

7. Robots (int64)

Thông tin về thẻ "robots" hoặc file "robots.txt" của trang (có thể chỉ định cho phép hoặc không cho phép bot thu thập dữ liệu).

8. IsResponsive (int64)

Cờ cho biết trang có giao diện responsive, tương thích với nhiều kích cỡ màn hình.

9. Pay (int64)

Cờ cho biết trang có nội dung hoặc chức năng thanh toán.

10. IsHTTPS (int64)

Cờ cho biết đường dẫn sử dụng giao thức HTTPS (1) hay HTTP (0).

11. Bank (int64)

Cờ cho biết nội dung trang liên quan đến ngân hàng.

12. HasTitle (int64)

Cờ cho biết trang có thẻ <title> hay không.

13. HasExternalFormSubmit (int64)

Cờ cho biết biểu mẫu trên trang gửi dữ liệu đến một miền bên ngoài hay không.

14. HasPasswordField (int64)

Cờ cho biết trang có trường nhập mật khẩu hay không.

15. Crypto (int64)

Cờ cho biết trang có nội dung liên quan đến tiền điện tử.

16. URLSimilarityIndex (float64)

Mức độ tương đồng của URL với các URL hợp pháp hoặc "chính thống" đã biết.

17. DomainTitleMatchScore (float64)

Mức độ tương đồng giữa tên miền và tiêu đề (title) của trang.

18. URLTitleMatchScore (float64)

Mức độ tương đồng giữa toàn bộ URL và tiêu đề của trang.

19. URLCharProb (float64)

Xác suất xuất hiện của tập ký tự trong URL, so sánh với tần suất chuẩn hoặc phổ biến.

20. CharContinuationRate (float64)

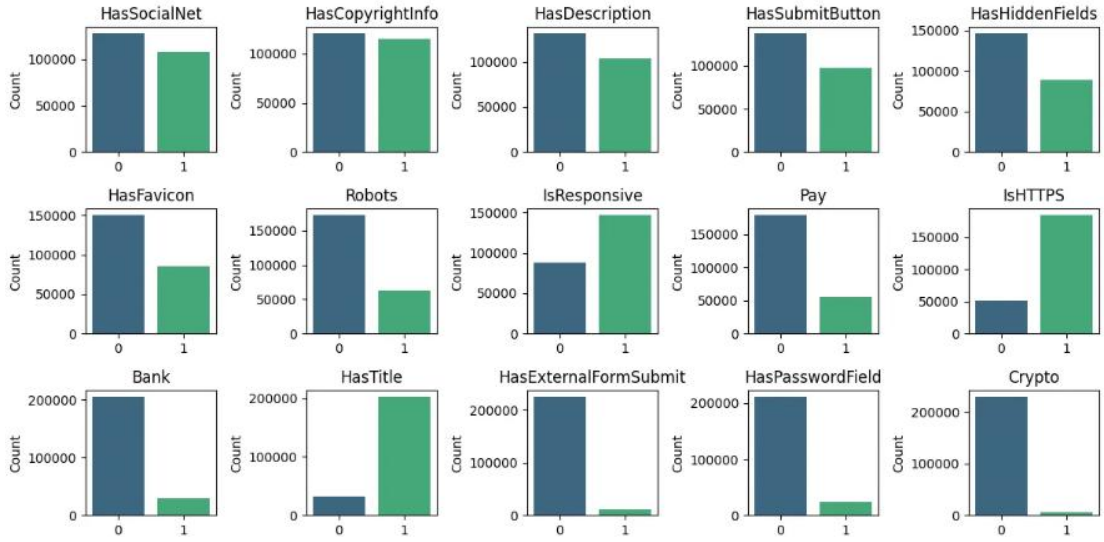
Mức độ liên tục của các ký tự chữ trong URL (dùng để phát hiện chuỗi bất thường hoặc xen kẽ ký tự không mong muốn).

21. label (int64)

Nhãn phân loại (thường: 1 = phishing, 0 = không phishing).

Các đặc trưng này được giữ lại do có tương quan cao (hoặc ý nghĩa thống kê cao) với nhãn sau quá trình phân tích, giúp mô hình học máy tập trung vào những tín hiệu quan trọng, giảm nhiễu và tối ưu hiệu suất phân loại.

2.5. Phân phối của đặc trưng categorical



Hình ảnh 16. Phân phối theo Categorical

- **HasSocialNet**: Số lượng trang không chứa liên kết mạng xã hội (0) và có chứa (1) khá tương đồng nhau.

- **HasCopyrightInfo**: Lượng trang có thông tin bản quyền (1) gần bằng lượng trang không có (0).

- **HasDescription**: Trang có thẻ meta description (1) ít hơn so với trang không có (0).

- **HasSubmitButton**: Nhìn chung, nhiều trang có nút Submit (1), nhưng tỉ lệ cũng không quá chênh lệch với trang không có (0).

- **HasHiddenFields**: Phần lớn mẫu không có trường ẩn (0), nhưng số mẫu có trường ẩn (1) cũng khá lớn.

- **HasFavicon**: Nhiều trang không có favicon (0) hơn so với có favicon (1).

- **Robots**: Số mẫu không có chỉ định robots (0) áp đảo so với mẫu có (1).

- **IsResponsive**: Số mẫu trang có thiết kế responsive (1) cao hơn so với không responsive (0).

- **Pay**: Rất ít trang có nội dung thanh toán (1) so với tổng thể (chủ yếu là 0).

- **IsHTTPS**: Phần lớn trang sử dụng HTTPS (1), thay vì HTTP (0).

- **Bank**: Rất ít trang có nội dung liên quan đến ngân hàng (1), so với đa số (0).

