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

BÁO CÁO CUỐI KỲ MÔN NHẬP MÔN HỌC MÁY

***Người hướng dẫn*: PGS.TS LÊ ANH CƯỜNG**

***Người thực hiện*: PHẠM TIẾN SANG – 52000794**

TRẦN THANH NHẬT THIÊN – 52100932

Lớp : 20050301

Khoá : 24

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

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

BÁO CÁO CUỐI KỲ MÔN NHẬP MÔN HỌC MÁY

***Người hướng dẫn*: PGS.TS LÊ ANH CƯỜNG**

***Người thực hiện*: PHẠM TIẾN SANG – 52000794**

TRẦN THANH NHẬT THIÊN – 52100932

Lớp : 20050301

Khoá : 24

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Để hoàn thành bài tiểu luận này, lời đầu tiên, chúng em xin trân trọng cảm ơn trường Đại Học Tôn Đức Thắng vì đã tạo điều kiện về cơ sở vật chất với hệ thống thư viện trực tuyến hiện đại, đa dạng các loại sách, tài liệu thuận lợi cho việc tìm kiếm, nghiên cứu thông tin. Đặc biệt, em xin gửi lời cảm ơn sâu sắc đến giảng viên bộ môn – GV.Lê Anh Cường, trong quá trình học tập môn Nhập môn học máy. Thầy đã giảng dạy tận tình, truyền đạt những kiến thức quý báu cho chúng em trong suốt thời gian học tập vừa qua. Từ những kiến thức này, chúng em đã dần trả lời được những câu hỏi trong bài báo cáo này. Thông qua bài báo cáo này em đã trình bày được những kiến thức quý báu mà em được tiếp thu từ thầy.

Do chưa có nhiều kinh nghiệm làm để tài cũng như những hạn chế, thiếu sót về kiến thức và khả năng lý luận, trong bài báo cáo chắc chắn sẽ không tránh khỏi những thiếu sót. Em rất mong nhận được sự nhận xét, ý kiến đóng góp, phê bình từ phía Thầy để bài báo nào này được hoàn thiện hơn và rút kinh nghiệm về sau.

Lời cuối cùng, kính chúc thầy thật nhiều sức khoẻ và thành công hơn trên con đường giảng dạy gieo mầm kiến thức quý báo cho chúng em trên con đường trao dồi kiến thức ở giảng đường đại học quý báu này.

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 12 năm 2023

Tác giả

(ký tên và ghi rõ họ tên)

Sang

Phạm Tiến Sang

Thiên

Trần Thanh Nhật Thiên

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

Đây là báo cáo cuối kỳ môn Nhập Môn Học Máy gồm 2 phần. Phần 1: sinh viên tìm kiểu về các phương pháp Optimizer, Continual Learning và Test Product. Phần 2: Sinh viên tiến hành xây dựng 1 model machine learning dựa trên các thuật toán đã được học.

MỤC LỤC

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

CÁC KÝ HIỆU

η - tỷ lệ học

CÁC CHỮ VIẾT TẮT

GD – Gradient Descent

SGD – Stochastic Gradient Descent

RNN - Recurrent Neural Network

FNN - Feed Forwark Neural Network

SVM – Support Vector Machine

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

DANH MỤC HÌNH

DANH MỤC BẢNG

CHƯƠNG 1 – CÂU 1

1.1 Các phương pháp Optimizer trong huấn luyện mô hình học máy

1.1.1 Optimizer là gì

Về cơ bản, thuật toán tối ưu là cơ sở để xây dựng mô hình neural network với mục đích "học " được các features (hay pattern) của dữ liệu đầu vào, từ đó có thể tìm 1 cặp weights và bias phù hợp để tối ưu hóa model. Optimizer giúp điều chỉnh và cập nhật các tham số của mô hình dựa trên dữ liệu huấn luyện để giảm thiểu hàm mất mát (loss function) và tối ưu hóa hiệu suất của mô hình dựa trên dữ liệu mới.

1.1.2 Thuật toán Gradient Descent (GD)

Thuật toán Gradient Descent là một trong những thuật toán cơ bản và quan trọng nhất trong Machine Learning và tối ưu hóa. Nó được sử dụng để tìm giá trị tối ưu của một hàm số bằng cách di chuyển từ điểm khởi đầu theo hướng ngược với Gradient của hàm số.

[1] Các bước cơ bản của Gradient Descent:

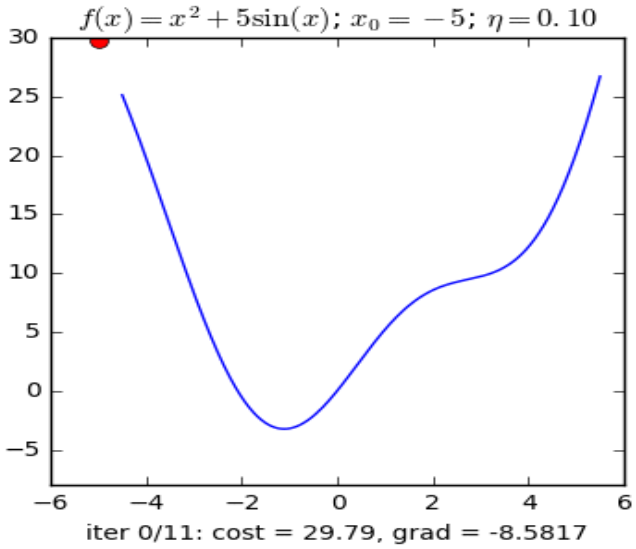
- Khởi tạo tham số: Bắt đầu từ một điểm khởi đầu ngẫu nhiên hoặc được chọn trước.
- Tính toán Gradient: Tính toán Gradient của hàm mất mát tại điểm hiện tại. Gradient là vector đạo hàm riêng theo từng tham số của hàm mất mát, chỉ ra hướng tăng nhanh nhất của hàm tại điểm đó.
- Di chuyển theo Gradient ngược: Thay đổi tham số theo hướng ngược với Gradient để giảm thiểu mất mát. Bước này sử dụng tỉ lệ học tập (learning rate) để quyết định kích thước bước di chuyển.
- Cập nhật tham số: Cập nhật tham số của mô hình bằng cách trừ Gradient nhân với tỉ lệ học tập từ giá trị hiện tại của tham số.
- Lặp lại quá trình: Tiếp tục quá trình tính toán gradient và cập nhật tham số cho đến khi đạt được điều kiện dừng (thường là số lần lặp hoặc khi hàm mất mát đạt một ngưỡng mong muốn).

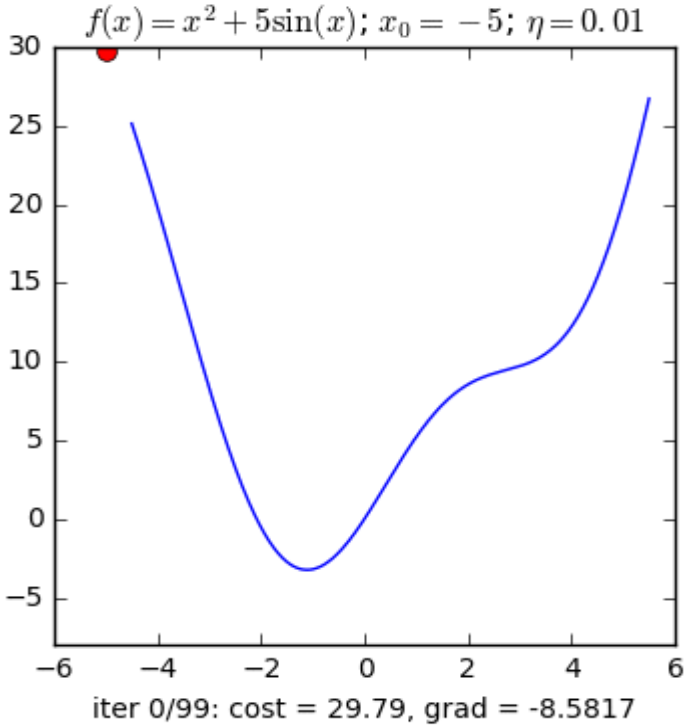
Loại Gradient Descent:

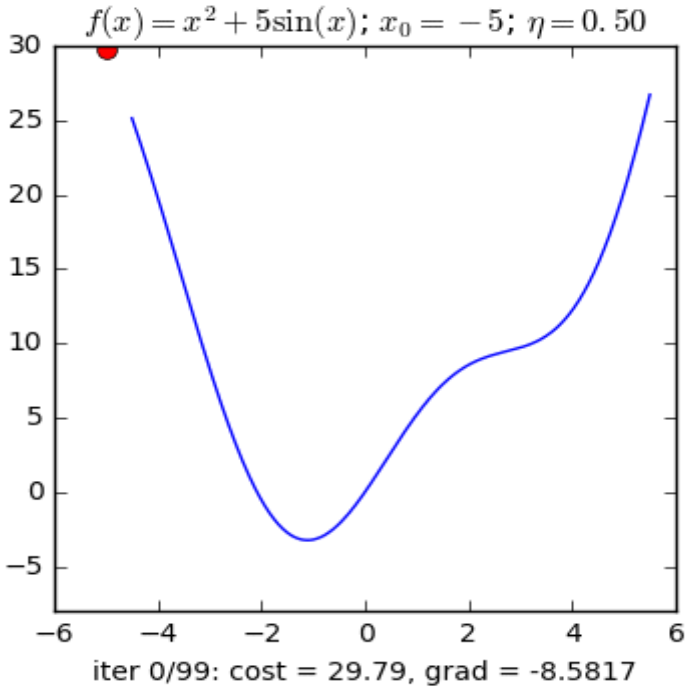
- Batch Gradient Descent: Tính toán Gradient dựa trên toàn bộ dữ liệu huấn luyện. Tốn nhiều bộ nhớ và thời gian tính toán khi dữ liệu lớn
- Stochastic Gradient Descent(SGD): Tính toán gradient trên từng mẫu dữ liệu ngẫu nhiên. Hiệu quả hơn với dữ liệu lớn nhưng có thể dao động hơn.
- Mini-batch Gradient Descent: Kết hợp giữa Batch GD và SGD bằng cách tính toán Gradient trên một lô dữ liệu nhỏ (mini-batch). Kompromiss giữa hai phương pháp trên.

Ví dụ:

- [2]Gradient cho hàm 1 biến:



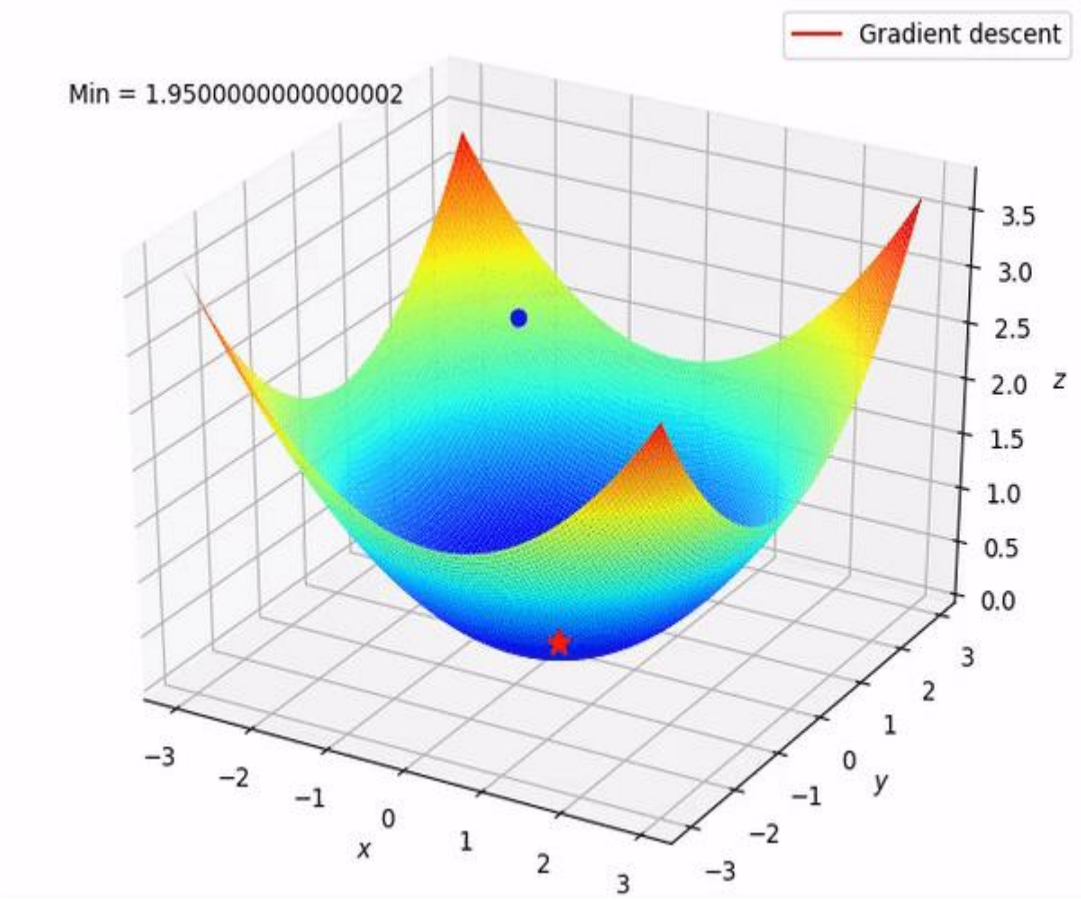




Hình 1: Gradient cho hàm 1 biến

Qua các hình trên ta thấy Gradient descent phụ thuộc vào nhiều yếu tố : như nếu chọn điểm x ban đầu khác nhau sẽ ảnh hưởng đến quá trình hội tụ; hoặc tốc độ học (learning rate) quá lớn hoặc quá nhỏ cũng ảnh hưởng: nếu tốc độ học quá nhỏ thì tốc độ hội tụ rất chậm ảnh hưởng đến quá trình training, còn tốc độ học quá lớn thì tiến nhanh tới đích sau vài vòng lặp tuy nhiên thuật toán không hội tụ, quanh quẩn quanh đích vì bước nhảy quá lớn.

[2]Gradient Descent cho hàm nhiều biến:



Hình 2: Gradient descent cho hàm nhiều biến

Ưu điểm: Thuật toán gradient descent cơ bản, dễ hiểu. Thuật toán đã giải quyết được vấn đề tối ưu model neural network bằng cách cập nhật trọng số sau mỗi vòng lặp.

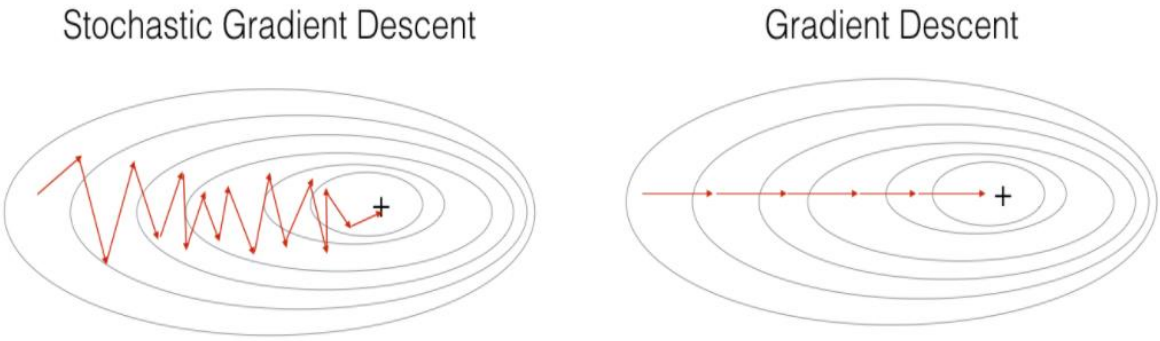
Nhược điểm:

- Vì đơn giản nên thuật toán còn nhiều hạn chế như phụ thuộc vào nghiệm khởi tạo ban đầu và learning rate.
- Ví dụ 1 hàm số có 2 global minimum thì tùy thuộc vào 2 điểm khởi tạo ban đầu sẽ cho ra 2 nghiệm cuối cùng khác nhau.
- Tốc độ học quá lớn sẽ khiến cho thuật toán không hội tụ, quanh quẩn bên đích vì bước nhảy quá lớn; hoặc tốc độ học nhỏ ảnh hưởng đến tốc độ training.

1.1.3 Thuật toán Stochastic Gradient Descent (SGD)

[1]Stochastic là 1 biến thể của Gradient Descent, nhưng khác với GD, nó chỉ sử dụng một mẫu dữ liệu ngẫu nhiên từ tập huấn luyện để tính toán gradient tại mỗi bước cập nhật . Thay vì sau mỗi epoch chúng ta sẽ cập nhật trọng số (Weight) 1 lần thì trong

mỗi epoch có N điểm dữ liệu chúng ta sẽ cập nhật trọng số N lần. Nhìn vào 1 mặt , SGD sẽ làm giảm đi tốc độ của 1 epoch. Tuy nhiên nhìn theo 1 hướng khác,SGD sẽ hội tụ rất nhanh chỉ sau vài epoch. Công thức SGD cũng tương tự như GD nhưng thực hiện trên từng điểm dữ liệu.