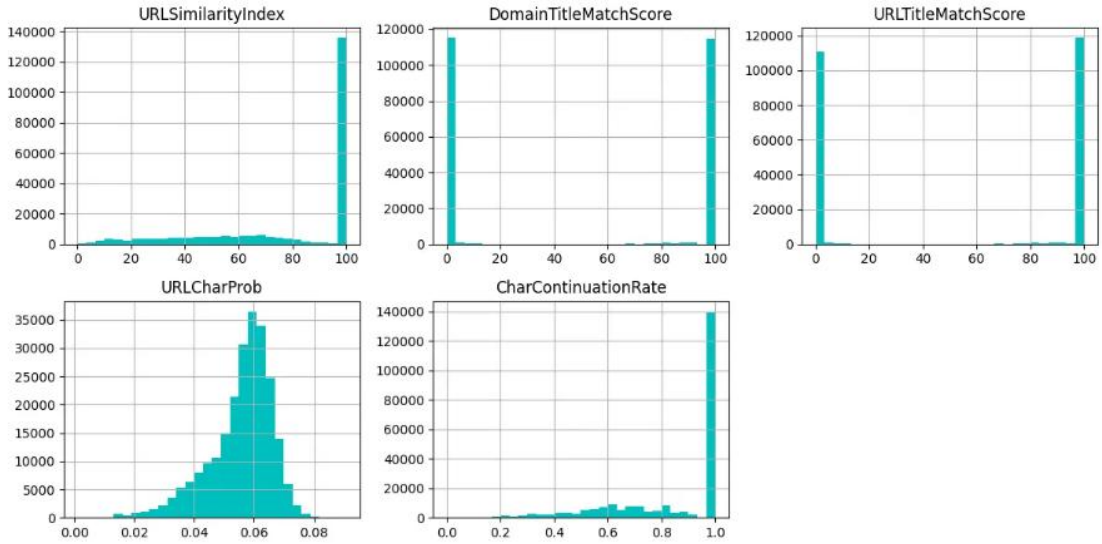
- **HasTitle**: Rất nhiều trang có thẻ tiêu đề (title) (1), chỉ một số nhỏ là không có (0).

- **HasExternalFormSubmit**: Rất ít trang gửi form ra miền ngoài (1).

- **HasPasswordField**: Phần lớn trang không có trường nhập mật khẩu (0).

- **Crypto**: Gần như toàn bộ mẫu không liên quan đến tiền điện tử (0).

2.6. Phân phối của đặc trưng numerical



Hình ảnh 17. Phân phối theo Numerical

- **URLSimilarityIndex, DomainTitleMatchScore, URLTitleMatchScore:**

- Đa phần giá trị tập trung ở hai cụm chính: gần 0 và gần 100.

- Điều này cho thấy nhiều trang hoặc là có tiêu đề/URL gần như không liên quan đến tên miền (cụm gần 0), hoặc là "trùng khớp" rất cao (cụm gần 100).

- Trường hợp tập trung ở giá trị cao (xấp xỉ 100) phản ánh tình huống trang web có URL hoặc tiêu đề rất sát với một miền/URL gốc được xem là "chính thống."

- **URLCharProb:**

- Các giá trị dao động chủ yếu trong khoảng 0.02–0.08, phân bố theo dạng "chuông" (bell-shape) tương đối.

- URLCharProb càng cao có nghĩa là các ký tự trong URL có xác suất xuất hiện thường gặp, trong khi các URLCharProb thấp cho thấy URL chứa nhiều ký tự ít phổ biến.

- **CharContinuationRate:**

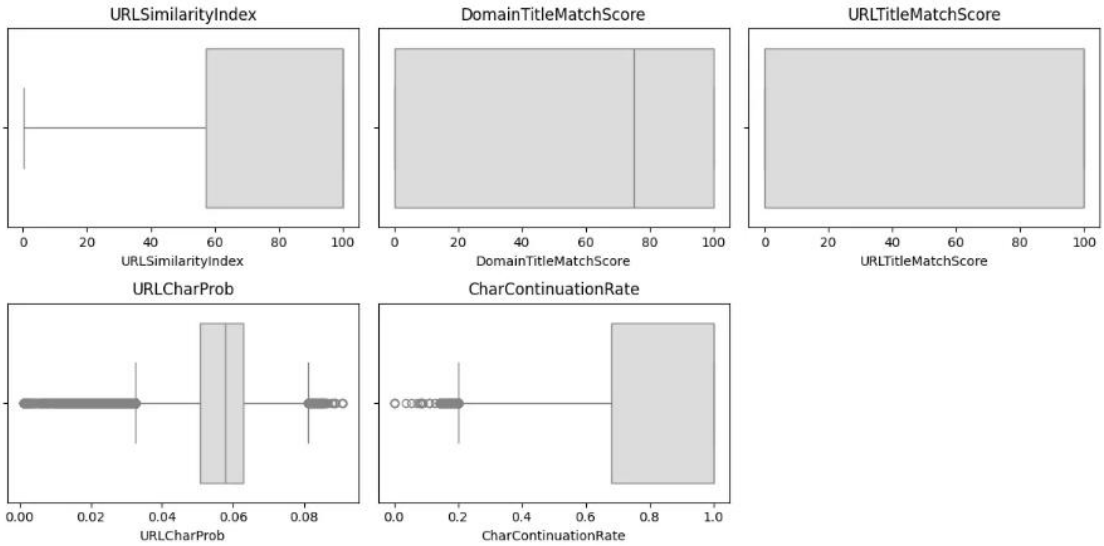
- Phân bố chênh lệch rõ: phần lớn mẫu có giá trị rất gần 1, trong khi một nhóm nhỏ trải từ 0 đến 0.8.

- Giá trị cao (gần 1) nghĩa là URL chủ yếu có các ký tự chữ (a-z, A-Z) nối tiếp nhau, ít xen kẽ ký tự đặc biệt hoặc con số.

- Ngược lại, giá trị thấp (gần 0) chỉ ra URL bị xen kẽ bởi nhiều ký tự khác nhau khiến chuỗi chữ không liên tục.

Nhìn chung, các đặc trưng này mang tính **phân cực** khá cao (thường dồn về rìa phân bố như 0 hoặc 100, 0 hoặc 1), hoặc có phân bố một đỉnh rõ rệt, cho thấy hành vi URL trong bộ dữ liệu khá riêng biệt giữa trang "tương tự" miền/tiêu đề chính thống và trang "không liên quan". Đặc điểm này có thể có lợi cho việc phân loại, vì mô hình sẽ dựa vào sự phân bố "hai cụm" hoặc "một đỉnh rõ" để tách biệt các trường hợp nghi ngờ lừa đảo (phishing).

2.7. Kiểm tra mối quan hệ giữa các đặc trưng và nhãn (label)



Hình ảnh 18. Mối quan hệ giữa các đặc trưng và nhãn

Dựa trên biểu đồ **Boxplot** của năm đặc trưng liên tục còn lại (URLSimilarityIndex, DomainTitleMatchScore, URLTitleMatchScore, URLCharProb, CharContinuationRate), ta có thể nhận thấy:

1. **URLSimilarityIndex, DomainTitleMatchScore, URLTitleMatchScore**

- Phân bố rất rộng, trải từ 0 đến gần 100, thể hiện qua hộp (box) và hai râu (whisker) khá dài.

- Có xu hướng giá trị **cực đại** xuất hiện nhiều (median và phần lớn dữ liệu nằm sát 100). Điều này gợi ý rằng một phần lớn trang có mức "khớp" (giữa URL, domain, và title) tương đối cao, song vẫn tồn tại các trường hợp giá trị thấp gần 0 (thể hiện "mismatch" rõ rệt).