Hình 3: Stochastic Gradient Descent và Gradient Descent

Ưu điểm:

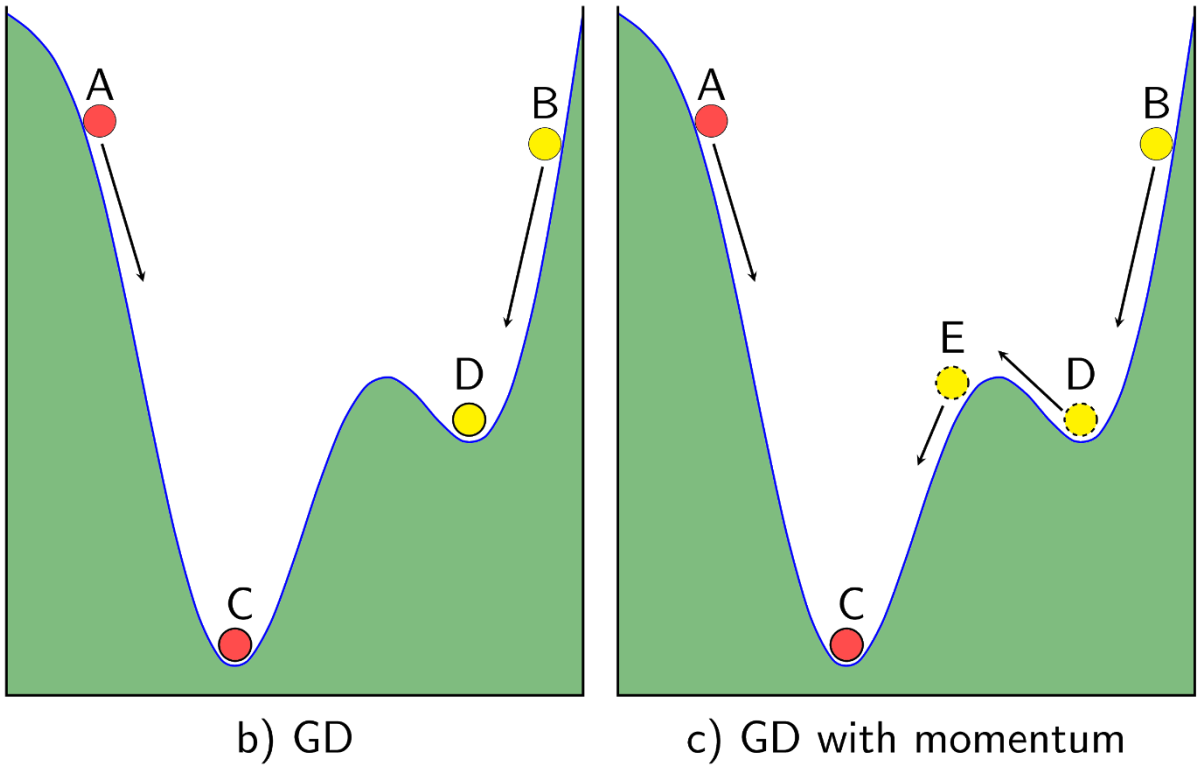
- Tính toán nhanh hơn: Vì chỉ sử dụng một mẫu dữ liệu, SGD thường nhanh hơn GD.
- Phù hợp với dữ liệu lớn: Đặc biệt hiệu quả khi làm việc với dữ liệu lớn do không cần tính toán gradient cho toàn bộ dữ liệu.
- Có thể thoát khỏi điểm tối ưu cục bộ: Do tính ngẫu nhiên của việc chọn mẫu dữ liệu, SGD có thể thoát khỏi điểm tối ưu cục bộ và tìm ra điểm tối ưu toàn cục tốt hơn.

Nhược điểm:

- Khó điều chỉnh kích thước (Learning Rate): Do tính ngẫu nhiên của việc chọn mẫu, việc điều chỉnh Leaning rate có thể khó khăn.
- Khả năng dao động: SGD có thể dao động quanh điểm tối ưu vì tính ngẫu nhiên trong việc chọn mẫu, làm cho quá trình hội tụ có thể không ổn định.

1.1.4 Thuật toán Momentum

[2]Để khắc phục các hạn chế trên của thuật toán Gradient Descent người ta dùng gradient descent with momentum. Vậy gradient with momentum là gì ?



Hình 4: Gradient Descent with momentum

Thuật toán Momentum được sử dụng trong việc tối ưu hóa và cập nhật trọng số trong quá trình huấn luyện mạng neural. Nó giúp tăng tốc độ hội tụ và tránh được các dao động không mong muốn trong quá trình tối ưu hóa. Cơ bản, Momentum sử dụng một biến momentum để tính toán việc cập nhật trọng số dựa trên gradient của các trọng số trước đó cùng với một tỷ lệ học (learning rate).

Đây là cách thuật toán Momentum hoạt động:

- Gradient Descent: Trong quá trình huấn luyện mạng neural, gradient của hàm mất mát được tính bằng cách sử dụng backpropagation để xác định hướng cần đi để giảm thiểu hàm mất mát.
- Momentum: Thay vì chỉ sử dụng gradient hiện tại, thuật toán Momentum sử dụng một biến momentum để tính toán việc cập nhật trọng số. Biến momentum lưu trữ hướng và tốc độ trước đó của gradient.
- Cập nhật trọng số: Công thức cập nhật trọng số trong thuật toán Momentum được tính dựa trên hai thành phần chính: gradient hiện tại và momentum từ các bước trước đó. Công thức cập nhật có thể được biểu diễn như sau:
    - $v_{t+1} = \gamma v_t + \eta \nabla J(\theta_t)$

- v_{t+1} là giá trị Momentum ở bước tiếp theo
- γ là hệ số momentum
- v_t là giá trị momentum ở bước hiện tại
- η là tỷ lệ học
- $\eta \nabla J(\theta_t)$ là gradient descent của hàm mất mát

- Cập nhật trọng số: Sau khi tính được v_{t+1} trọng số mới được cập nhật theo công thức: $\theta_{t+1} = \theta_t - v_{t+1}$

Ưu điểm:

- Tăng tốc độ hội tụ: Momentum giúp tăng tốc độ hội tụ của quá trình tối ưu hóa bằng cách giảm thiểu các dao động và zig-zags trong đường đi tìm kiếm cực tiểu.
- Vượt qua các điểm tối tiểu cục bộ: Nhờ vào khả năng vượt qua các điểm tối tiểu cục bộ, Momentum có thể tìm kiếm và hội tụ đến cực tiểu toàn cục tốt hơn trong không gian tối ưu.
- Ổn định hơn trong quá trình tối ưu hóa: Các đối tượng tối ưu hóa với Momentum ít bị dao động, giúp quá trình học ổn định hơn.

Nhược điểm:

- Khả năng vượt quá điểm tối ưu: Đôi khi, Momentum có thể khiến cho quá trình tối ưu hóa vượt qua điểm cực tiểu mong muốn và tăng tốc độ tối ưu hóa ở mức độ không cần thiết, dẫn đến việc không đạt được sự hội tụ chính xác.
- Cần chọn hệ số momentum phù hợp: Việc chọn giá trị hợp lý cho hệ số momentum có thể ảnh hưởng đáng kể đến hiệu suất của thuật toán. Nếu chọn giá trị không phù hợp, có thể dẫn đến hậu quả không mong muốn như việc vượt quá điểm cực tiểu hoặc hội tụ chậm.
- Khả năng overshooting: Trong một số trường hợp, Momentum có thể gây ra hiện tượng overshooting, khiến cho quá trình tối ưu hóa vượt qua điểm cực tiểu mong muốn và không ổn định.

1.1.5 Thuật toán Adagrad

[1]Adagrad là một phương pháp tối ưu hóa tỷ lệ học tập (learning rate) cho mỗi tham số dựa trên lịch sử của gradient của tham số đó. Ý tưởng chính là điều chỉnh tỷ lệ học tập của mỗi tham số dựa trên tần suất và độ lớn của gradient đã được quan sát. Không giống như các thuật toán trước đó thì learning rate hầu như giống nhau trong quá trình training (learning rate là hằng số), Adagrad coi learning rate là 1 tham số. Tức là Adagrad sẽ cho learning rate biến thiên sau mỗi thời điểm t.

Công thức cập nhật: Trong Adagrad, cập nhật của tham số $\boldsymbol{\theta_{t+1}}$ tại thời điểm t+1 được tính bằng công thức sau.

$$\boldsymbol{\theta_{t+1,i}} = \boldsymbol{\theta_{t,i}} - \frac{\boldsymbol{\eta}}{\sqrt{\boldsymbol{G_{t,ii}} + \in}} \times \boldsymbol{g_{t,1}}$$

- $\theta_{t+1,i}$ là giá trị cập nhật tiếp theo của tham số i.
- $\theta_{t,i}$ là giá trị hiện tại của tham số i.
- η là tỷ lệ học tập (learning rate)
- $\boldsymbol{G_{t,i}}$ là gradient của tham số i tại bước thời gian t
- $\in$ là một hằng số nhỏ (thường là khoảng 10^{-8} thêm vào trong mẫu để tránh việc chia cho 0.
- $\boldsymbol{G_{t,ii}}$ là tổng bình phương của các gradient trước đó cho tham số i cho đến thời điểm t

Điều chỉnh tỷ lệ học tập: Adagrad điều chỉnh tỷ lệ học tập của mỗi tham số dựa trên lịch sử của gradient đã được quan sát. Nếu gradient của một tham số lớn, nó sẽ nhận được một tỷ lệ học tập thấp hơn và ngược lại. Điều này giúp giảm độ nhạy cảm của việc chọn tỷ lệ học tập và cải thiện hiệu suất huấn luyện cho các mô hình học máy.

Ưu điểm:

- Tự điều chỉnh tỷ lệ học tập dựa trên lịch sử gradient.
- Hiệu suất tốt cho các tham số có gradient lớn.

Nhược điểm:

- Có thể dẫn đến việc tỷ lệ học tập giảm đi quá nhanh vì việc tích lũy bình phương gradient có thể làm cho mẫu trong công thức trở nên quá lớn, dẫn đến việc chia tỷ lệ học tập đi một giá trị rất nhỏ.

1.1.6 Thuật toán RMSPROP

[1]Thuật toán RMSprop cũng là một thuật toán tối ưu hóa gradient trong học máy và huấn luyện mô hình. Nó được thiết kế để cải thiện một số nhược điểm của Adagrad, đặc biệt là vấn đề về việc giảm tỷ lệ học tập quá nhanh do tích lũy bình phương gradient. RMSprop giải quyết vấn đề tỷ lệ học giảm dần của Adagrad bằng cách chia tỷ lệ học cho trung bình của bình phương gradient.

Ý tưởng cơ bản của RMSprop: RMSprop sử dụng ý tưởng của Adagrad nhưng thay đổi cách tính toán tổng bình phương gradient. Thay vì tích lũy tất cả gradient trước đó, RMSprop sử dụng trung bình có trọng số của bình phương gradient.

Công thức cập nhật trong RMSprop:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[g^2]t,i+\in}} \times g_{t,i}$$

- $\theta_{t+1,i}$ là giá trị cập nhật tiếp theo của tham số i
- $\theta_{t,i}$ là giá trị hiện tại của tham số i
- η là tỷ lệ học tập (learning rate)
- $g_{t,i}$ là gradient của tham số i tại bước thời gian t
- $E[g^2]t,i$ là trung bình có trọng số của bình phương gradient $g_{t,i}$ tới thời điểm t cho tham số i
- $\in$ là một hằng số nhỏ (thường là khoảng 10^{-8} thêm vào trong mẫu để tránh việc chia cho 0.

Điều chỉnh tỷ lệ học tập trong RMSprop: RMSprop sử dụng trung bình có trọng số của bình phương gradient để điều chỉnh tỷ lệ học tập. Điều này giúp giảm sự phụ thuộc quá mức vào gradient lớn và ngăn chặn việc giảm tỷ lệ học tập quá nhanh như trong Adagrad.

Ưu điểm:

- Ưu điểm rõ nhất của RMSprop là giải quyết được vấn đề tốc độ học giảm dần của Adagrad (vấn đề tốc độ học giảm dần theo thời gian sẽ khiến việc training chậm dần, có thể dẫn tới bị đóng băng)

Nhược điểm:

- Thuật toán RMSprop có thể cho kết quả nghiệm chỉ là local minimum chứ không đạt được global minimum như Momentum. Vì vậy người ta sẽ kết hợp cả 2 thuật toán Momentum với RMSprop cho ra 1 thuật toán tối ưu Adam. Chúng ta sẽ trình bày nó trong phần sau.

1.1.7 Thuật toán Adam

[2]Thuật toán Adam (Adaptive Moment Estimation) là một thuật toán tối ưu hóa gradient phổ biến và mạnh mẽ trong học máy và việc huấn luyện mô hình. Adam kết hợp cả RMSprop và một phần tử điều chỉnh động cho tỷ lệ học tập để cải thiện hiệu suất tối ưu hóa mô hình. Nếu giải thích theo hiện tượng vật lí thì Momentum giống như 1 quả cầu lao xuống dốc, còn Adam như 1 quả cầu rất nặng có ma sát, vì vậy nó dễ dàng vượt qua local minimum tới global minimum và khi tới global minimum nó không mất nhiều thời gian dao động qua lại quanh đích vì nó có ma sát nên dễ dừng lại hơn.

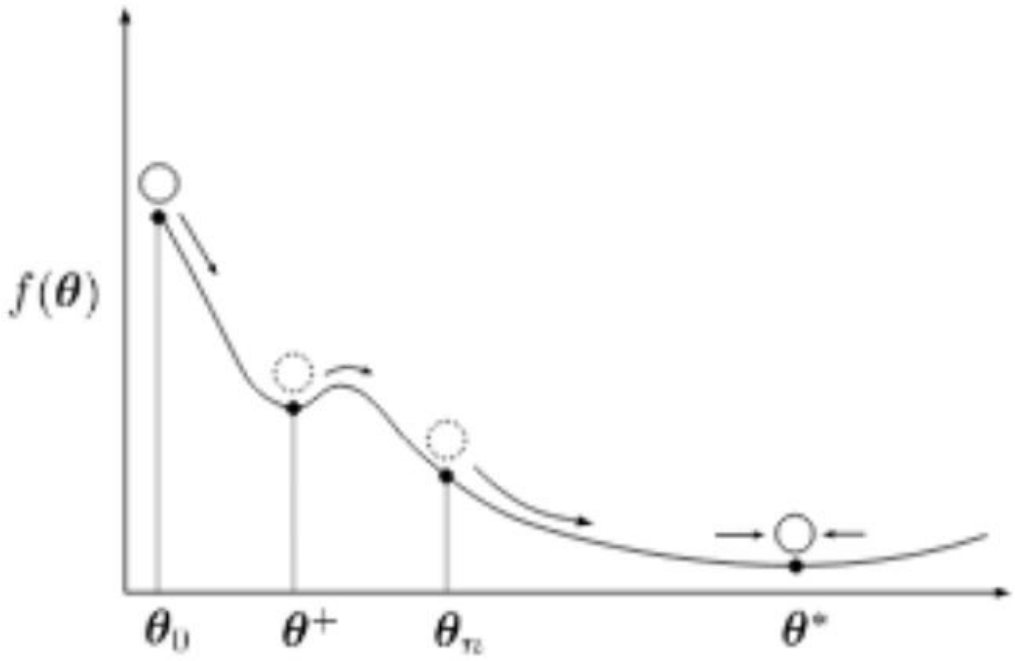


Figure 2: Heavy Ball with Friction, where the ball with mass overshoots the local minimum θ^+ and settles at the flat minimum θ^*.

Hình 5: Minh họa cho thuật toán Adam

Ý tưởng cơ bản của Adam: Adam kết hợp hai ý tưởng chính: ý tưởng của RMSprop về việc sử dụng trung bình có trọng số của bình phương gradient và ý tưởng về việc sử dụng một phần tử điều chỉnh động cho tỷ lệ học tập.

Công thức cập nhật trong Adam:

$$m_{t+1} = \beta_{1 \times m_t} + (1 - \beta_1) \times g_t$$

$$v_{t+1} = \beta_{2 \times v_t} + (1 - \beta_2) \times g_t^2$$

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \beta_1^{t+1}}$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1 - \beta_2^{t+1}}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_{t+1} + \varepsilon}} \times \hat{m}_{t+1}$$

- m_t và v_t là các ước lượng của bình phương gradient và bình phương gradient đã được thay đổi bởi các hệ số mô-men β_1 và β_2 tương ứng.
- g_t là gradient tại bước thời gian t.
- m_t và v_t là sự điều chỉnh của m_t và v_t để khắc phục hiệu ứng của việc khởi tạo ban đầu ở 0.
- η là tỷ lệ học tập (learning rate).
- $\in$ là một hằng số nhỏ (thường là khoảng 10^{-8} thêm vào trong mẫu để tránh việc chia cho 0.
- t là thời điểm hiện tại.

Điều chỉnh tỷ lệ học tập trong Adam: Adam sử dụng sự điều chỉnh của m_t và v_t để tính toán tỷ lệ học tập tại mỗi bước, giúp kiểm soát tỷ lệ học tập cho từng tham số trong quá trình huấn luyện mô hình.

Ưu điểm:

- Hiệu suất tốt trong việc tối ưu hóa mô hình.
- Tự điều chỉnh tỷ lệ học tập cho từng tham số.
- Hoạt động tốt với dữ liệu có nhiễu và mô hình lớn.

Nhược điểm:

- Có một số hằng số cần phải điều chỉnh, như β_1, β_2 và ϵ.

1.1.8 Tổng kết

Thuật toán	Ưu điểm	Nhược điểm
Gradient Descent	- Đơn giản và dễ triển khai. - Hội tụ đến điểm cực tiểu toàn cục nếu hàm mục tiêu là lồi và có đạo hàm liên tục.	- Tốn thời gian nếu tập dữ liệu lớn. - Dễ rơi vào điểm cực tiểu cục bộ. - Cần tính toán toàn bộ tập dữ liệu ở mỗi bước cập nhật.
Stochastic Gradient Descent	- Nhanh hơn do chỉ sử dụng một mẫu dữ liệu mỗi lần cập nhật.	- Không ổn định, dao động lớn vì gradient từ từng mẫu dữ liệu có thể khá nhiễu.