2. **URLCharProb**

- Phần chính của dữ liệu nằm trong khoảng 0.04–0.06, xung quanh giá trị trung vị (median).

- Nhiều giá trị "nhô" ra ở cả hai phía dưới 0.02 và trên 0.08 (các chấm tròn ngoài boxplot), cho thấy một số URL chứa ký tự rất "đặc biệt" hoặc cực kỳ phổ thông so với "chuẩn" trung bình.

3. **CharContinuationRate**

- Thể hiện tỷ lệ liên tục của ký tự chữ trong URL, với phần lớn mẫu tập trung rất cao (gần 1).

- Tuy vậy, râu dưới (lower whisker) và các điểm outlier trải dài tới gần 0, cho thấy một số URL có nhiều ký tự xen kẽ (chữ, số, ký tự đặc biệt) đến mức làm giảm đáng kể độ liên tục.

- ***Ý nghĩa:*** Các đặc trưng này có phân bố phân cực (rất thấp hoặc rất cao), hoặc xuất hiện nhiều outliers. Trong bài toán phát hiện phishing, các giá trị "cực đoan" (rất thấp hoặc rất cao) thường mang nhiều thông tin, vì chúng thể hiện URL hoặc tên miền có vẻ rất bất thường hoặc rất giống với một trang chính thống.

- ***Hướng xử lý:***

1. **Giữ lại outliers (không loại bỏ)**

- Trong bài toán an ninh mạng/phishing, những điểm bất thường thường chứa thông tin quý giá. Loại bỏ outliers có thể vô tình xóa mất những mẫu quan trọng.

2. **Chuẩn hóa (Scaling) hoặc biến đổi (Transformation)**

- Dùng các phép biến đổi **log**, **min-max scaling**, hoặc **robust scaling** để giảm độ "nghiêng" (skewness) của phân phối, giúp mô hình học ổn định hơn.

3. **Rà soát và xác nhận dữ liệu cực đoan**

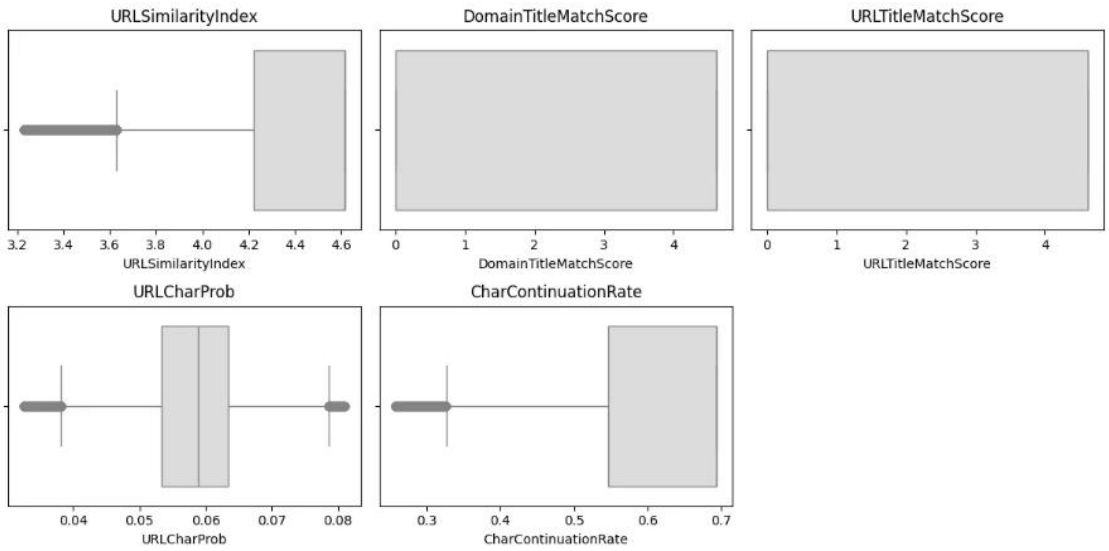
- Đảm bảo các giá trị cực đoan không đến từ lỗi thu thập dữ liệu (data collection error).

- Nếu xác nhận chính xác về mặt nghiệp vụ, hãy xem chúng là tín hiệu quan trọng để phân biệt phishing.

4. **Kết hợp với các đặc trưng khác**

- Các chỉ số như URLCharProb hay CharContinuationRate nên được xem xét song song với cờ nhị phân (HasObfuscation, IsDomainIP, v.v.), vì tính bất thường về ký tự thường đi kèm các thủ thuật làm rối (obfuscation).

- ***Sau quá trình xử lý dữ liệu lệch (Loại bỏ outliers bằng IQR:***



Hình ảnh 19. Sau quá trình xử lý dữ liệu lệch

a) Hiện tượng outliers ban đầu

- Trước khi lọc, boxplot cho thấy nhiều giá trị cực đoan ở cả hai phía (rất thấp hoặc rất cao).

- Một số mẫu có URLSimilarityIndex, DomainTitleMatchScore, URLTitleMatchScore = 0 hoặc gần 100; URLCharProb và CharContinuationRate cũng xuất hiện các giá trị gần 0 (hoặc trên 0.08, gần 1…) nằm ngoài "khoảng" phân bố chính.

b) Cách làm: Áp dụng quy tắc IQR

- **Q1 = quantile(0.25)** và **Q3 = quantile(0.75)**.

- **IQR = Q3 – Q1**.

- **Ngưỡng dưới** = Q1 – 1.5 × IQR và **ngưỡng trên** = Q3 + 1.5 × IQR.

- Mọi điểm nằm ngoài [lower_bound, upper_bound] (theo phương pháp trên) được coi là outlier và bị loại khỏi tập dữ liệu.

c) Tác động lên dữ liệu

- Sau khi loại bỏ outliers, các boxplot **"co lại"** trong phạm vi hẹp hơn.

- Với một số đặc trưng như URLSimilarityIndex, thay vì trải dài 0–100, nay chỉ còn quanh khoảng 3.2–4.6 (theo ví dụ hình). Điều này cho thấy phần

lớn mẫu "bình thường" thật ra tập trung ở một vùng giá trị khá nhỏ, còn lại là các điểm cực đoan.

- Tương tự, DomainTitleMatchScore, URLTitleMatchScore, URLCharProb, CharContinuationRate cũng được giới hạn trong dải hẹp hơn, loại bỏ các mẫu quá cao/thấp so với "phần đông."

d) Lưu ý về việc loại bỏ outliers

- **Ưu điểm**:

- Mô hình học máy (đặc biệt là mô hình nhạy với outliers) có thể ít bị ảnh hưởng bởi những giá trị cực đoan.

- Giúp phân phối dữ liệu "cân đối" hơn, giảm độ lệch (skewness).

- **Nhược điểm**:

- Trong lĩnh vực **phát hiện phishing**, chính các điểm "cực đoan" lại có thể là dấu hiệu quan trọng của hành vi bất thường. Loại bỏ những giá trị này có nguy cơ làm mất thông tin về các trang phishing đặc biệt tinh vi.

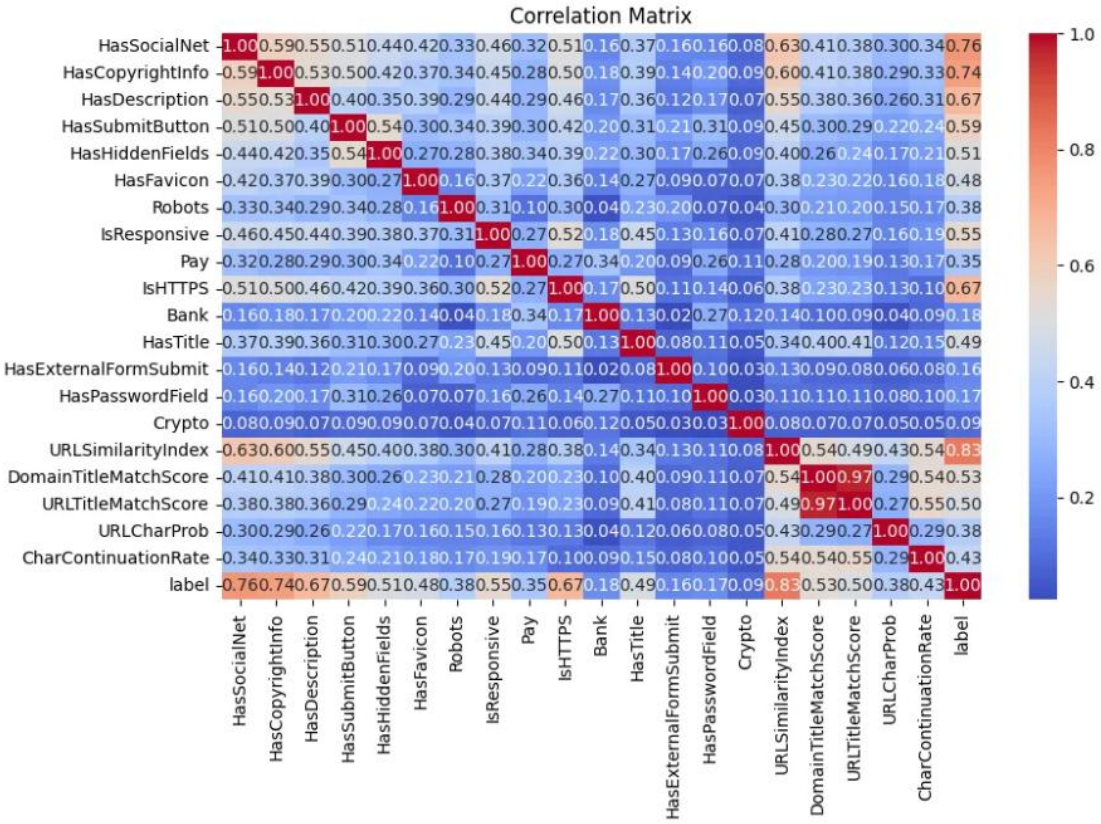
- Cần đánh giá cẩn thận, vì outliers trong an ninh mạng không phải lúc nào cũng là "nhiễu," mà có thể là đặc trưng của tấn công.

Kết luận:

- **Ý nghĩa ban đầu** của các đặc trưng: Cho thấy cách URL/trang web tương quan với "mức bình thường" (thông qua chỉ số tương đồng, tần suất ký tự, độ liên tục ký tự).

- **Sau xử lý IQR**: Dữ liệu được "làm sạch" khỏi những điểm cực đoan, giúp việc huấn luyện một số mô hình trở nên ổn định. Tuy nhiên, cần **xem xét bài toán cụ thể** (đặc biệt trong lĩnh vực phát hiện mối đe dọa) trước khi quyết định loại bỏ những mẫu "bất thường," vì chúng có thể chứa thông tin quý giá trong nhận dạng phishing.

Bảng ma trận tương quan:



Hình ảnh 20. Ma trận tương quan

Bảng ma trận tương quan cho ta biết hai biến trong dữ liệu có liên quan chặt chẽ đến mức nào:

- **Tương quan dương** (ô màu đỏ, hệ số gần +1) nghĩa là khi một đặc trưng tăng, đặc trưng kia cũng tăng.

- **Tương quan âm** (ô màu xanh, hệ số gần -1) nghĩa là khi một đặc trưng tăng, đặc trưng kia giảm.

- **Tương quan gần 0** (màu trung tính) thể hiện hai đặc trưng gần như không liên hệ.

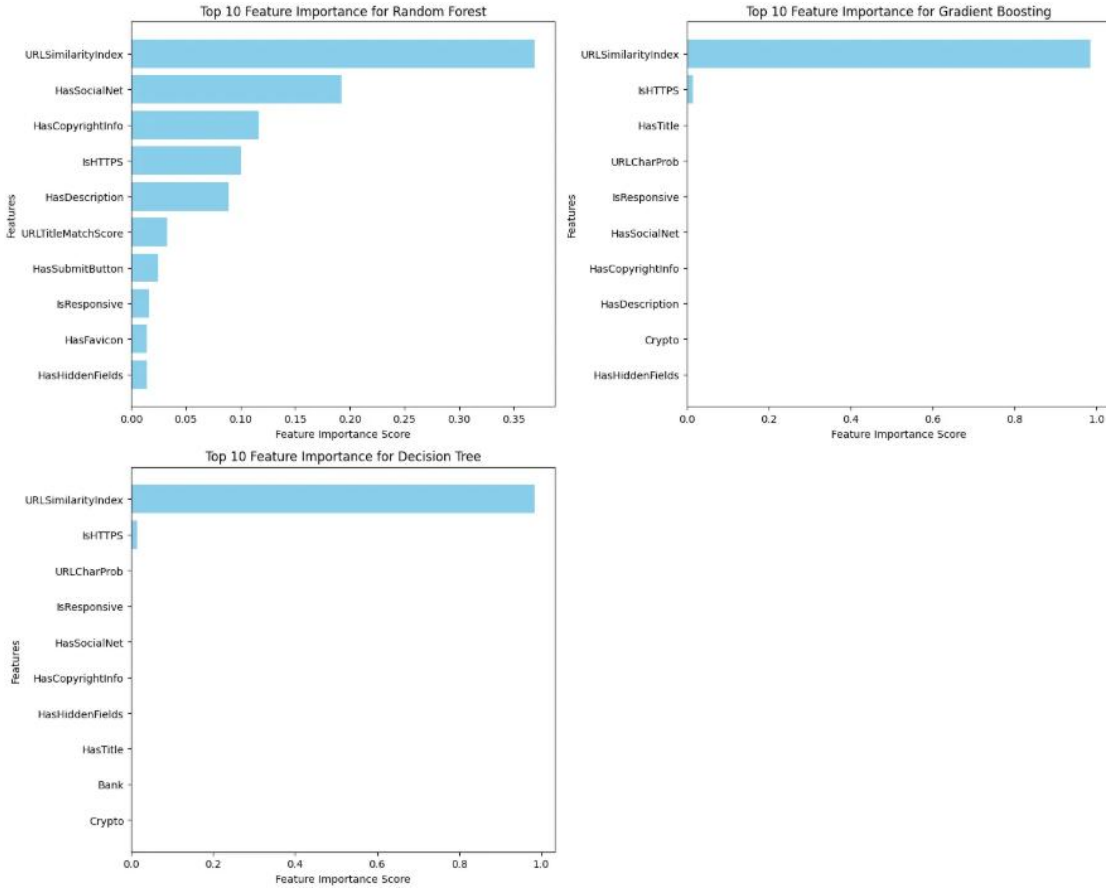
Cột hoặc hàng **"label"** trong ma trận thể hiện mức độ "liên quan" giữa từng đặc trưng với việc phân loại (phishing hay không). Các giá trị lớn (gần +1 hoặc -1) cho biết đặc trưng đó rất quan trọng, vì nó có mối quan hệ mạnh với kết quả (label).