	- Dễ dàng áp dụng cho tập dữ liệu lớn	- Khó điều chỉnh tỷ lệ học tập một cách hiệu quả.
Momentum	- Giúp tránh khỏi sự dao động quá mức và tăng tốc độ hội tụ trong quá trình huấn luyện. - Hiệu quả khi hàm mục tiêu có nhiều đồi lớn.	- Cần điều chỉnh hệ số Momentum.
Adagrad	- Tự điều chỉnh tỷ lệ học tập cho từng tham số dựa trên lịch sử của gradient. - Hiệu suất tốt cho các tham số có gradient lớn.	- Có thể dẫn đến việc tỷ lệ học tập giảm đi quá nhanh vì việc tích lũy bình phương gradient.
RMSprop	- Giảm vấn đề của việc giảm tỷ lệ học tập quá nhanh trong Adagrad bằng cách sử dụng trung bình có trọng số của bình phương gradient. - Hiệu suất tốt trong việc tối ưu hóa mô hình.	- Vẫn còn một số hằng số cần phải điều chỉnh.
Adam	- Kết hợp ưu điểm của RMSprop và Momentum, cung cấp hiệu suất cao và ổn định. - Tự điều chỉnh tỷ lệ học tập và giúp hội tụ nhanh chóng.	- Cần điều chỉnh nhiều hệ số.

Bảng 1: Bảng so sánh các phương pháp Optimizer

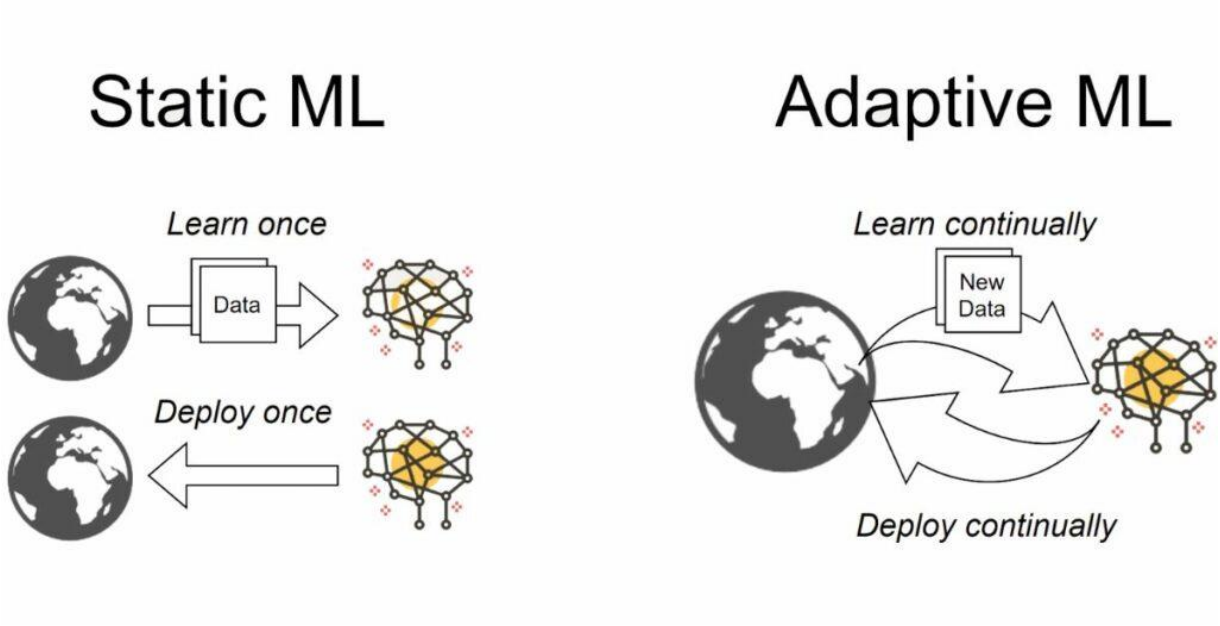
Tổng kết: Mỗi thuật toán có ưu điểm và nhược điểm riêng. Lựa chọn thuật toán phù hợp phụ thuộc vào bản chất của bài toán, cấu trúc mô hình, và tập dữ liệu được sử dụng. Trong thực tế, việc kết hợp và tinh chỉnh các phương pháp cũng thường được áp

dụng để tối ưu hóa quá trình huấn luyện mô hình. Adam thường được xem là một lựa chọn mạnh mẽ và phổ biến trong nhiều trường hợp.

1.2 Tìm hiểu về Continual Learning và Test Production

1.2.1 Continual Learning

[3]Continual Learning (CL), hay còn gọi là Lifelong Learning, là một lĩnh vực nghiên cứu trong học máy tập trung vào việc phát triển các mô hình có khả năng liên tục học hỏi từ dữ liệu mới mà không quên đi kiến thức đã học từ trước đó. Mục tiêu chính của Continual Learning là giải quyết vấn đề "catastrophic forgetting" (quên kinh hoàng), khi một mô hình học máy mất đi khả năng dự đoán hoặc hiệu suất trên dữ liệu cũ khi được huấn luyện lại trên dữ liệu mới.



Hình 6: Continual Learning

Trong học máy truyền thống, một mô hình được đào tạo trên một tập dữ liệu cố định và dự kiến sẽ thực hiện một nhiệm vụ duy nhất. Tuy nhiên, cách tiếp cận này trở nên có vấn đề khi dữ liệu và nhiệm vụ thay đổi và linh hoạt, vì mô hình phải có khả năng thích ứng và học hỏi từ dữ liệu mới theo thời gian. Đây là lúc CL phát huy tác dụng, cho phép mô hình liên tục học hỏi và cải thiện mà không quên kiến thức trước đó.

Một trong những thách thức đáng kể trong CL là vấn đề quên thảm khốc, trong đó một mô hình được đào tạo về nhiều nhiệm vụ cần ghi nhớ thông tin đã học được từ các nhiệm vụ trước đó khi tiếp xúc với dữ liệu mới. Nhiều kỹ thuật khác nhau đã được phát triển để vượt qua thách thức này, bao gồm cả mạng chính quy hóa và mạng tăng cường bộ nhớ.

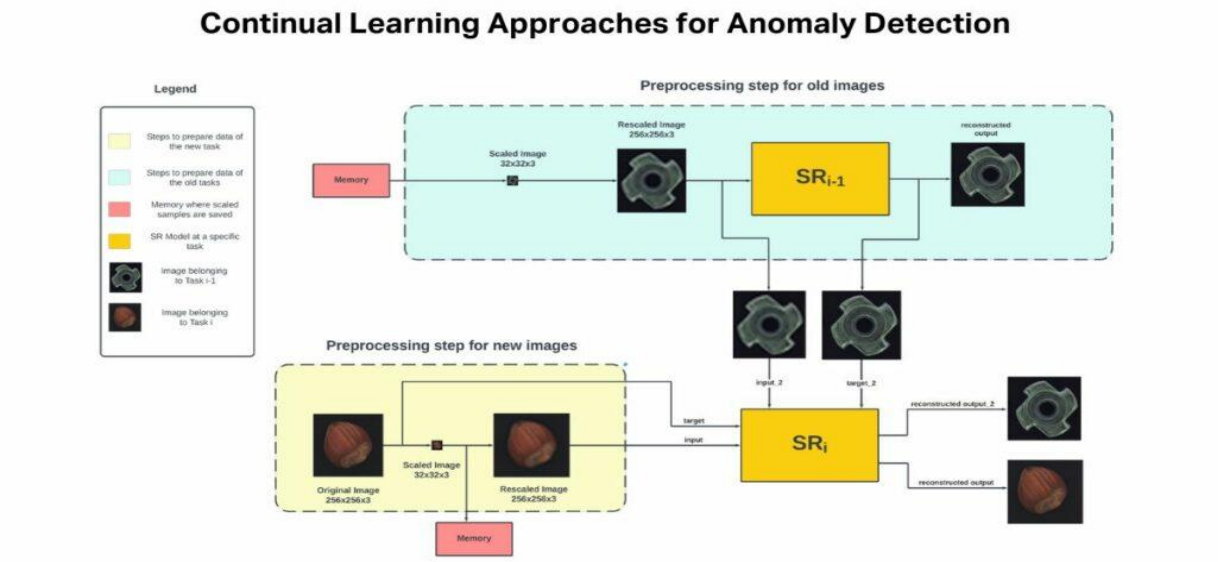
Chính quy hóa liên quan đến việc thêm các ràng buộc vào quá trình học tập để ngăn chặn việc trang bị quá mức cho dữ liệu mới. Mặt khác, mạng tăng cường bộ nhớ bao gồm các thành phần bộ nhớ lưu trữ thông tin từ các tác vụ trước đó và sử dụng thông tin này để cải thiện hiệu suất trong các tác vụ mới.

Kiến trúc của một mô hình cũng đóng một vai trò quan trọng trong khả năng thích ứng của nó. Một số mô hình được thiết kế để linh hoạt và dễ thích ứng hơn, với các kiến trúc có thể kết hợp thông tin và kiến thức mới dễ dàng hơn. Ví dụ, kiến trúc mô-đun cho phép các thành phần khác nhau của mô hình được đào tạo và điều chỉnh độc lập, giúp tăng tính linh hoạt trong việc thích ứng với các nhiệm vụ và dữ liệu mới.

Tính sẵn có của dữ liệu dành riêng cho nhiệm vụ cũng ảnh hưởng đến khả năng thích ứng của mô hình. Các mô hình có quyền truy cập vào lượng lớn dữ liệu dành riêng cho nhiệm vụ sẽ có khả năng thích ứng và học hỏi tốt hơn vì chúng có nhiều thông tin hơn để làm cơ sở cho dự đoán của mình. Đây là lý do tại sao một số mô hình được đào tạo về lượng dữ liệu khổng lồ để cải thiện khả năng thích ứng và khả năng khái quát hóa cho các nhiệm vụ mới.

Tóm lại, CL là một lĩnh vực quan trọng trong học máy nhằm giải quyết thách thức của các mô hình đào tạo có thể liên tục học hỏi và thích ứng với các nhiệm vụ và dữ liệu mới mà không quên kiến thức trước đó. Việc sử dụng các kỹ thuật như mạng chính quy hóa và tăng cường bộ nhớ, cũng như thiết kế kiến trúc mô hình, đóng một vai trò quan trọng trong khả năng thích ứng của các mô hình CL.

Ứng dụng của Continual Learning: Học tập liên tục mang lại một số lợi ích vốn có có thể phục vụ cho các trường hợp sử dụng sau.



Hình 7: Continual Learning ứng dụng cho Anomaly Detection

Phát hiện bất thường (Anomaly Detection) – Học liên tục có thể đặc biệt hữu ích trong các tình huống phát hiện bất thường trong đó việc phân phối dữ liệu thay đổi theo

thời gian và các thuật toán học máy truyền thống có thể không hiệu quả. Mục tiêu của kiểu học liên tục này là liên tục theo dõi hành vi bình thường của hệ thống để tìm hiểu các cấp độ vận hành, quy trình hoặc luồng dữ liệu tiêu chuẩn của nó và phát hiện các điểm bất thường bằng cách so sánh dữ liệu mới với hành vi bình thường đã học. Để thực hiện điều này, thuật toán học liên tục được huấn luyện trên luồng dữ liệu để tìm hiểu hành vi bình thường của hệ thống. Khi có dữ liệu mới, dữ liệu đó sẽ được so sánh với hành vi bình thường đã học được và những sai lệch so với tiêu chuẩn sẽ được gắn cờ là bất thường. Thuật toán học liên tục sau đó có thể được cập nhật để kết hợp dữ liệu mới, cho phép nó thích ứng với những thay đổi trong hành vi thông thường theo thời gian. Ví dụ: trong lĩnh vực tài chính, hành vi thông thường của các giao dịch có thể thay đổi theo thời gian khi các tác nhân độc hại phát triển các kỹ thuật mới để tránh bị phát hiện. Việc học hỏi liên tục có thể giúp phát hiện những điểm bất thường đang gia tăng này bằng cách cập nhật mô hình để nắm bắt hành vi đang thay đổi. Ngược lại với các thuật toán học máy truyền thống, được đào tạo trên một tập dữ liệu cố định và giả định rằng việc phân phối dữ liệu không thay đổi, các thuật toán học liên tục có thể liên tục thích ứng và cải thiện theo thời gian.

Cá nhân hóa (Personalization) – một trường hợp sử dụng khác của việc học tập liên tục là trong các hệ thống đề xuất được cá nhân hóa với mục tiêu là cung cấp các đề xuất cập nhật và có tính tùy chỉnh cao cho người dùng. Bằng cách liên tục tìm hiểu sở thích và hành vi của người dùng, hệ thống đề xuất có thể cải thiện khả năng đưa ra đề xuất chính xác và phù hợp. Để đạt được điều này, thuật toán học liên tục được đào tạo dựa trên dữ liệu tương tác lịch sử của người dùng, chẳng hạn như lịch sử mua hàng hoặc truy vấn tìm kiếm, để tìm hiểu sở thích và kiểu hành vi của họ. Khi có dữ liệu mới, mô hình sẽ cập nhật hiểu biết về sở thích và hành vi của người dùng. Thông tin cập nhật này sau đó được sử dụng để đưa ra các khuyến nghị chính xác và phù hợp hơn cho người dùng. Ví dụ: trong hệ thống thương mại điện tử, mô hình có thể sử dụng lịch sử mua hàng của người dùng để tìm hiểu về sở thích của họ đối với một số loại sản phẩm hoặc nhãn hiệu nhất định. Theo thời gian, khi người dùng tiếp tục mua hàng, mô hình sẽ liên tục cập nhật hiểu biết về sở thích của họ và sử dụng thông tin này để đưa ra đề xuất được cá nhân hóa cho các sản phẩm hoặc dịch vụ khác mà người dùng có thể quan tâm. Kiểu học hỏi liên tục này cho phép mô hình để thích ứng với việc thay đổi sở thích của người dùng và cung cấp các đề xuất ngày càng chính xác theo thời gian.

Dự báo (Forecasting) – Học tập liên tục cũng có thể được áp dụng trong lĩnh vực dự báo để liên tục cập nhật các dự đoán dựa trên dữ liệu mới khi có sẵn. Cách tiếp cận này cho phép các mô hình thích ứng với những thay đổi trong phân phối dữ liệu và duy trì độ chính xác của chúng theo thời gian, ngay cả khi xử lý dữ liệu phức tạp và động. Trong hệ thống dự báo, thuật toán học liên tục được huấn luyện trên luồng dữ liệu lịch sử, chẳng hạn như dữ liệu tài chính hoặc dữ liệu chuỗi thời gian, để đưa ra dự đoán về các sự kiện trong tương lai. Khi có dữ liệu mới, mô hình sẽ cập nhật hiểu biết của nó về các mối quan hệ trong dữ liệu và sử dụng thông tin này để tinh chỉnh các dự đoán của

nó. Ví dụ, trong hệ thống dự báo tài chính, thuật toán học liên tục có thể được huấn luyện trên luồng dữ liệu giá cổ phiếu để dự đoán giá cổ phiếu trong tương lai. Khi có dữ liệu mới, mô hình có thể liên tục cập nhật hiểu biết về xu hướng thị trường và sử dụng thông tin này để tinh chỉnh các dự đoán về giá cổ phiếu trong tương lai. Khả năng thích ứng và cải tiến liên tục này, ngay cả trong môi trường dữ liệu phức tạp và năng động, khiến việc học hỏi liên tục trở thành một công cụ có giá trị cho các ứng dụng dự báo

Phương pháp trong Continual Learning

- Regularization: Định chuẩn các trọng số của mô hình để giảm thiểu hiện tượng quên.
- Replay: Lưu trữ và tái chế dữ liệu cũ để huấn luyện lại mô hình.
- Distillation: Sử dụng kiến thức đã học từ quá khứ để huấn luyện mô hình mới.
- Memory-based Approaches: Sử dụng bộ nhớ để lưu trữ thông tin quan trọng từ.

Các kỹ thuật trong Continual Learning:

- Elastic Weight Consolidation (EWC): Đánh giá và bảo vệ các trọng số quan trọng cho nhiệm vụ trước.
- Generative Replay: Sử dụng mô hình sinh (generative model) để tạo dữ liệu giả mạo từ quá khứ để huấn luyện mô hình trên dữ liệu mới.
- Progressive Neural Networks (PNN): Mở rộng mô hình mạng nơ-ron khi thêm nhiệm vụ mới, giữ nguyên kiến thức đã học từ các nhiệm vụ trước đó.
- Meta-Learning: Huấn luyện mô hình để học cách học từ các nhiệm vụ khác nhau, từ đó áp dụng kiến thức đó vào các nhiệm vụ mới.

Continual Learning cho tương lai: Một khía cạnh thú vị khác của việc học tập liên tục là phát triển các phương pháp cho phép hệ thống trí tuệ nhân tạo liên tục học hỏi mà không cần giám sát. Các thuật toán học có giám sát truyền thống dựa vào dữ liệu huấn luyện được gắn nhãn để học các nhiệm vụ mới, nhưng trong nhiều ứng dụng trong thế giới thực, dữ liệu thường không được gắn nhãn hoặc không đầy đủ. Các phương pháp học liên tục không giám sát nhằm mục đích khắc phục hạn chế này bằng cách cho phép các hệ thống AI học các cách biểu diễn mới từ dữ liệu thô hoặc phi cấu trúc mà không cần giám sát rõ ràng. Như chúng ta đã thấy ở trên, các mô hình tổng quát, chẳng hạn như Bộ mã hóa tự động biến đổi (VAE) và Mạng đối thủ sáng tạo (GAN), đã cho thấy kết quả đầy hứa hẹn trong việc học liên tục không giám sát bằng cách tạo ra dữ liệu mới từ các biểu diễn hiện có.

Việc tích hợp học tập liên tục với các lĩnh vực khác của học máy là một khía cạnh thú vị khác có tiềm năng to lớn. Ví dụ, siêu học tập, là quá trình học để học, có thể được kết hợp với học tập liên tục để cho phép hệ thống AI nhanh chóng thích ứng với các

nhiệm vụ mới bằng cách tận dụng kinh nghiệm trước đó. Một lĩnh vực tích hợp khác là học tăng cường, trong đó hệ thống AI có thể được đào tạo để liên tục học các kỹ năng mới trong môi trường bằng cách tối đa hóa tín hiệu khen thưởng. Trong các hệ thống như vậy, các thuật toán học tập liên tục có thể được sử dụng để tránh tình trạng quên lãng nghiêm trọng, cho phép hệ thống AI xây dựng dựa trên kinh nghiệm và kiến thức trước đó khi gặp những thách thức mới.