Nhìn vào ma trận, ta có thể:

- **Xác định đặc trưng quan trọng**: Những ô tương quan cao với **label** (màu đỏ đậm) thường là các yếu tố giúp ta phân biệt giữa trang phishing và trang hợp pháp.

- **Phát hiện trùng lặp**: Nếu hai đặc trưng có tương quan quá cao, chúng có thể mang thông tin giống nhau, khiến mô hình có thể "thừa" dữ liệu. Khi đó, ta cân nhắc giữ lại một đặc trưng hoặc dùng kỹ thuật giảm nhiễu.

2.8. Kiểm tra lại sự ảnh hưởng của các đặc trưng



Hình ảnh 21. Sự ảnh hưởng của các đặc trưng

Ba biểu đồ trên cho thấy điểm chung nổi bật: URLSimilarityIndex được xếp hạng cao nhất về tầm quan trọng trong cả ba mô hình (Random Forest, Gradient Boosting, Decision Tree). Điều này gợi ý rằng mức độ tương đồng của URL với các địa chỉ "chính thống" là tín hiệu then chốt để phát hiện phishing. Bên cạnh đó, IsHTTPS cũng thường xuyên nằm trong nhóm đặc trưng quan trọng, phản ánh vai trò của việc sử dụng giao thức bảo mật trong việc phân biệt các trang đáng tin cậy với trang lừa đảo. Một số yếu tố như

HasSocialNet, HasCopyrightInfo, HasDescription, hay HasTitle tiếp tục có mặt trong top 10, nhưng trọng số của chúng thấp hơn hẳn so với hai đặc trưng cốt lõi kể trên. Điểm khác biệt giữa ba mô hình nằm ở cách phân phối "Feature Importance": Gradient Boosting tập trung gần như toàn bộ trọng số vào URLSimilarityIndex, trong khi Random Forest phân bổ đều hơn ở các đặc trưng khác, và Decision Tree (đơn lẻ) cũng đặc biệt nhấn mạnh URLSimilarityIndex và IsHTTPS. Nhìn chung, sự ưu ái đối với những đặc trưng liên quan đến URL và giao thức bảo mật cho thấy các trang phishing thường để lộ rõ sơ hở ở phần địa chỉ và cách thức mã hóa, tạo nên tiền đề vững chắc để mô hình nhận diện hành vi lừa đảo.

3. Huấn luyện mô hình với các thuật toán machine learning

- Các thuật toán để so sánh: Random Forest, Gradient Boosting, Logistic Regression, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision Tree.

3.1. Random Forest

- **Cơ chế**: Là một tập hợp (ensemble) gồm nhiều cây quyết định (Decision Trees) huấn luyện độc lập trên các mẫu dữ liệu khác nhau (chọn ngẫu nhiên các hàng và cột). Kết quả cuối cùng được lấy theo nguyên tắc đa số phiếu (hoặc trung bình trong bài toán hồi quy).

- **Ưu điểm**:

- Khả năng **giảm overfitting** so với cây quyết định đơn lẻ, do có tính "đa dạng" khi lấy mẫu.

- Hoạt động tốt với dữ liệu dạng bảng (tabular data), xử lý được cả dữ liệu dạng số và phân loại.

- **Dễ mở rộng** và tối ưu (có thể tăng số cây để cải thiện độ chính xác).

- **Nhược điểm**:

- Mô hình lớn (nhiều cây) có thể **tốn bộ nhớ** và thời gian dự đoán (inference) cao.

- **Khó giải thích** hơn so với một cây quyết định đơn lẻ.

3.2. Gradient Boosting

- **Cơ chế**: Huấn luyện mô hình theo từng "lớp" (stage) nối tiếp nhau. Mô hình ở lớp sau cố gắng khắc phục lỗi của lớp trước, dần "tăng cường" (boost) độ chính xác.

- **Ưu điểm**:

- Khả năng **học từ các lỗi** và cải thiện qua từng bước, thường cho **độ chính xác rất cao**.

- Với tham số phù hợp (learning rate, số lượng cây...), mô hình có thể **tổng quát tốt**.

- **Nhược điểm**:

- Dễ bị **overfitting** nếu không tinh chỉnh tham số cẩn thận.

- Thời gian huấn luyện lâu hơn Random Forest khi dữ liệu lớn, do tính chất tuần tự của quá trình boosting.

3.3. Logistic Regression

- **Cơ chế**: Là mô hình **tuyến tính** nhằm dự đoán xác suất thuộc về một lớp dựa trên hàm sigmoid (hoặc logit).

- **Ưu điểm**:

- **Đơn giản, dễ giải thích**, các trọng số (coefficients) cho thấy mức độ ảnh hưởng của từng đặc trưng.

- **Nhanh** trong huấn luyện và suy luận, dễ cập nhật khi có dữ liệu mới.

- **Nhược điểm**:

- **Khó nắm bắt quan hệ phi tuyến**. Nếu dữ liệu phức tạp, cần kỹ thuật feature engineering (tạo thêm biến mới hoặc sử dụng polynomial features…).

- Có thể **kém linh hoạt** hơn mô hình ensemble khi dữ liệu có nhiều tương tác phức tạp.

3.4. Support Vector Machine (SVM)

- **Cơ chế**: Xác định một "siêu phẳng" (hyperplane) phân tách tối ưu hai lớp, hoặc sử dụng kernel để xử lý quan hệ phi tuyến.

- **Ưu điểm**:

- **Hiệu quả** với dữ liệu có ít nhiễu, số chiều không quá lớn.

- Thường cho **độ chính xác cao** trong nhiều bài toán nếu chọn kernel tốt và tinh chỉnh tham số (C, gamma…).

- **Nhược điểm**:

- **Khó mở rộng** và suy luận chậm nếu dữ liệu rất lớn.

- Việc chọn kernel và siêu tham số cũng **khó** và tốn thời gian.

3.5. K-Nearest Neighbors (KNN)

- **Cơ chế**: Dự đoán dựa trên **k láng giềng gần nhất** (k-nearest neighbors) trong không gian đặc trưng.

- **Ưu điểm**:

- **Đơn giản** về mặt ý tưởng, không cần giai đoạn huấn luyện phức tạp.

- Thích hợp với dữ liệu đã được chuẩn hóa, số lượng đặc trưng không quá lớn.

- **Nhược điểm**:

- **Dự đoán chậm** với dữ liệu lớn (do phải tính khoảng cách đến từng điểm), cần cấu trúc dữ liệu đặc biệt (KD-tree, Ball-tree) để tối ưu.

- **Nhạy cảm** với giá trị của k và các đặc trưng gây nhiễu (noise).

3.6. Decision Tree

- **Cơ chế**: Tạo các nút (node) dựa trên việc chọn đặc trưng phân chia (split) tốt nhất để tách dữ liệu thành các nhóm đồng nhất (pure). Mỗi đường đi từ gốc đến lá là một "quy tắc."

- **Ưu điểm**:

- **Dễ giải thích** vì có cấu trúc phân nhánh trực quan.

- Xử lý tốt các biến liên tục và phân loại, **không đòi hỏi chuẩn hóa** dữ liệu.

- **Nhược điểm**:

- **Dễ bị overfitting** nếu cây quá sâu (không giới hạn độ sâu hoặc không tỉa cành).

- **Kém ổn định**; thay đổi nhỏ ở dữ liệu có thể dẫn đến cây rất khác.

Index	Model	Accuracy	Precision	Recall	F1-Score	ROC AUC

0	Random Forest	0.9998347341580885	0.9998133283554228	0.9999253229781196	0.99986932253066625	0.9999657003077715
1	Gradient Boosting	0.9998347341580885	0.9997386987196237	1.0	0.99986933228799177	0.9999990829600338
2	Logistic Regression	0.9992681084143923	0.998843844403834	1.0	0.999421587834686	0.9999927212200325
3	Support Vector Machine	0.9994805930682784	0.9991792269810476	1.0	0.999589445004292 1	0.9999986478155791
4	K-Nearest Neighbors	0.9971668712815186	0.9961664433526872	0.9993652453140168	0.997763280521901 2	0.9986892633881387
5	Decision Tree	0.9997166871281519	0.9998132935026139	0.9997386304234187	0.999775960569060 2	0.9997087912615359

Bảng 4. Decision Tree

4. Đánh giá kết quả và so sánh kết quả

- **Độ chính xác (Accuracy)** và các chỉ số khác (Precision, Recall, F1, ROC AUC) đều **cực kỳ cao** (> 99.7% ở tất cả các mô hình).

- **Ensemble methods (Random Forest, Gradient Boosting)** thường nhỉnh hơn về độ tin cậy, chứng minh sức mạnh của tập hợp các cây. Gradient Boosting có **ROC AUC** gần như tuyệt đối (0.99999).

- **Logistic Regression** và **SVM** cũng cho **kết quả xuất sắc**, Recall thường chạm mức 1.0 (phát hiện chính xác tất cả mẫu dương).

- **KNN** có chỉ số thấp nhất trong nhóm, nhưng **vẫn rất cao** (Accuracy ~ 99.7%).

- **Decision Tree** đơn lẻ gây ấn tượng với Accuracy ~99.97% nhưng về mặt lý thuyết có thể kém ổn định hơn Random Forest/Gradient Boosting.

5. Phân tích overfitting và phương pháp khắc phục

5.1. Kiểm tra overfitting

Được xác định dựa vào **chênh lệch giữa Train Accuracy** và **Test Accuracy**.

Trong code:

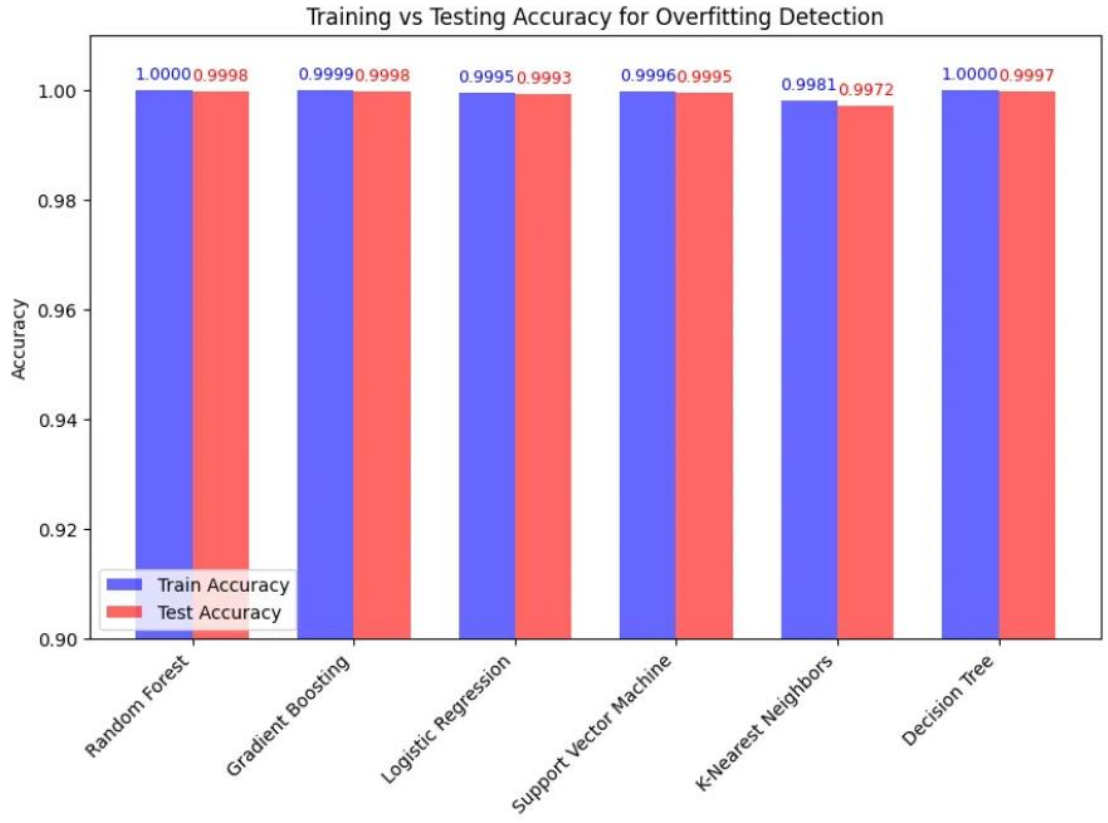
```
"Overfitting": (train_accuracy - test_accuracy) > 0.05
```