Những đổi mới trong học tập liên tục có tiềm năng tác động lớn đến cả sự phát triển của học máy và các ứng dụng trong thế giới thực của nó. Trong lĩnh vực học máy, học liên tục có khả năng cải thiện đáng kể hiệu suất của các mô hình AI và giảm nhu cầu đào tạo lại thường xuyên. Bằng cách cho phép các mô hình AI liên tục học hỏi và thích ứng với dữ liệu và nhiệm vụ mới, các nhà nghiên cứu có thể tạo ra các hệ thống AI mạnh mẽ và linh hoạt hơn, có thể hoạt động trong môi trường năng động và xử lý việc phân phối dữ liệu đang phát triển. Điều này sẽ cho phép các hệ thống máy học duy trì độ chính xác theo thời gian và tránh tình trạng quên lãng nghiêm trọng, một thách thức lớn trong đào tạo AI truyền thống.

Trong thế giới kinh doanh, những đổi mới trong học tập liên tục có khả năng cách mạng hóa cách các tổ chức sử dụng AI để giải quyết các vấn đề phức tạp. Ví dụ: bằng cách kết hợp việc học tập liên tục vào hệ thống AI của mình, doanh nghiệp có thể sử dụng AI để liên tục theo dõi và phân tích lượng lớn dữ liệu, cung cấp thông tin chi tiết theo thời gian thực và đưa ra quyết định sáng suốt dựa trên các điều kiện phát triển. Điều này có thể đặc biệt có giá trị trong các ngành như tài chính, chăm sóc sức khỏe và bán lẻ, nơi lượng lớn dữ liệu phải được phân tích và xử lý nhanh chóng. Ngoài ra, học hỏi liên tục có thể giúp các tổ chức cá nhân hóa hệ thống AI của họ để đáp ứng nhu cầu riêng biệt của khách hàng và nhân viên, mang lại trải nghiệm phù hợp và hiệu quả hơn. Nhìn chung, tác động của việc học hỏi liên tục đối với hoạt động kinh doanh sẽ rất đáng kể, cho phép các tổ chức đi trước đối thủ và đạt được mục tiêu của mình một cách hiệu quả và năng suất hơn.

1.2.2 Tìm hiểu về Test Production

Test production trong machine learning đóng vai trò quan trọng trong việc chuyển đổi mô hình từ quá trình phát triển và thử nghiệm sang môi trường sản xuất thực tế. Đây là giai đoạn cuối cùng và quan trọng nhất trước khi mô hình được triển khai rộng rãi và sử dụng trong các ứng dụng thực tế. Quá trình này đòi hỏi sự cẩn thận, kiểm soát chặt chẽ và việc đánh giá toàn diện để đảm bảo tính ổn định, hiệu suất và đáng tin cậy của mô hình trong môi trường hoạt động thực tế.

Khám Phá Chi Tiết Test Production Trong Machine Learning

- Chuẩn Bị Dữ Liệu:Trước khi chuyển mô hình vào môi trường sản xuất, việc chuẩn bị dữ liệu là yếu tố quan trọng. Đảm bảo rằng dữ liệu trong môi trường sản xuất phản ánh đầy đủ các biến thể có thể xảy ra. Có thể cần tiến hành xử lý dữ liệu mới để phù hợp với mô hình hoặc thiết lập quy trình tự động hóa cho việc cập nhật dữ liệu mới.

- Triển Khai Mô Hình:Quá trình triển khai mô hình cần phải được thực hiện cẩn thận và có kế hoạch. Mô hình được triển khai vào môi trường sản xuất nhưng chưa được sử dụng để đưa ra quyết định hoặc tương tác trực tiếp với người dùng. Thay vào đó, nó hoạt động trong chế độ giám sát để thu thập dữ liệu và đánh giá hiệu suất.
- Kiểm Tra Tính Ổn Định:Một trong những mục tiêu chính của test production là đảm bảo rằng mô hình không gây ra lỗi hoặc sự cố không mong muốn trong môi trường sản xuất. Việc này bao gồm kiểm tra tính ổn định của mô hình trong mọi điều kiện và tình huống có thể xảy ra.
- Đánh Giá Hiệu Suất: Đánh giá hiệu suất của mô hình trong môi trường sản xuất là quan trọng để đảm bảo rằng nó hoạt động như dự kiến. Điều này bao gồm việc theo dõi các metric hiệu suất cụ thể (độ chính xác, recall, precision, F1-score, v.v.) và so sánh chúng với tiêu chuẩn đã đề ra từ quá trình thử nghiệm trước đó.
- Giám Sát Liên Tục và Điều Chỉnh: Test production không phải là bước duy nhất mà là quá trình liên tục. Việc giám sát mô hình sau khi triển khai là cần thiết để phát hiện sớm các vấn đề có thể phát sinh và thực hiện điều chỉnh khi cần thiết.
- Quản Lý Rủi Ro: Việc triển khai mô hình vào môi trường sản xuất cũng đi kèm với rủi ro, bao gồm việc mô hình không hoạt động chính xác, tác động không mong muốn đến người dùng hoặc hệ thống. Điều này yêu cầu kế hoạch dự phòng và cách xử lý khi có sự cố xảy ra.
- Thử Nghiệm Dữ Liệu Mới: Việc kiểm tra mô hình trên dữ liệu mới là cần thiết để đảm bảo rằng nó có thể xử lý các tình huống mới mà nó chưa từng trải qua trong quá trình huấn luyện và thử nghiệm.
- Test production không chỉ là việc chuyển đổi mô hình từ một giai đoạn sang giai đoạn khác, mà còn là một quá trình liên tục và kỹ lưỡng. Việc thực hiện một kiểm thử chặt chẽ và có kế hoạch có thể giúp cải thiện tính ổn định và hiệu suất của mô hình trong môi trường thực tế, tăng cường tin cậy và sự hài lòng của người dùng.

CHƯƠNG 2 – CÂU 2

2.1 Phân tích, thống kê, hiểu dữ liệu

Tên dữ liệu: Customer – Churn – Records

Link github: T-TNThien/Final-Report-Machine-Learning (github.com)

Tóm tắt: Dữ liệu trên là tập hợp về thông tin khách hàng của một ngân hàng về thông tin cá nhân, hành vi giao dịch, các chỉ số về sử dụng sản phẩm/ dịch vụ. Dữ liệu này được sử dụng để phân tích lý do vì sao khách hàng chuyển đổi hoặc rời bỏ, từ đó tạo ra các chiến lược giữ chân khách hàng, cải thiện dịch vụ hoặc điều chỉnh sản phẩm để giữ chân khách hàng tốt hơn.

Giới thiệu dữ liệu

STT	Tên	Thông tin
1	RowNumber	Số thứ tự dòng
2	Customerid	ID của khách hàng
3	Surname	Họ của khách hàng
4	CreditScore	Điểm tín dụng của khách hàng
5	Geography	Quốc tịch của khách hàng
6	Gender	Giới tính
7	Age	Tuổi
8	Tenure	Thời gian sở hữu tài khoản
9	Balance	Số dư trong tài khoản
10	NumOfProducts	Số lượng sản phẩm sử dụng
11	HasCrCard	Có thẻ tín dụng hay không
12	IsActiveMember	Khách hàng có hoạt động không
13	EstimatedSalary	Mức lương ước tính
14	Exited	Khách hàng đã rời đi không

15	Complain	Khách hàng có khiếu nại không
16	Satisfaction	Điểm hài lòng
17	Card Type	Loại thẻ của khách hàng
18	Point Earned	Số điểm đã kiếm được

Bảng 2: Bảng phân tích dữ liệu

Thông tin dữ liệu: Dữ liệu bao gồm 1000 dòng và 18 cột
Sử dụng câu lệnh "df.info()" để xem chi tiết hơn.

```
df.info()
✓ 0.0s
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   RowNumber           10000 non-null  int64
 1   CustomerId          10000 non-null  int64
 2   Surname             10000 non-null  object
 3   CreditScore         9990 non-null   float64
 4   Geography           10000 non-null  object
 5   Gender              10000 non-null  object
 6   Age                 9985 non-null   float64
 7   Tenure              10000 non-null  int64
 8   Balance             9979 non-null   float64
 9   NumOfProducts       10000 non-null  int64
 10  HasCrCard           10000 non-null  int64
 11  IsActiveMember      10000 non-null  int64
 12  EstimatedSalary     9979 non-null   float64
 13  Exited              10000 non-null  int64
 14  Complain            10000 non-null  int64
 15  Satisfaction Score  10000 non-null  int64
 16  Card Type           10000 non-null  object
 17  Point Earned        9979 non-null   float64
dtypes: float64(5), int64(9), object(4)
memory usage: 1.4+ MB
```

Hình 8: Thông tin dữ liệu

Có thể thấy dữ liệu trong quá trình thu thập đã bị thiếu vì thế các giá trị bị thiếu sẽ được thay thế bằng giá trị trung bình của dữ liệu nằm bên trong cột bị thiếu.

```
for column in df.columns:
    if df[column].dtype in ['float64', 'int64']:
        column_mean = df[column].mean()
        df[column].fillna(column_mean, inplace=True)
```

✓ 0.0s

Hình 9: Thay thế dữ liệu bằng giá trị trung bình

Loại bỏ những dữ liệu không cần thiết cho quá trình phân tích và xác định các giá trị rời rạc và giá trị liên tục của dữ liệu.

```
df.drop(columns=["RowNumber", "CustomerId", "Surname"], inplace=True)
```

✓ 0.0s

Đếm số lượng giá trị duy nhất

```
df.nunique()
```

✓ 0.0s

```
CreditScore           461
Geography               3
Gender                  2
Age                    71
Tenure                 11
Balance              6364
NumOfProducts           4
HasCrCard               2
IsActiveMember          2
EstimatedSalary      9979
Exited                  2
Complain                2
Satisfaction Score      5
Card Type               4
Point Earned          786
dtype: int64
```

Hình 10: Xử lý dữ liệu

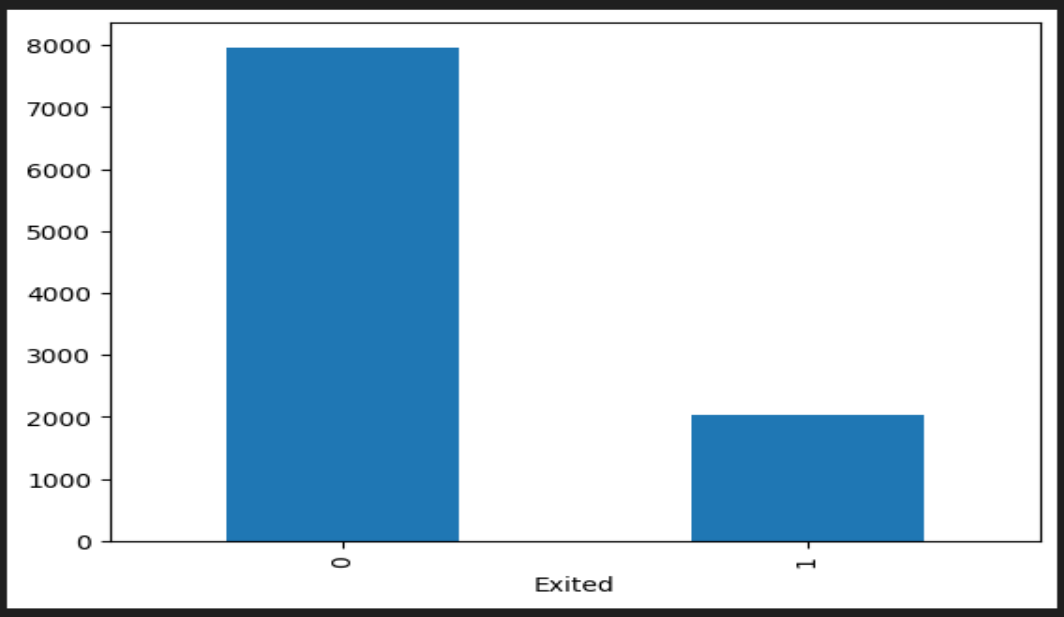
Thông qua câu lệnh “df.nunique()” có thể thấy dữ liệu sau khi loại bỏ 3 cột có 6 cột chứa biến liên tục và 9 cột chứa biến rời rạc, các biến đó sẽ được biểu diễn qua đồ

thị với biểu đồ cột (biến rời rạc) và biểu đồ Histogram(biến liên tục). Đầu tiên chúng ta sẽ biểu diễn cột "Exited". Đây sẽ mục tiêu huấn luyện mô hình học máy

```
g = df['Exited'].value_counts()
print(g)
g.plot.bar()
plt.show()
```

[10] ✓ 0.2s

```
Exited
0    7962
1    2038
Name: count, dtype: int64
```

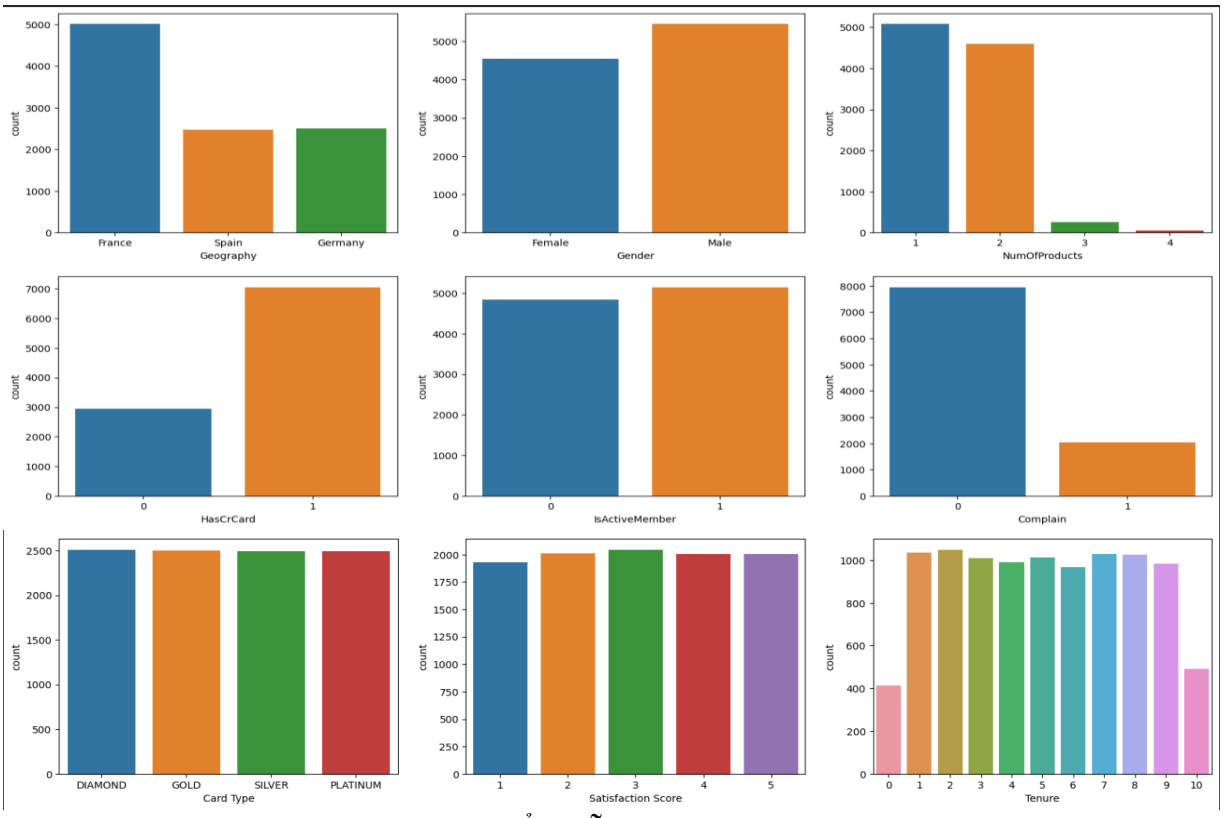


Hình 11: Biểu diễn cột Exited

Biểu diễn các giá trị rời rạc.

```
fig, axes = plt.subplots(3, 3, figsize = (20, 15))
sns.countplot ( ax = axes[0, 0], data = df, x = "Geography")
sns.countplot ( ax = axes[0, 1], data = df, x = "Gender")
sns.countplot ( ax = axes[0, 2], data = df, x = "NumOfProducts")
sns.countplot ( ax = axes[1, 0], data = df, x = "HasCrCard")
sns.countplot ( ax = axes[1, 1], data = df, x = "IsActiveMember")
sns.countplot ( ax = axes[1, 2], data = df, x = "Complain")
sns.countplot ( ax = axes[2, 0], data = df, x = "Card Type")
sns.countplot ( ax = axes[2, 1], data = df, x = "Satisfaction Score")
sns.countplot ( ax = axes[2, 2], data = df, x = "Tenure")
plt.show()
```

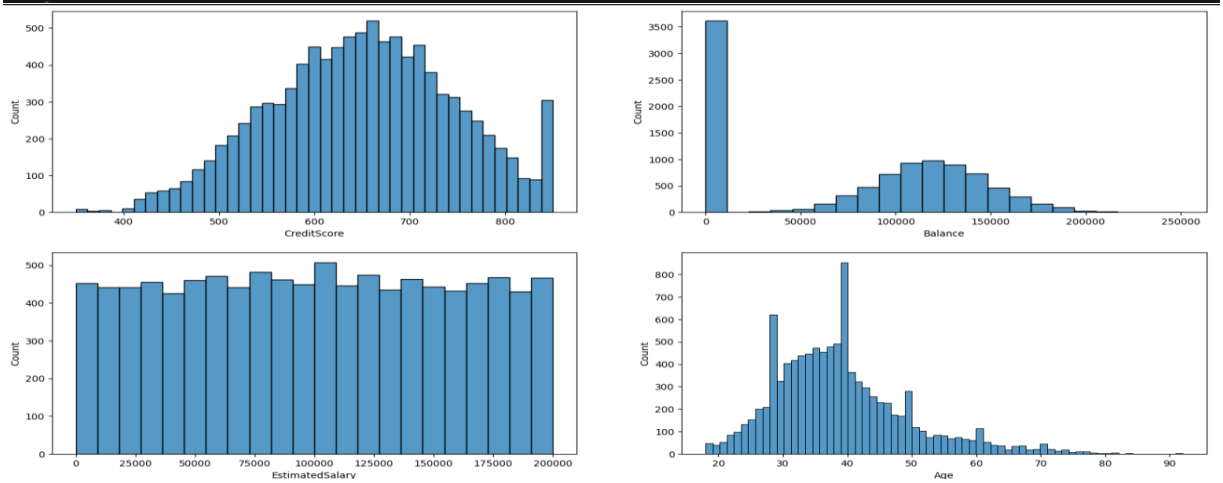
✓ 1.0s

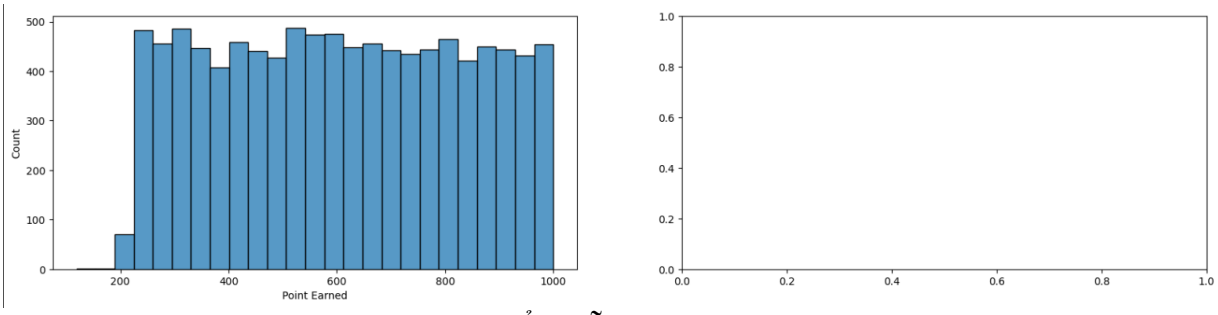


Hình 12: Biểu diễn các giá trị rời rạc

Biểu diễn các giá trị liên tục.

```
fig, axes = plt.subplots(3, 2, figsize = (20, 15))
sns.histplot ( ax = axes[0, 0], data = df, x = "CreditScore")
sns.histplot ( ax = axes[0, 1], data = df, x = "Balance")
sns.histplot ( ax = axes[1, 0], data = df, x = "EstimatedSalary")
sns.histplot ( ax = axes[1, 1], data = df, x = "Age")
sns.histplot ( ax = axes[2, 0], data = df, x = "Point Earned")
plt.show()
```

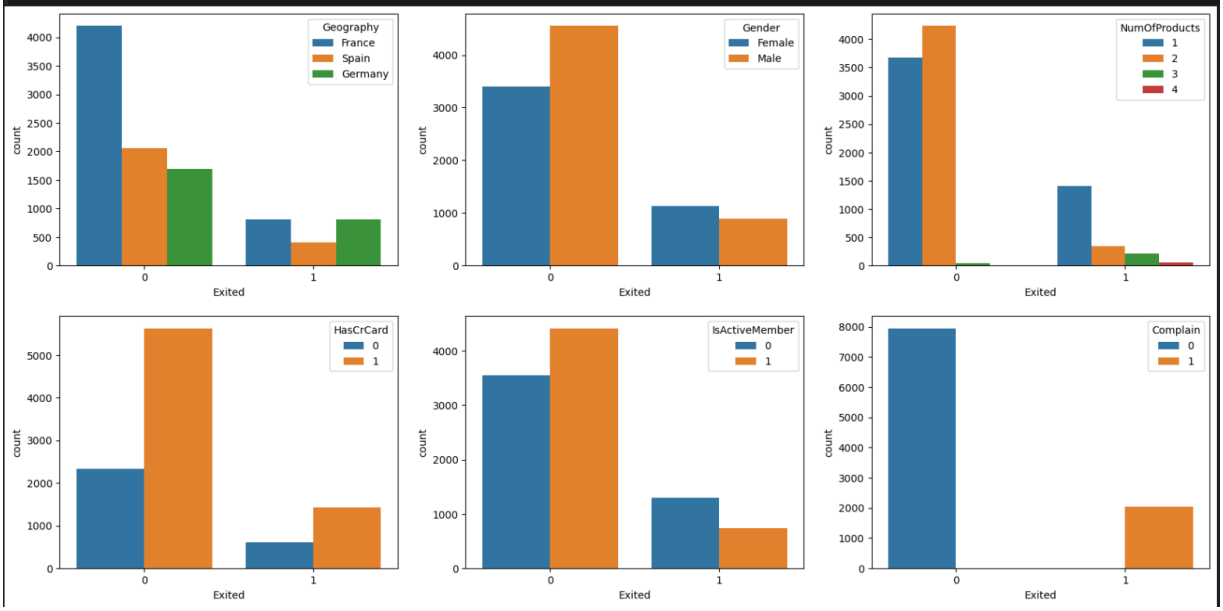


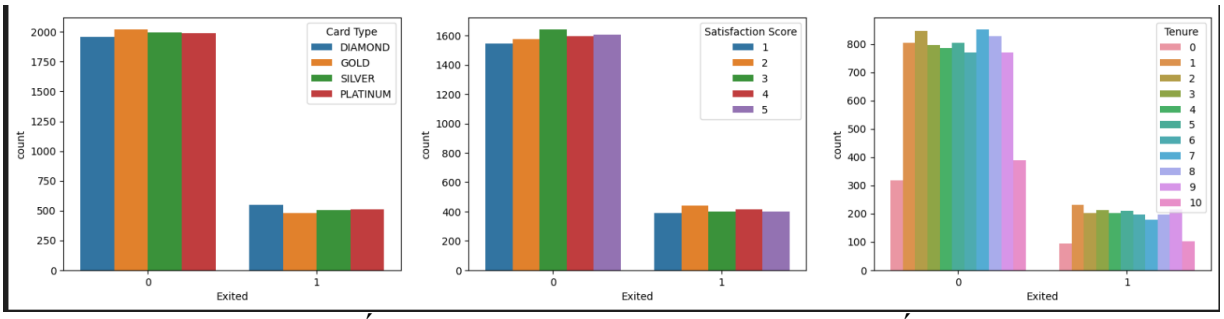


Hình 13: Biểu diễn các giá trị rời rạc

Biểu diễn mối liên quan giữa các cột "Exited" và các cột chứa giá trị rời rạc.

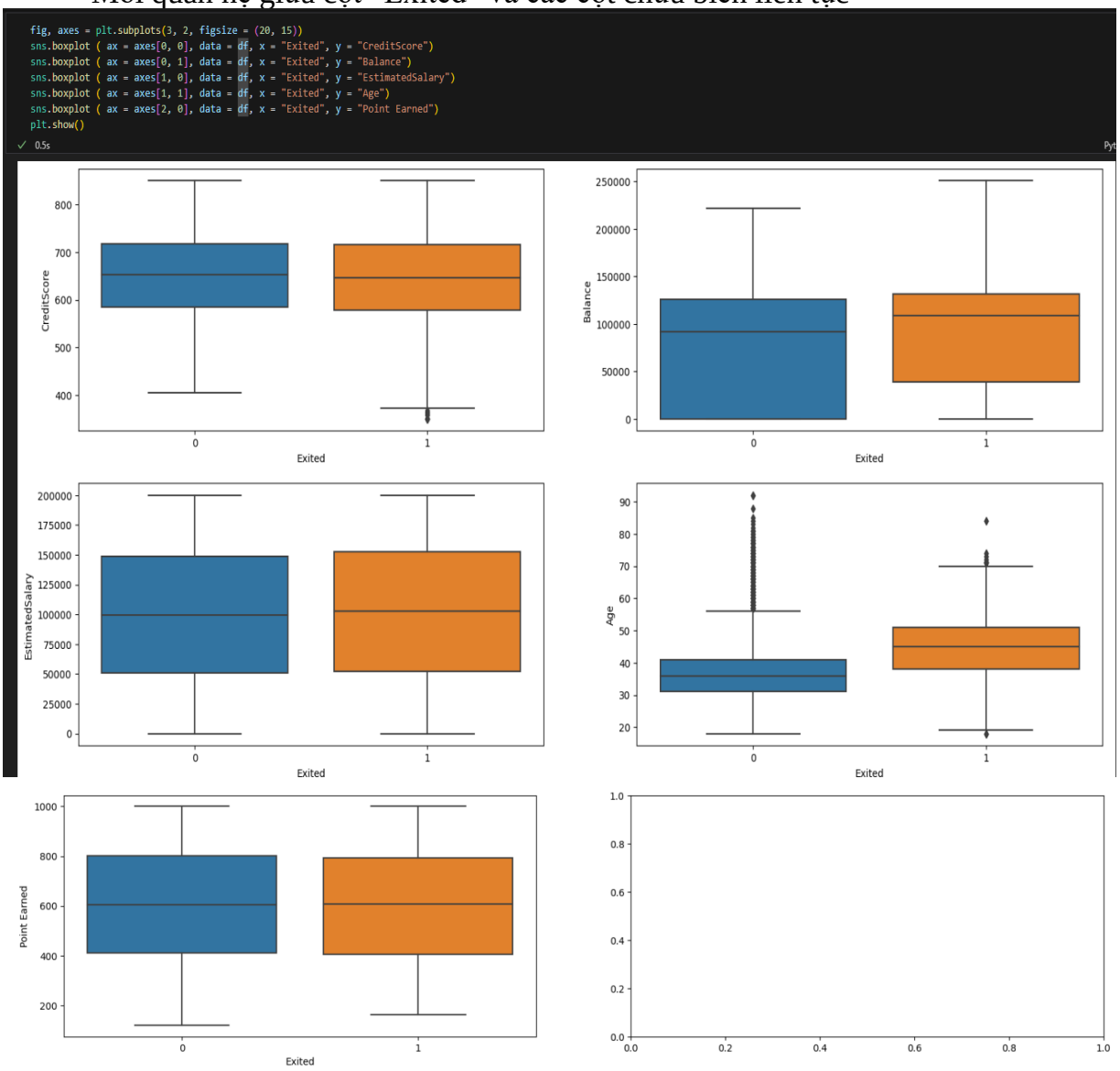
```
fig, axes = plt.subplots(3, 3, figsize = (20, 15))
sns.countplot ( ax = axes[0, 0], data = df, x = "Exited", hue = "Geography")
sns.countplot ( ax = axes[0, 1], data = df, x = "Exited", hue = "Gender")
sns.countplot ( ax = axes[0, 2], data = df, x = "Exited", hue = "NumOfProducts")
sns.countplot ( ax = axes[1, 0], data = df, x = "Exited", hue = "HasCrCard")
sns.countplot ( ax = axes[1, 1], data = df, x = "Exited", hue = "IsActiveMember")
sns.countplot ( ax = axes[1, 2], data = df, x = "Exited", hue = "Complain")
sns.countplot ( ax = axes[2, 0], data = df, x = "Exited", hue = "Card Type")
sns.countplot ( ax = axes[2, 1], data = df, x = "Exited", hue = "Satisfaction Score")
sns.countplot ( ax = axes[2, 2], data = df, x = "Exited", hue = "Tenure")
plt.show()
✓ 1.0s
```





Hình 14: Mối quan hệ giữa cột "Exited" và các biến rời rạc

Mối quan hệ giữa cột "Exited" và các cột chứa biến liên tục



Hình 15: Mối quan hệ giữa cột "Exited" và các cột chứa biến liên tục

Kết luận: Qua phân tích bằng biểu đồ có thể thấy các cột "Card Type", "Satisfaction Score", "Point Earned", "CreditScore", "EstimatedSalary" không có tác động nhiều đến quyết định ở lại hay rời đi của khách hàng, chính vì thế nó sẽ được loại bỏ để tránh gây nhiễu trong quá trình xây dựng Model.

```
df.drop(columns=["Card Type", "Satisfaction Score", "Point Earned", "CreditScore", "EstimatedSalary"], inplace=True)
#df.drop(columns=["Complain" ], inplace=True)
df.head()
```

✓ 0.0s

	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Exited	Complain
0	France	Female	42.0	2	0.000000	1	1	1	1	1
1	Spain	Female	41.0	1	83807.860000	1	0	1	0	1
2	France	Female	42.0	8	76417.752209	3	1	0	1	1
3	France	Female	39.0	1	0.000000	2	0	0	0	0
4	Spain	Female	43.0	2	125510.820000	1	1	1	0	0

Hình 16: Loại bỏ các dữ liệu không cần thiết

2.2 Xây dựng model

2.2.1 Tiền xử lý dữ liệu

Chuyển đổi dữ liệu bằng phương pháp Label Encoding.

```
## chuyển đổi dữ liệu
types = df.dtypes
names = list(df.columns)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for i in range(len(types)):
    if types[i]=='object':
        le.fit_transform(df[names[i]])
        df[names[i]] = le.transform(df[names[i]])
```

✓ 0.0s

Hình 17: Chuyển đổi dữ liệu

Lựa chọn dữ liệu để training và Test.

Lựa chọn dữ liệu

```
columns_to_select = df.columns[df.columns != 'Exited']
X = df.loc[:, columns_to_select]
y = df.iloc[:, df.columns.get_loc('Exited')]

#X.head()
y.head()
#print(X.shape)
#print(y.shape)
```

✓ 0.0s

Hình 18: Lựa chọn dữ liệu

Chuẩn hóa dữ liệu bằng phương pháp MinmaxScaler.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
print(X_scaled[0])
```

✓ 0.0s

Hình 19: Chuẩn hóa dữ liệu

Phân chia dữ liệu theo tỷ lệ 70% train và 30% test.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.30, random_state= 42)
print(len(X_train), len(X_test), len(y_train), len(y_test))
```

✓ 0.0s

Hình 20: Phân chia dữ liệu

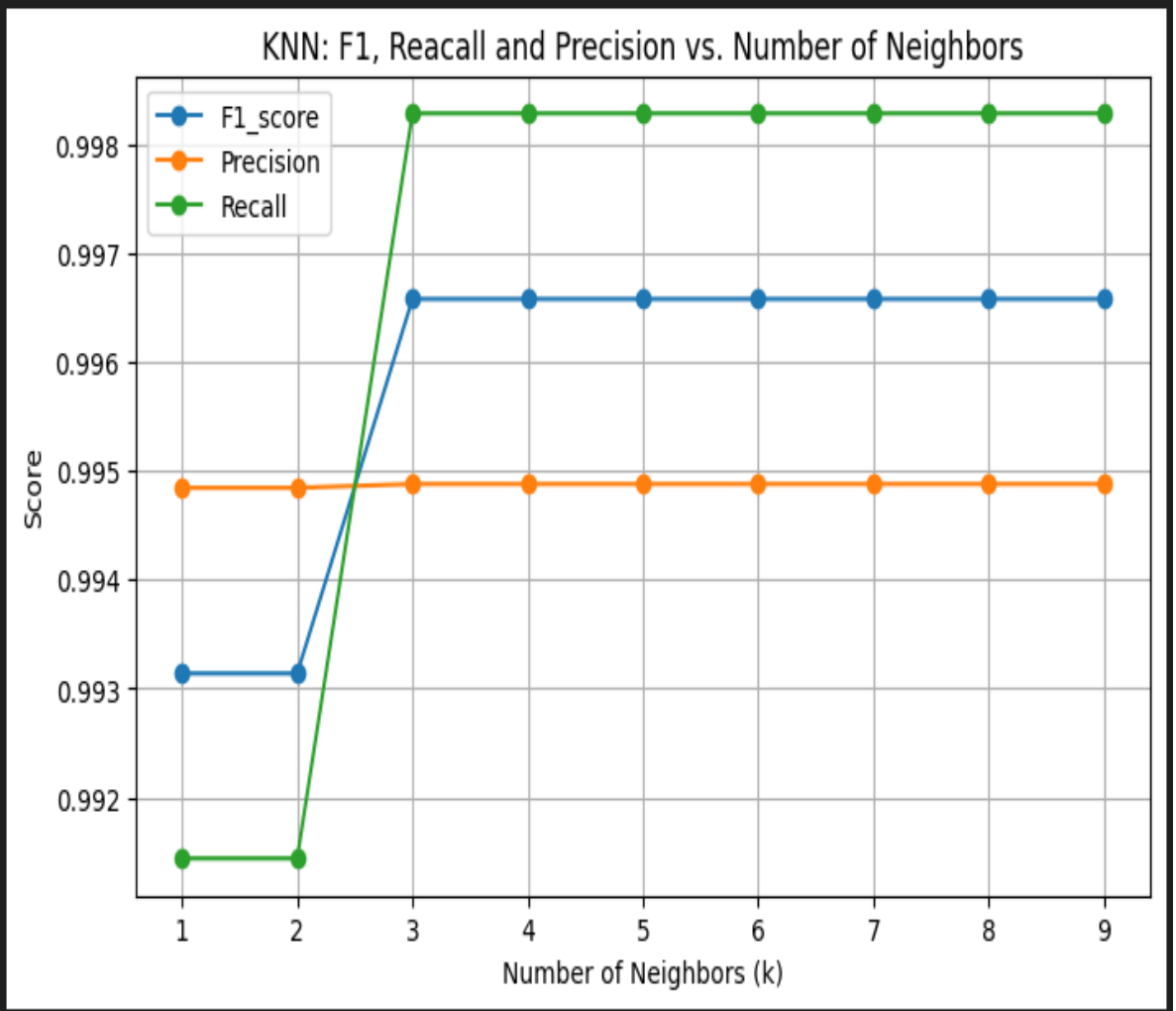
2.3 Xây dựng model dựa trên thuật toán cơ bản

Xây dựng model dựa trên thuật toán Knn.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score, precision_score, recall_score
# Khởi tạo các danh sách để lưu giá trị
k_values = []
f1_scores = []
precision_scores = []
recall_scores = []

for k in range(1, 10):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_predict = knn.predict(X_test)


    f1 = f1_score(y_test, y_predict)
    precision = precision_score(y_test, y_predict)
    recall = recall_score(y_test, y_predict)


    print(f"K = {k}, F1 = {f1}, Precision = {precision}, , Recall = {recall}")

    # Lưu giá trị của k, accuracy score và precision score vào danh sách tương ứng
    k_values.append(k)
    f1_scores.append(f1)
    precision_scores.append(precision)
    recall_scores.append(recall)



# Vẽ đồ thị
plt.figure(figsize=(8, 5))

plt.plot(k_values, f1_scores, marker='o', label='F1_score')
plt.plot(k_values, precision_scores, marker='o', label='Precision')
plt.plot(k_values, recall_scores, marker = 'o', label = 'Recall')

plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Score')
plt.title('KNN: F1, Reacall and Precision vs. Number of Neighbors')
plt.legend()
plt.grid(True)
plt.show()
```

Hình 21: Model dựa trên thuật toán Knn.