Nếu chênh lệch lớn hơn 5%, ta đánh giá mô hình có hiện tượng overfitting.

index	Model	Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Overfitting
4	K-Nearest Neighbors	0.9980640287090377	0.997166871281518 6	0.9974347296706974	0.99 616 644 335 268 72	0.99951 567986 43904	0.9993 65245 31401 68	false
2	Logistic Regression	0.9995337 1423175	0.9992681 08414392 3	0.9992740404303637	0.998843 8444038 34	0.99999 068615 12383	1.0	false
3	Support Vector Machine	0.9996399 56558693	0.9994805 93068278 4	0.9994321778307331	0.999179 2269810 476	1.0	1.0	false
1	Gradient Boosting	0.9999350 741335348	0.9998347 34158088 5	0.9998975 58159027	0.999738 6987196 237	1.0	1.0	false
5	Decision Tree	1.0	0.9997166 87128151 9	1.0	0.999813 2935026 139	1.0	0.9997 38630 42341 87	false
0	Random Forest	1.0	0.9998347 34158088 5	1.0	0.999813 3283554 228	1.0	0.9999 25322 97811 96	false

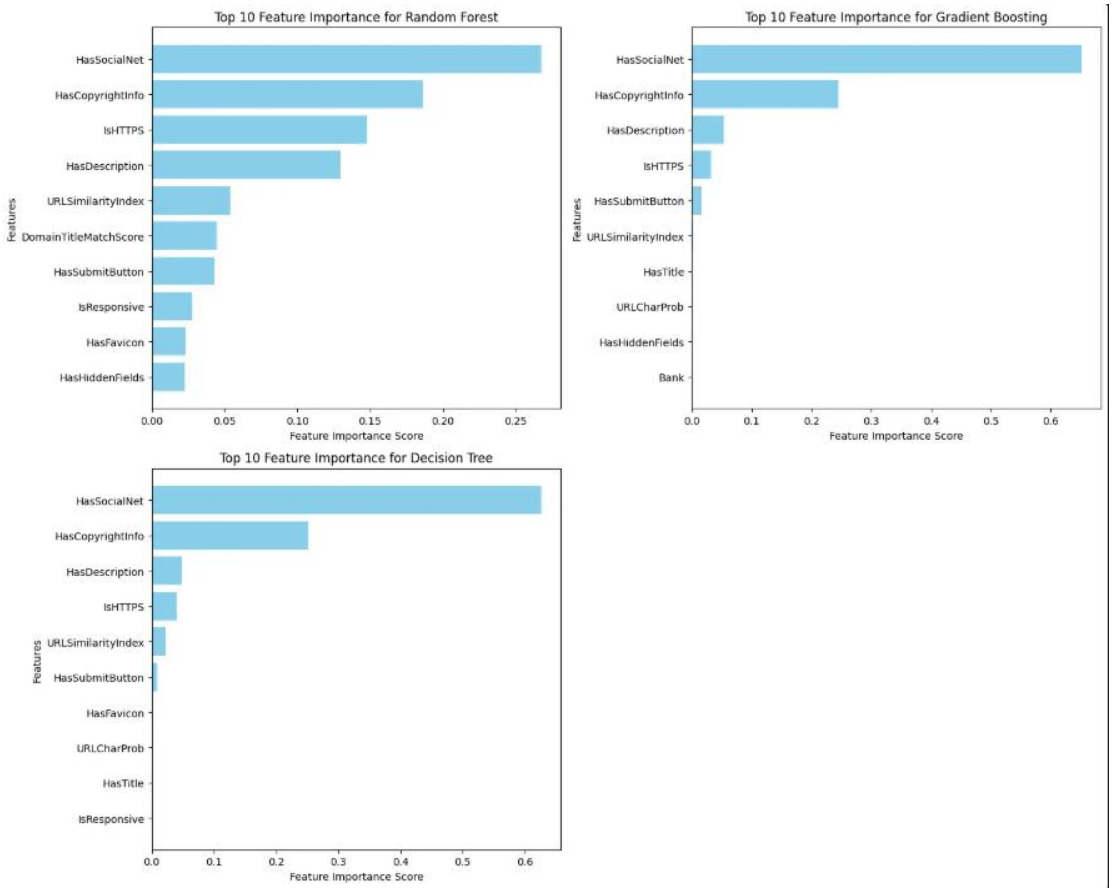
Bảng 5. Overfitting

- **Không có mô hình nào bị overfitting** (chênh lệch Accuracy < 0.05).
- Mọi mô hình đều đạt độ chính xác rất cao (≈ 0.999 hoặc hơn).
- Precision, Recall trên tập Test cũng rất cao, chứng tỏ mô hình nhận diện được đầy đủ và chính xác (gần như không bỏ sót).



Hình ảnh 22. Overfitting

5.2. Đặc trưng quan trọng nhất sau khi tái huấn luyện



Hình ảnh 23. Feature Important

Các biểu đồ trên phản ánh mức độ quan trọng (feature importance) của mười đặc trưng nổi bật đối với ba mô hình (Random Forest, Gradient Boosting, Decision Tree). Điểm chung dễ nhận thấy là HasSocialNet chiếm vị trí cao nhất hoặc gần như cao nhất trong cả ba mô hình, ngụ ý rằng sự hiện diện (hoặc không) của các liên kết mạng xã hội ảnh hưởng mạnh đến cách mô hình phân biệt trang hợp pháp và trang phishing. Tiếp theo, HasCopyrightInfo và HasDescription cũng được đánh giá cao, cho thấy thông tin bản quyền và thẻ mô tả trang đóng vai trò quan trọng trong việc nhận diện tính chân thực của website. Ngoài ra, IsHTTPS cũng nằm trong nhóm đặc trưng quan trọng ở cả ba mô hình, gợi ý rằng giao thức bảo mật là một trong những tín hiệu có giá trị phân loại. Điểm khác biệt giữa ba mô hình đến từ cách chúng phân bổ trọng số: Random Forest phân tán tầm quan trọng đều hơn, trong khi Gradient Boosting tập trung chủ yếu vào HasSocialNet, còn Decision Tree lại dành phần lớn "ưu thế" cho HasSocialNet nhưng vẫn xếp HasCopyrightInfo và HasDescription ở thứ hạng cao. Nhìn chung, các mô hình thống nhất việc xếp hạng các đặc trưng

liên quan đến mạng xã hội, bản quyền, và bảo mật (HTTPS), phản ánh rằng những yếu tố này thường là dấu hiệu mạnh để phát hiện hành vi lừa đảo trực tuyến.