```
K = 1, F1 = 0.9931389365351629, Precision = 0.9948453608247423, , Recall = 0.9914383561643836
K = 2, F1 = 0.9931389365351629, Precision = 0.9948453608247423, , Recall = 0.9914383561643836
K = 3, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
K = 4, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
K = 5, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
K = 6, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
K = 7, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
K = 8, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
K = 9, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
```



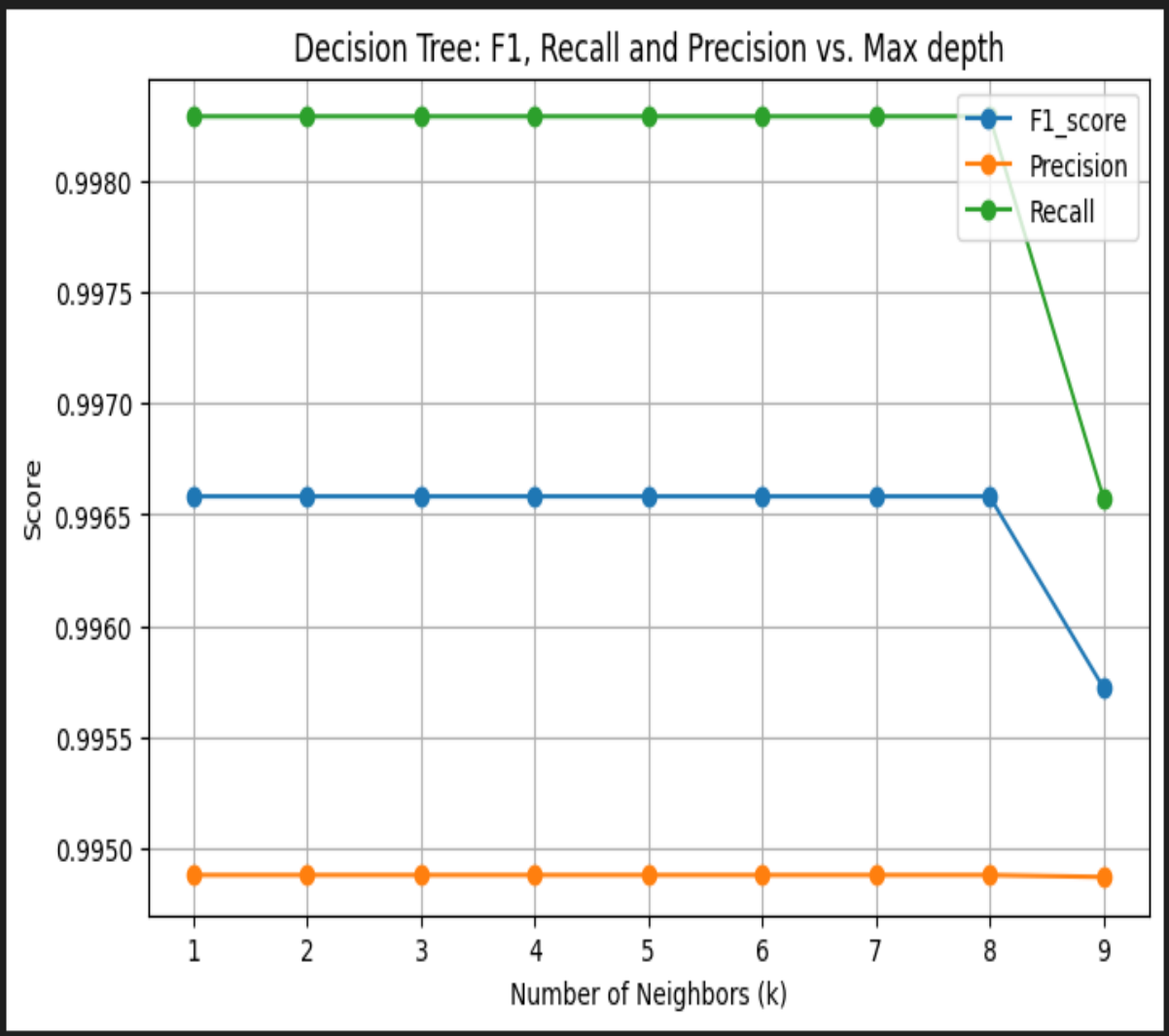
Hình 22: Kết quả

Xây dựng Model dựa trên thuật toán Decision Tree.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score, precision_score, recall_score

# Khởi tạo các danh sách để lưu giá trị
depth_values = []
f1_scores = []
precision_scores = []
recall_scores = []

for depth in range(1, 10):
    reg = DecisionTreeClassifier(max_depth=depth, random_state=0)
    reg.fit(X_train, y_train)
    y_predict = reg.predict(X_test)

    f1 = f1_score(y_test, y_predict)
    precision = precision_score(y_test, y_predict)
    recall = recall_score(y_test, y_predict)

    print(f"depth = {depth}, F1 = {f1}, Precision = {precision}, , Recall = {recall}")


    # Lưu giá trị của depth, f1 score và precision score vào danh sách tương ứng
    depth_values.append(depth)
    f1_scores.append(f1)
    precision_scores.append(precision)
    recall_scores.append(recall)

# Vẽ đồ thị
plt.figure(figsize=(8, 5))

plt.plot(depth_values, f1_scores, marker='o', label='F1_score')
plt.plot(depth_values, precision_scores, marker='o', label='Precision')
plt.plot(depth_values, recall_scores, marker = 'o', label = 'Recall')

plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Score')
plt.title('Decision Tree: F1, Recall and Precision vs. Max depth')
plt.legend()
plt.grid(True)
plt.show()
```

Hình 23: Model dựa trên thuật toán Decision Tree

```
depth = 1, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
depth = 2, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
depth = 3, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
depth = 4, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
depth = 5, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
depth = 6, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
depth = 7, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
depth = 8, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
depth = 9, F1 = 0.9957228400342172, Precision = 0.9948717948717949, , Recall = 0.9965753424657534
```



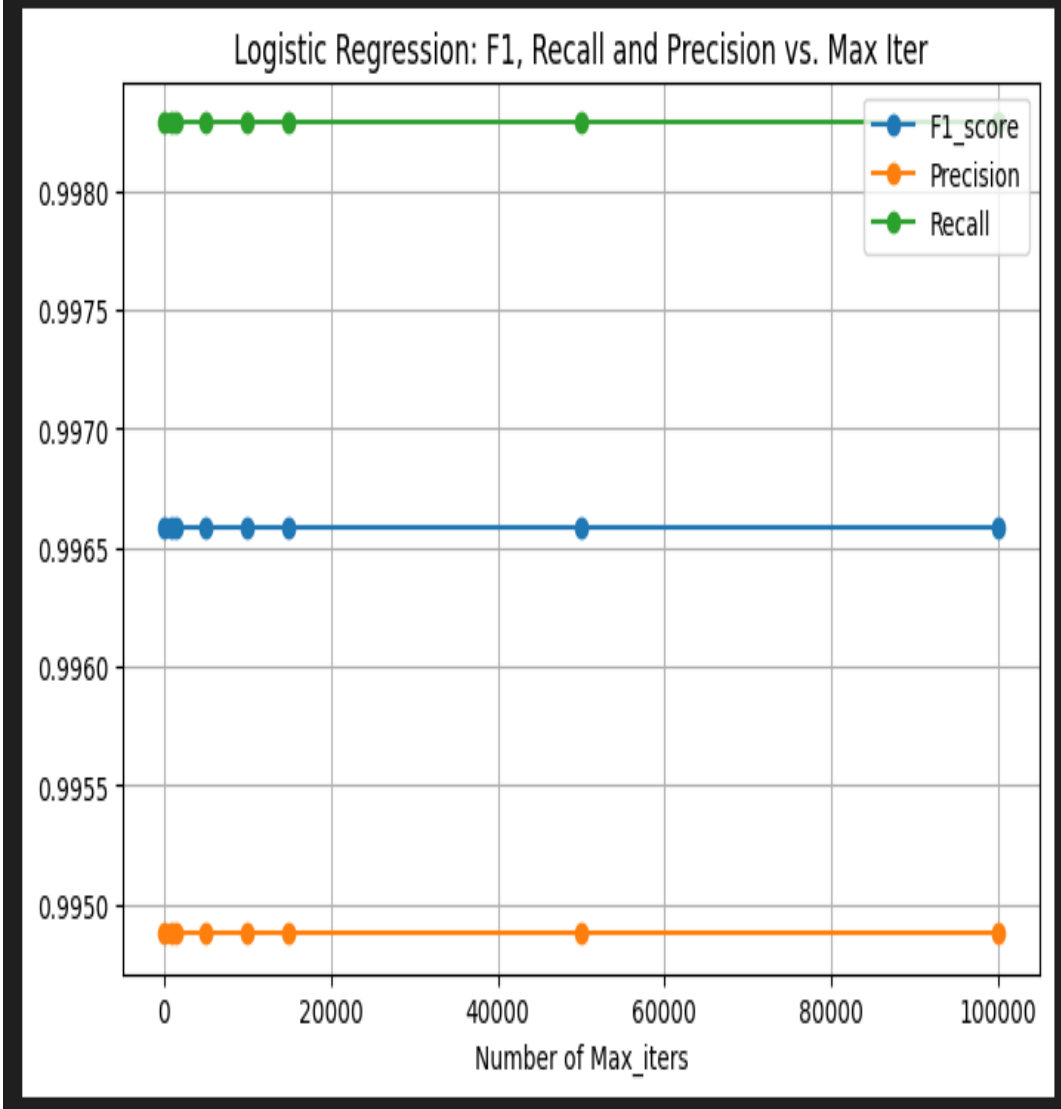
Hình 24: Kết quả

Xây dựng Model dựa trên thuật toán Logistic Regression.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, precision_score, recall_score

max_iters = [100, 1000, 1500, 5000, 10000, 15000, 50000, 100000]
f1_scores = []
precision_scores = []
recall_scores = []

for i in max_iters:
    model = LogisticRegression(max_iter=i)
    model.fit(X_train, y_train)
    y_predict = model.predict(X_test)


    f1 = f1_score(y_test, y_predict)
    precision = precision_score(y_test, y_predict)
    recall = recall_score(y_test, y_predict)

    print(f"Max_iter = {i}, F1 = {f1}, Precision = {precision}, , Recall = {recall}")
    f1_scores.append(f1)
    precision_scores.append(precision)
    recall_scores.append(recall)

# Vẽ đồ thị
plt.figure(figsize=(8, 5))

plt.plot(max_iters, f1_scores, marker='o', label='F1_score')
plt.plot(max_iters, precision_scores, marker='o', label='Precision')
plt.plot(max_iters, recall_scores, marker = 'o', label = 'Recall')

plt.xlabel('Number of Max_iters')
plt.title('Logistic Regression: F1, Recall and Precision vs. Max Iter')
plt.legend()
plt.grid(True)
plt.show()
```

Hình 25: Model dựa trên thuật toán Logistic Regression

```
Max_iter = 100, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
Max_iter = 1000, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
Max_iter = 1500, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
Max_iter = 5000, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
Max_iter = 10000, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
Max_iter = 15000, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
Max_iter = 50000, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
Max_iter = 100000, F1 = 0.9965811965811966, Precision = 0.9948805460750854, , Recall = 0.9982876712328768
```



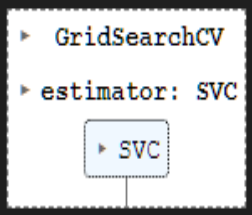
Hình 26: Kết quả

2.4 Xây dựng model dựa trên các thuật toán phức tạp hơn

Xây dựng Model với thuật toán SVM

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report

svm = SVC()

param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'degree': [2, 3, 4]
}

grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

✓ 13.5s




```
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Hyperparameters: ", best_params)
print("Best Cross-Validation Score: ", best_score)
```

✓ 0.0s

```
Best Hyperparameters:  {'C': 0.1, 'degree': 2, 'kernel': 'linear'}
Best Cross-Validation Score:  0.9985714285714286
```

Hình 27: Model dựa trên thuật toán SVM và kết quả

Xây dựng Model với thuật toán Recurrent Neural Network (RNN).

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout

# Khởi tạo mô hình Sequential
model = Sequential()

# Thêm một lớp LSTM với số unit là 50 và đầu vào có shape (số cột, số quan sát)
model.add(LSTM(units=50, input_shape=(X_train.shape[1], 1), return_sequences=True))
model.add(Dropout(0.2))  # Dropout layer để tránh overfitting

# Lớp LSTM thứ 2 với số unit 50
model.add(LSTM(units=50))
model.add(Dropout(0.2))

# Lớp fully connected
model.add(Dense(units=1, activation='sigmoid'))

# Compile mô hình
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Reshape dữ liệu cho phù hợp với input của mô hình LSTM (samples, time steps, features)
X_train_reshaped = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_reshaped = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Huấn luyện mô hình trên dữ liệu
model.fit(X_train_reshaped, y_train, epochs=10, batch_size=32, validation_data=(X_test_reshaped, y_test))

# Đánh giá mô hình trên dữ liệu test
loss, accuracy = model.evaluate(X_test_reshaped, y_test)
print(f'Loss: {loss}, Accuracy: {accuracy}')
```

Hình 28: Model dựa trên thuật toán RNN

```
Epoch 1/10
219/219 [==============================] - 6s 12ms/step - loss: 0.4879 - accuracy: 0.7943 - val_loss: 0.2730 - val_accuracy: 0.9107
Epoch 2/10
219/219 [==============================] - 2s 9ms/step - loss: 0.0872 - accuracy: 0.9793 - val_loss: 0.0152 - val_accuracy: 0.9987
Epoch 3/10
219/219 [==============================] - 2s 9ms/step - loss: 0.0159 - accuracy: 0.9986 - val_loss: 0.0099 - val_accuracy: 0.9987
Epoch 4/10
219/219 [==============================] - 2s 9ms/step - loss: 0.0121 - accuracy: 0.9986 - val_loss: 0.0092 - val_accuracy: 0.9987
Epoch 5/10
219/219 [==============================] - 2s 9ms/step - loss: 0.0117 - accuracy: 0.9986 - val_loss: 0.0098 - val_accuracy: 0.9987
Epoch 6/10
219/219 [==============================] - 2s 9ms/step - loss: 0.0105 - accuracy: 0.9986 - val_loss: 0.0092 - val_accuracy: 0.9987
Epoch 7/10
219/219 [==============================] - 2s 9ms/step - loss: 0.0104 - accuracy: 0.9986 - val_loss: 0.0095 - val_accuracy: 0.9987
Epoch 8/10
219/219 [==============================] - 2s 9ms/step - loss: 0.0107 - accuracy: 0.9986 - val_loss: 0.0093 - val_accuracy: 0.9987
Epoch 9/10
219/219 [==============================] - 2s 9ms/step - loss: 0.0097 - accuracy: 0.9986 - val_loss: 0.0093 - val_accuracy: 0.9987
Epoch 10/10
219/219 [==============================] - 2s 9ms/step - loss: 0.0106 - accuracy: 0.9986 - val_loss: 0.0091 - val_accuracy: 0.9987
94/94 [==============================] - 0s 3ms/step - loss: 0.0091 - accuracy: 0.9987
Loss: 0.009104852564632893, Accuracy: 0.9986666440963745
```

Hình 29: Kết quả

Xây dựng model dựa trên thuật toán Feed Forwark Neural Network.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Tạo một mô hình tuần tự
model = Sequential()

# Thêm các lớp vào mô hình
model.add(Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dropout(0.3))  # Thêm dropout để điều chuẩn
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))  # Lớp output với activation sigmoid cho bài toán phân loại nhị phân

# Compile mô hình
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Huấn luyện mô hình
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=1)

# Đánh giá mô hình trên dữ liệu kiểm thử
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Độ chính xác trên tập kiểm thử: {accuracy * 100:.2f}%')
```

```
Epoch 1/50
219/219 [==============================] - 1s 3ms/step - loss: 0.1427 - accuracy: 0.9504 - val_loss: 0.0090 - val_accuracy: 0.9987
Epoch 2/50
219/219 [==============================] - 1s 3ms/step - loss: 0.0114 - accuracy: 0.9984 - val_loss: 0.0097 - val_accuracy: 0.9987
Epoch 3/50
219/219 [==============================] - 1s 3ms/step - loss: 0.0104 - accuracy: 0.9986 - val_loss: 0.0100 - val_accuracy: 0.9987
Epoch 4/50
219/219 [==============================] - 1s 3ms/step - loss: 0.0119 - accuracy: 0.9986 - val_loss: 0.0091 - val_accuracy: 0.9987
Epoch 5/50
219/219 [==============================] - 1s 2ms/step - loss: 0.0117 - accuracy: 0.9984 - val_loss: 0.0094 - val_accuracy: 0.9987
Epoch 6/50
219/219 [==============================] - 1s 3ms/step - loss: 0.0126 - accuracy: 0.9986 - val_loss: 0.0091 - val_accuracy: 0.9987
Epoch 7/50
219/219 [==============================] - 1s 3ms/step - loss: 0.0107 - accuracy: 0.9986 - val_loss: 0.0096 - val_accuracy: 0.9987
Epoch 8/50
219/219 [==============================] - 1s 3ms/step - loss: 0.0104 - accuracy: 0.9986 - val_loss: 0.0093 - val_accuracy: 0.9987
Epoch 9/50
219/219 [==============================] - 1s 2ms/step - loss: 0.0102 - accuracy: 0.9986 - val_loss: 0.0096 - val_accuracy: 0.9987
Epoch 10/50
219/219 [==============================] - 1s 2ms/step - loss: 0.0095 - accuracy: 0.9986 - val_loss: 0.0099 - val_accuracy: 0.9987
Epoch 11/50
219/219 [==============================] - 1s 3ms/step - loss: 0.0107 - accuracy: 0.9986 - val_loss: 0.0095 - val_accuracy: 0.9987
Epoch 12/50
219/219 [==============================] - 1s 2ms/step - loss: 0.0110 - accuracy: 0.9986 - val_loss: 0.0097 - val_accuracy: 0.9987
Epoch 13/50
...
Epoch 50/50
219/219 [==============================] - 1s 3ms/step - loss: 0.0069 - accuracy: 0.9987 - val_loss: 0.0158 - val_accuracy: 0.9987
94/94 [==============================] - 0s 1ms/step - loss: 0.0158 - accuracy: 0.9987
Độ chính xác trên tập kiểm thử: 99.87%
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Hình 30: Xây dựng model trên thuật toán FFNN và kết quả

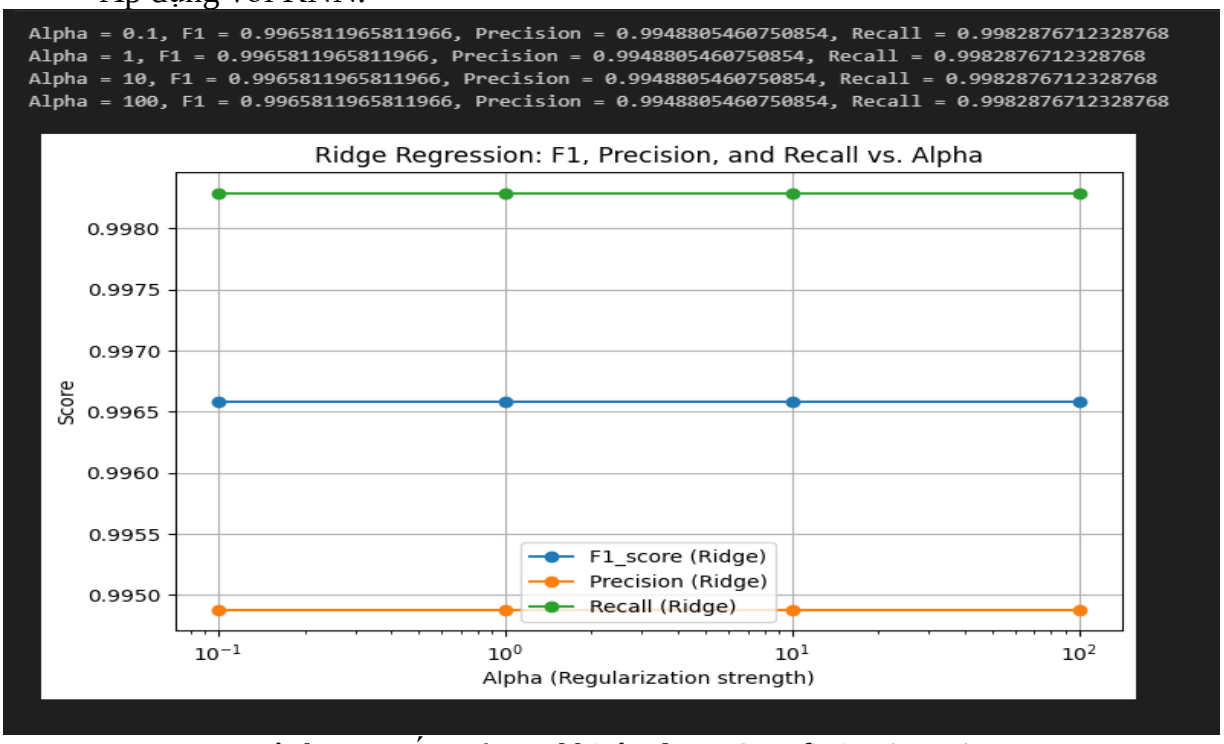
2.5 Xử lý Overfitting

Áp dụng mô hình hồi quy Ridge để xử lý Overfittng với các thuật toán KNN, Decisicion Tree và Logistic Regrssion.

```
from sklearn.linear_model import Ridge
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import make_pipeline



# Standardize features to have mean=0 and variance=1
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize lists to store values
ridge_f1_scores = []
ridge_precision_scores = []
ridge_recall_scores = []

alpha_values = [0.1, 1, 10, 100]  # Varying alpha values for regularization

for alpha in alpha_values:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train_scaled, y_train)
    y_predict_ridge = ridge.predict(X_test_scaled)
    # Assuming y_predict_ridge contains predicted values for regression

    # Evaluate the model
    f1_ridge = f1_score(y_test, y_predict_ridge.round())  # Adjust predictions if needed
    precision_ridge = precision_score(y_test, y_predict_ridge.round())
    recall_ridge = recall_score(y_test, y_predict_ridge.round())

    print(f"Alpha = {alpha}, F1 = {f1_ridge}, Precision = {precision_ridge}, Recall = {recall_ridge}")

    # Append scores to lists
    ridge_f1_scores.append(f1_ridge)
    ridge_precision_scores.append(precision_ridge)
    ridge_recall_scores.append(recall_ridge)

# Plotting Ridge regression scores
plt.figure(figsize=(8, 5))

plt.plot(alpha_values, ridge_f1_scores, marker='o', label='F1_score (Ridge)')
plt.plot(alpha_values, ridge_precision_scores, marker='o', label='Precision (Ridge)')
plt.plot(alpha_values, ridge_recall_scores, marker='o', label='Recall (Ridge)')

plt.xlabel('Alpha (Regularization strength)')
plt.ylabel('Score')
plt.title('Ridge Regression: F1, Precision, and Recall vs. Alpha')
plt.xscale('log')  # Using logarithmic scale for alpha values
plt.legend()
plt.grid(True)
plt.show()
```

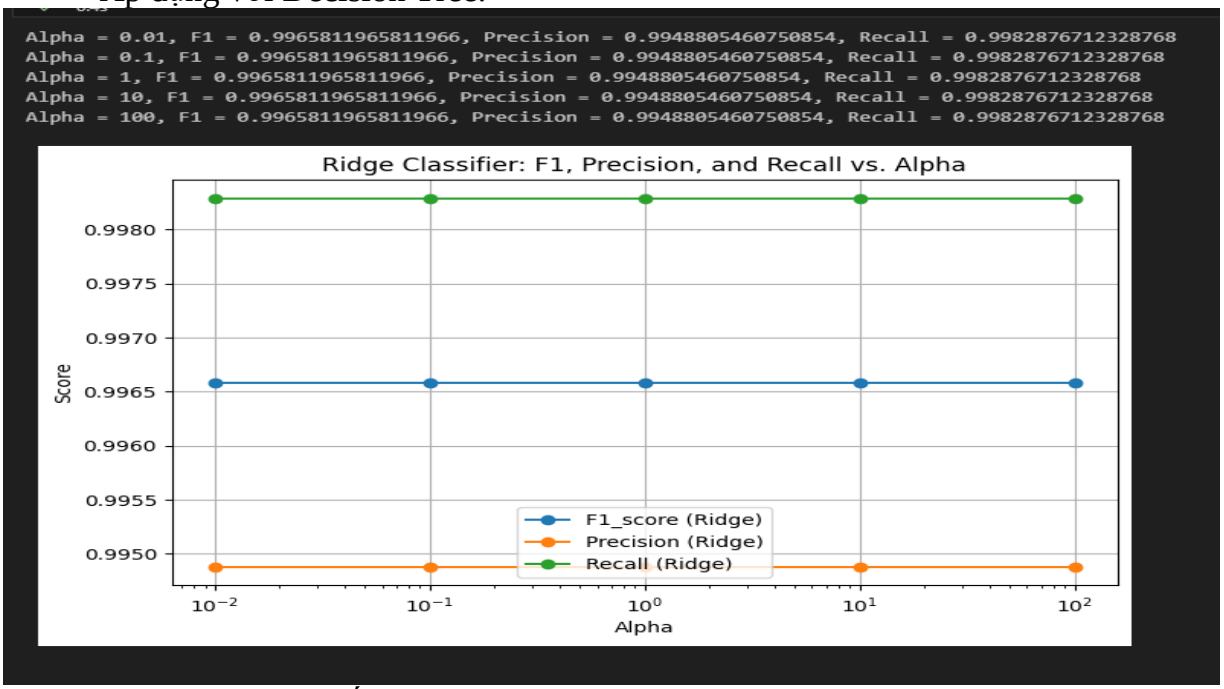
Hình 31: Mô hình hồi quy Ridge

Áp dụng với KNN.



Hình 32: Kết quả sau khi áp dụng Overfitting(KNN)

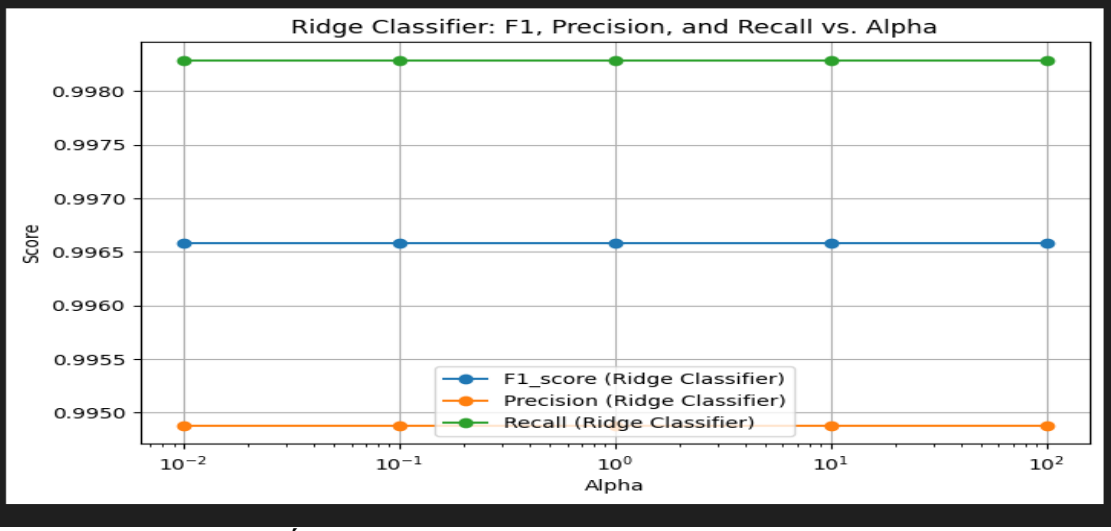
Áp dụng với Decision Tree.



Hình 33: Kết quả sau khi áp dụng Overfitting(Decision Tree)

Áp dụng với Logistic Regrssion.

```
Alpha = 0.01, F1 = 0.9965811965811966, Precision = 0.9948805460750854, Recall = 0.9982876712328768
Alpha = 0.1, F1 = 0.9965811965811966, Precision = 0.9948805460750854, Recall = 0.9982876712328768
Alpha = 1, F1 = 0.9965811965811966, Precision = 0.9948805460750854, Recall = 0.9982876712328768
Alpha = 10, F1 = 0.9965811965811966, Precision = 0.9948805460750854, Recall = 0.9982876712328768
Alpha = 100, F1 = 0.9965811965811966, Precision = 0.9948805460750854, Recall = 0.9982876712328768
```



Hình 34: Kết quả sau khi áp dụng Overfitting(Logistic Regression)

Với thuật toán SVM áp dụng biện pháp giảm giá trị C và sử dụng hàm kernel đơn giản hơn.

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report

svm = SVC()

param_grid = {
    'C': [0.01, 0.1, 0.5],  # Giảm giá trị C
    'kernel': ['linear'],  # Sử dụng kernel đơn giản
}

grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Hyperparameters: ", best_params)
print("Best Cross-Validation Score: ", best_score)
```

```
[77]  ✓ 0.3s

...  Best Hyperparameters:  {'C': 0.01, 'kernel': 'linear'}
     Best Cross-Validation Score:  0.9985714285714286
```

Hình 35: Kết quả sau khi áp dụng Overfitting(SVM)

Với thuật toán RNN giảm tỷ lệ Dropout để xử lý Overfitting.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout

model = Sequential()

model.add(LSTM(units=25, input_shape=(X_train.shape[1], 1), return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=25))
model.add(Dropout(0.2))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.fit(X_train_reshaped, y_train, epochs=5, batch_size=32, validation_data=(X_test_reshaped, y_test))

loss, accuracy = model.evaluate(X_test_reshaped, y_test)
print(f'Loss: {loss}, Accuracy: {accuracy}')
```

✓ 12.6s

```
Epoch 1/5
219/219 [==============================] - 5s 11ms/step - loss: 0.5129 - accuracy: 0.7911 - val_loss: 0.4255 - val_accuracy: 0.8053
Epoch 2/5
219/219 [==============================] - 2s 7ms/step - loss: 0.2114 - accuracy: 0.9143 - val_loss: 0.0406 - val_accuracy: 0.9987
Epoch 3/5
219/219 [==============================] - 2s 7ms/step - loss: 0.0342 - accuracy: 0.9981 - val_loss: 0.0141 - val_accuracy: 0.9987
Epoch 4/5
219/219 [==============================] - 2s 7ms/step - loss: 0.0174 - accuracy: 0.9986 - val_loss: 0.0105 - val_accuracy: 0.9987
Epoch 5/5
219/219 [==============================] - 2s 7ms/step - loss: 0.0131 - accuracy: 0.9986 - val_loss: 0.0094 - val_accuracy: 0.9987
94/94 [==============================] - 0s 3ms/step - loss: 0.0094 - accuracy: 0.9987
Loss: 0.009363985620439053, Accuracy: 0.9986666440963745
```

Hình 36: Kết quả sau khi áp dụng Overfiiting (RNN)

Với thuật toán FFNN tăng tỷ lệ dropout để loại bỏ một phần lớn neuron.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

model = Sequential()

model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dropout(0.5))  # Tỷ lệ Dropout tăng lên để loại bỏ một phần lớn neuron

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=30, batch_size=32, validation_data=(X_test, y_test), verbose=1)

loss, accuracy = model.evaluate(X_test, y_test)
print(f'Độ chính xác trên tập kiểm thử: {accuracy * 100:.2f}%')
```