5.3. Thử ngiệm chống overfitting

Sau khi loại bỏ đặc trưng quan trọng nhất rồi tái huấn luyện

Index	Model	Accuracy	Precision	Recall	F1-Score	ROC AUC
0	Random Forest	0.988879969779960 3	0.990054013782827 3	0.9923829437682026	0.991217110783747	0.9991383552414221
1	Gradient Boosting	0.971054868259514 6	0.973259259259259 3	0.9811813904861474	0.977204269086311 4	0.9943378871707973
2	Logistic Regression	0.984488620266314 1	0.987861811391223 2	0.9876036143678 59	0.987732696006124 3	0.9985954376127666
3	Support Vector Machine	0.985952403437529 5	0.989568143578239	0.9882010305429 02	0.988884114559007 6	0.9985798083747532
4	K-Nearest Neighbors	0.988714703938048 9	0.989650037230081 9	0.9925322978119633	0.991089071995824 2	0.9956095410976652
5	Decision Tree	0.978137690055718 2	0.983109118086696 5	0.9823015458143529	0.982705166037876 8	0.9954611820122992

Bảng 6. Thử nghiệm chống Overfitting

Bảng trên cho thấy tất cả các mô hình đều đạt hiệu năng cao, với độ chính xác (Accuracy) không dưới 97%, đi kèm Precision, Recall và F1-Score vượt ngưỡng 0.97. Nổi bật nhất là Random Forest, đạt Accuracy khoảng 0.9889 và Recall khoảng 0.9924, kèm theo ROC AUC lên đến 0.9991, thể hiện khả năng phân loại rất mạnh mẽ. Kế đến, K-Nearest Neighbors cũng tiến sát Random Forest về cả độ chính xác (0.9887) và độ phủ (Recall 0.9925). Các mô hình SVM và Logistic Regression dù xếp sau một chút nhưng vẫn có thành tích

vượt trội, khi đều mang lại Accuracy trên 0.984 và đặc biệt, ROC AUC cũng xấp xỉ 0.9986 – 0.9987. Gradient Boosting và Decision Tree lần lượt có Accuracy 0.9711 và 0.9781, kèm theo Precision, Recall trên 0.98, tuy vẫn rất ấn tượng, song chưa bằng hai mô hình dẫn đầu. Nhìn chung, dữ liệu được phân tách rõ ràng và các mô hình đều hoạt động tốt, cho thấy nếu ưu tiên hiệu năng cao nhất, Random Forest hay KNN là lựa chọn tối ưu, trong khi SVM và Logistic Regression cũng giữ được độ ổn định và dễ triển khai.

6. Kết luận

Trong quá trình xây dựng mô hình phân loại nhị phân, một trong những thách thức lớn nhất là hiện tượng **overfitting**, khi mô hình học quá kỹ vào tập huấn luyện và không tổng quát hóa tốt với dữ liệu mới. Để khắc phục vấn đề này, chúng tôi đã áp dụng nhiều kỹ thuật **regularization và tối ưu hóa mô hình** nhằm cải thiện khả năng tổng quát của các thuật toán.

Dưới đây là những thay đổi quan trọng được thực hiện đối với từng mô hình:

- **Random Forest**: Độ sâu của cây được giới hạn ở **10 tầng (`max_depth=10`)** nhằm tránh việc học quá kỹ vào dữ liệu huấn luyện.

-Gradient Boosting: Áp dụng **tốc độ học thấp (`learning_rate=0.01`)** và chỉ lấy **80% dữ liệu huấn luyện (`subsample=0.8`)** để tăng tính tổng quát và giảm phụ thuộc vào dữ liệu gốc.

- **Logistic Regression**: Sử dụng **L2 Regularization (`penalty='l2'`)** với hệ số phạt `C=0.1` nhằm giảm trọng số của các đặc trưng không quan trọng, giúp mô hình tránh học nhiễu.

- **Support Vector Machine (SVM)**: Giảm tham số **C xuống 0.1 (`C=0.1`)**, giúp mở rộng vùng margin và cải thiện khả năng tổng quát hóa. Đồng thời, sử dụng **kernel tuyến tính (`linear`)** thay vì phi tuyến để tránh độ phức tạp không cần thiết.

- **K-Nearest Neighbors (KNN)**: Tăng số lượng hàng xóm từ giá trị mặc định lên **5 (`n_neighbors=5`)**, giúp trung bình hóa dự đoán và giảm nhạy cảm với nhiễu.

- **Decision Tree**: Giới hạn độ sâu **(`max_depth=5`)** và yêu cầu mỗi nhánh phải có ít nhất 1**0 mẫu (`min_samples_split=10`)**, tránh việc chia nhỏ quá mức và giảm nguy cơ overfitting.

Việc áp dụng các kỹ thuật trên giúp cải thiện khả năng tổng quát của các mô hình, giảm sự chênh lệch giữa độ chính xác trên tập huấn luyện và tập kiểm tra. Điều này giúp các mô hình không chỉ hoạt động tốt trên dữ liệu huấn luyện mà còn có thể **dự đoán chính xác trên dữ liệu mới**.

Để tiếp tục tối ưu hóa mô hình, có một số khuyến nghị sau:

1. **Kiểm tra thêm bằng Cross-Validation** để đánh giá độ ổn định của mô hình.

2. **Xử lý outliers và chuẩn hóa dữ liệu** nhằm giúp mô hình hoạt động hiệu quả hơn.

3. **Tối ưu hóa hyperparameter** bằng Grid Search hoặc Bayesian Optimization để tìm ra cấu hình tốt nhất.

4. **Kiểm tra trên tập dữ liệu thực tế** để đánh giá hiệu suất thực sự của mô hình ngoài môi trường huấn luyện.

TÀI LIỆU THAM KHẢO

1. Christopher M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

2. Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980. (arXiv Link)

3. Optimizing Gradient Descent - Google AI Blog