```
Epoch 1/30
219/219 [==============================] - 1s 3ms/step - loss: 0.3093 - accuracy: 0.8843 - val_loss: 0.0209 - val_accuracy: 0.9987
Epoch 2/30
219/219 [==============================] - 1s 2ms/step - loss: 0.0322 - accuracy: 0.9956 - val_loss: 0.0097 - val_accuracy: 0.9987
Epoch 3/30
219/219 [==============================] - 1s 2ms/step - loss: 0.0216 - accuracy: 0.9973 - val_loss: 0.0096 - val_accuracy: 0.9987
Epoch 4/30
219/219 [==============================] - 1s 2ms/step - loss: 0.0205 - accuracy: 0.9976 - val_loss: 0.0103 - val_accuracy: 0.9987
Epoch 5/30
219/219 [==============================] - 1s 2ms/step - loss: 0.0180 - accuracy: 0.9986 - val_loss: 0.0098 - val_accuracy: 0.9987
Epoch 6/30
219/219 [==============================] - 1s 2ms/step - loss: 0.0194 - accuracy: 0.9981 - val_loss: 0.0096 - val_accuracy: 0.9987
Epoch 7/30
219/219 [==============================] - 1s 2ms/step - loss: 0.0146 - accuracy: 0.9986 - val_loss: 0.0099 - val_accuracy: 0.9987
Epoch 8/30
219/219 [==============================] - 1s 2ms/step - loss: 0.0156 - accuracy: 0.9984 - val_loss: 0.0101 - val_accuracy: 0.9987
Epoch 9/30
219/219 [==============================] - 1s 2ms/step - loss: 0.0174 - accuracy: 0.9986 - val_loss: 0.0103 - val_accuracy: 0.9987
Epoch 10/30
219/219 [==============================] - 1s 2ms/step - loss: 0.0138 - accuracy: 0.9986 - val_loss: 0.0109 - val_accuracy: 0.9987
Epoch 11/30
219/219 [==============================] - 1s 2ms/step - loss: 0.0127 - accuracy: 0.9986 - val_loss: 0.0106 - val_accuracy: 0.9987
Epoch 12/30
219/219 [==============================] - 1s 2ms/step - loss: 0.0142 - accuracy: 0.9986 - val_loss: 0.0106 - val_accuracy: 0.9987
Epoch 13/30
...
Epoch 30/30
219/219 [==============================] - 1s 2ms/step - loss: 0.0127 - accuracy: 0.9986 - val_loss: 0.0106 - val_accuracy: 0.9987
94/94 [==============================] - 0s 1ms/step - loss: 0.0106 - accuracy: 0.9987
Độ chính xác trên tập kiểm thử: 99.87%
```

Hình 37: Kết quả sau khi áp dụng Overfitting (FFNN)

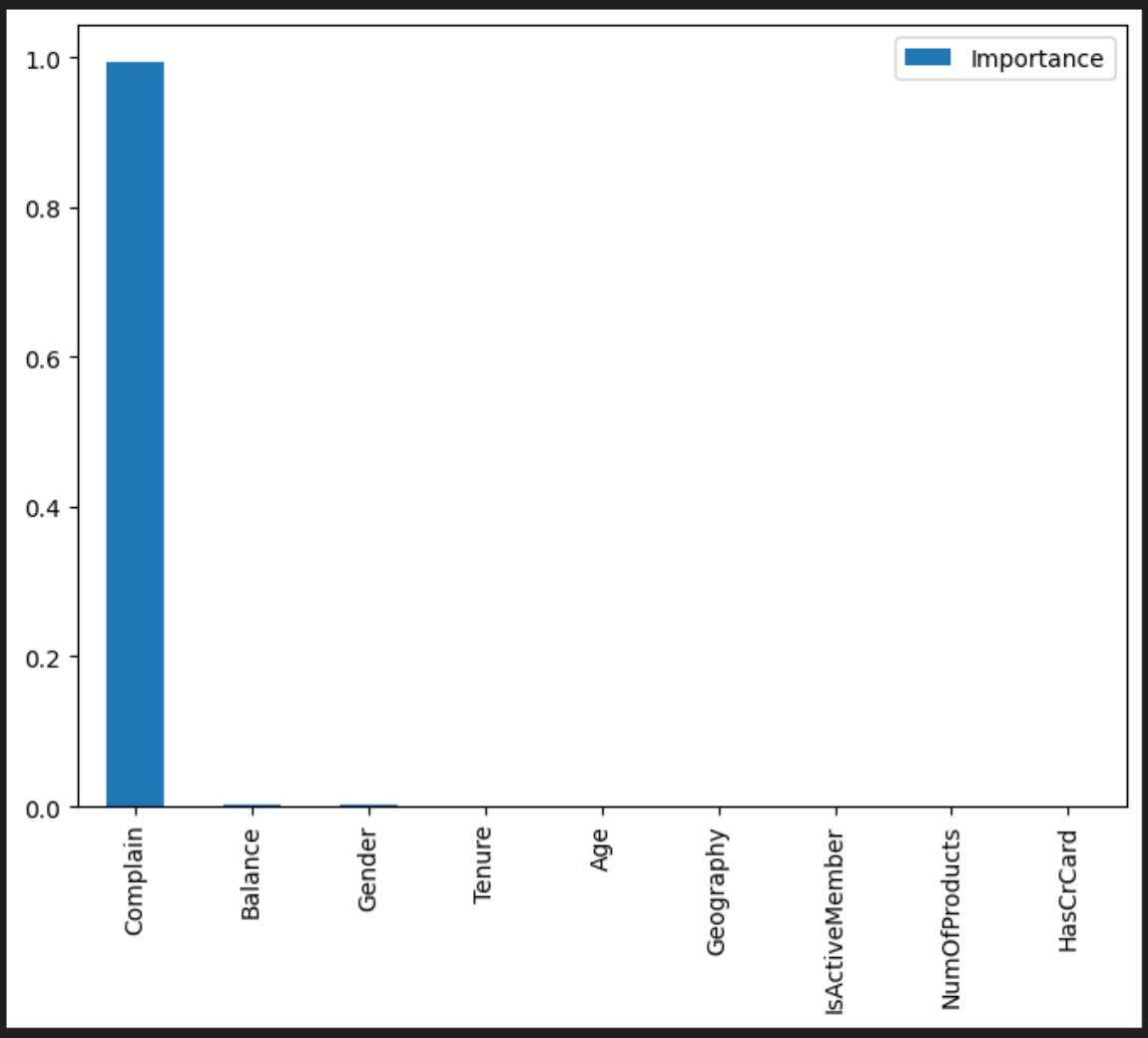
2.6 Đánh giá

Thông qua các kết lần đầu chạy thuật toán và sau khi đã áp dụng các biện pháp nhằm xử lý hiện tượng Overfitting, chúng ta có thể thấy rằng model hiện tại đang hoạt động không hiệu quả, với tỷ lệ chính xác quá cao lên đến 99%. Có thể thấy độ chính xác cao như vậy là không bình thường nên biện pháp được đưa ra là tiến hành đánh giá các feature tham gia xây dựng model bằng hàm “feature_importances_”.

```
feat_importance = reg.feature_importances_
print("feat importance = " + str(feat_importance))
feat_importances = pd.DataFrame(reg.feature_importances_, index=X.columns, columns=["Importance"])
print(feat_importances)
feat_importances.sort_values(by='Importance', ascending=False, inplace=True)
feat_importances.plot(kind='bar', figsize=(8, 6))

✓ 0.1s
```

Hình 38: Đánh giá lại feature

```
 9.93535608e-01]
                Importance
Geography         0.000256
Gender            0.001265
Age               0.000667
Tenure            0.001145
Balance           0.002937
NumOfProducts     0.000000
HasCrCard         0.000000
IsActiveMember    0.000194
Complain          0.993536

<Axes: >
```



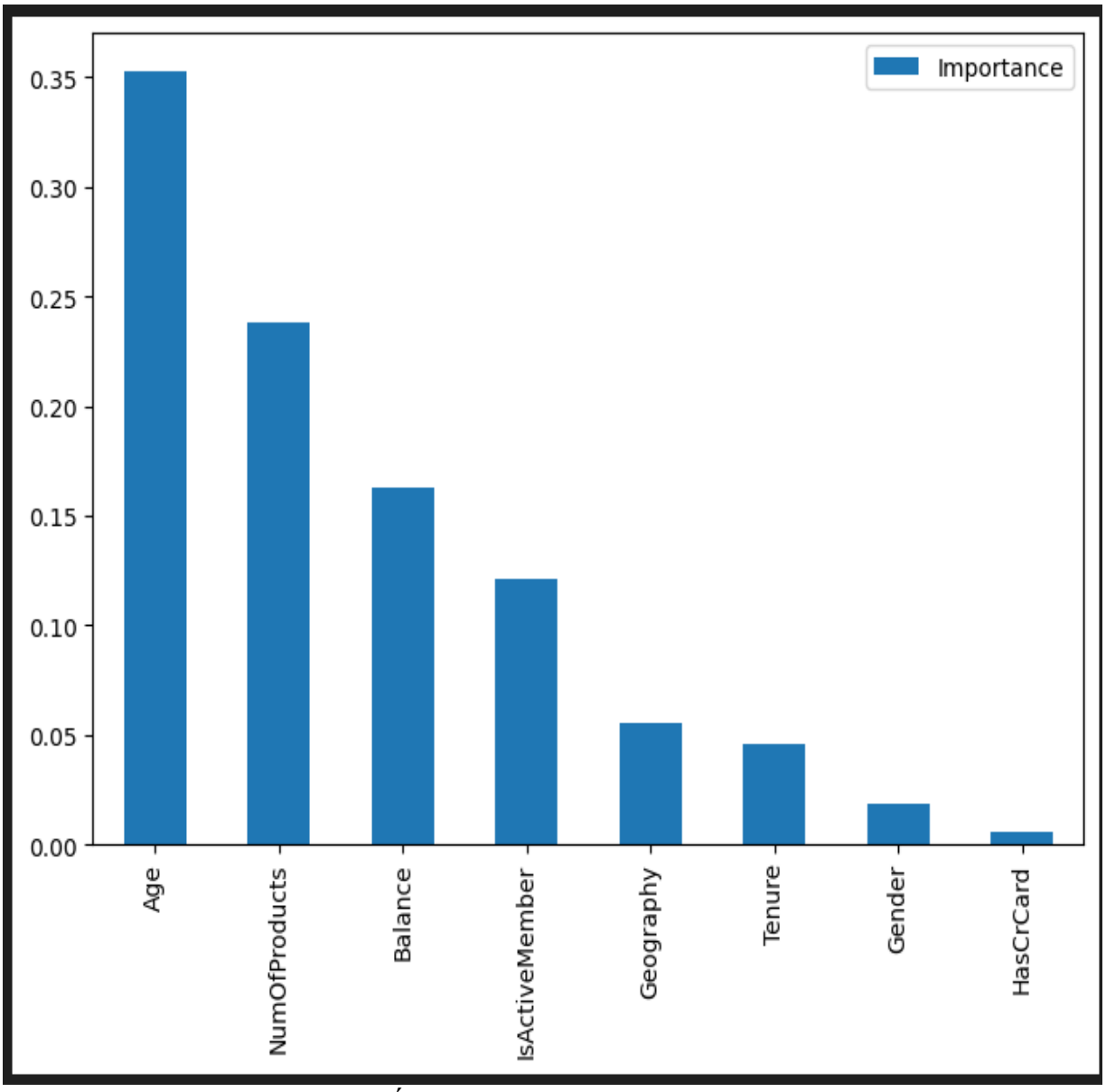
Hình 39: Kết sau khi đánh giá lại Feature

Có thể thấy feature “complain” chiếm một trọng số quá lớn cho nên nó đã ảnh hưởng rất lớn đến việc xây dựng model, khiến cho kết quả của các thuật toán đều nằm ở mức 99%. Hướng giải quyết được đưa ra chính là loại bỏ feature này.

```
df.drop(columns=["Card Type", "Satisfaction Score", "Point Earned", "CreditScore", "EstimatedSalary"], inplace=True)
df.drop(columns=["Complain" ], inplace=True)
df.head()
```

✓ 0.0s

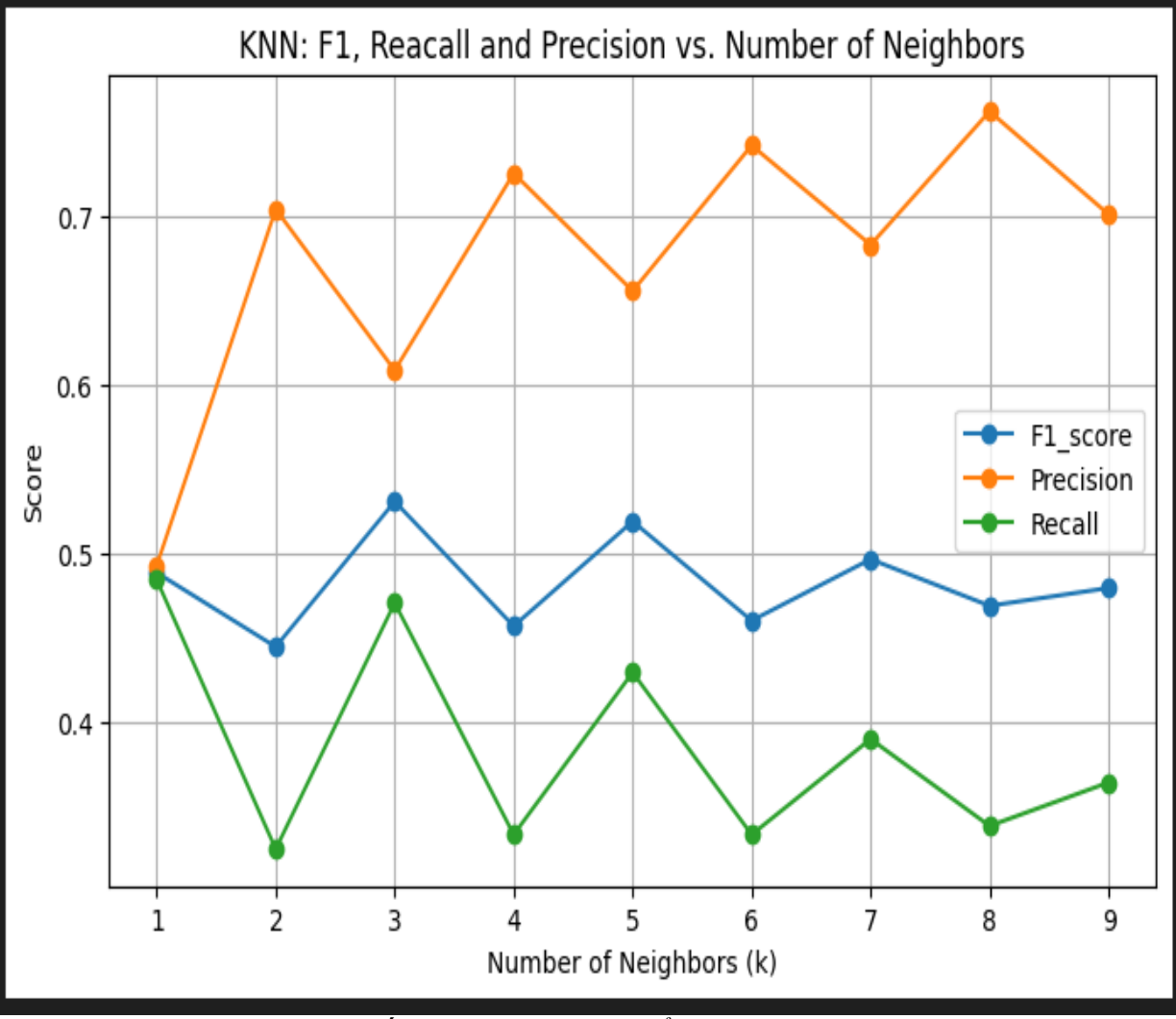
Hình 40: Loại bỏ Feature



Hình 41: Trọng số của các Feature sau khi loại bỏ Complain

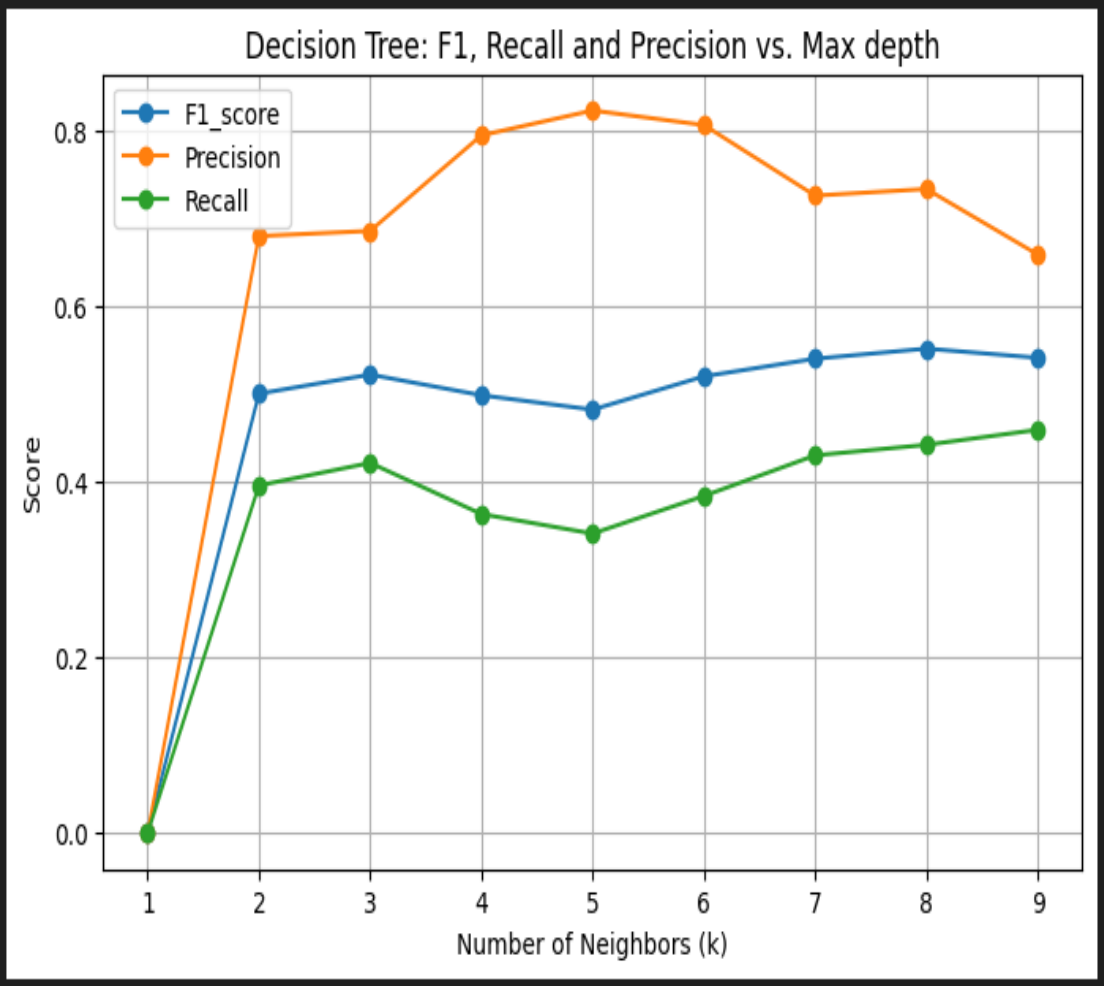
Kết quả của các thuật toán sau khi loại bỏ "Complain"

```
K = 1, F1 = 0.48877374784110533, Precision = 0.4930313588850174, , Recall = 0.4845890410958904
K = 2, F1 = 0.4449648711943794, Precision = 0.7037037037037037, , Recall = 0.3253424657534247
K = 3, F1 = 0.5308880308880309, Precision = 0.6084070796460177, , Recall = 0.4708904109589041
K = 4, F1 = 0.45720984759671746, Precision = 0.724907063197026, , Recall = 0.3339041095890411
K = 5, F1 = 0.5191313340227508, Precision = 0.6553524804177546, , Recall = 0.4297945205479452
K = 6, F1 = 0.4604486422668241, Precision = 0.7414448669201521, , Recall = 0.3339041095890411
K = 7, F1 = 0.49673202614379086, Precision = 0.6826347305389222, , Recall = 0.3904109589041096
K = 8, F1 = 0.46919431279620855, Precision = 0.7615384615384615, , Recall = 0.339041095890411
K = 9, F1 = 0.47972972972972977, Precision = 0.7006578947368421, , Recall = 0.3647260273972603
```



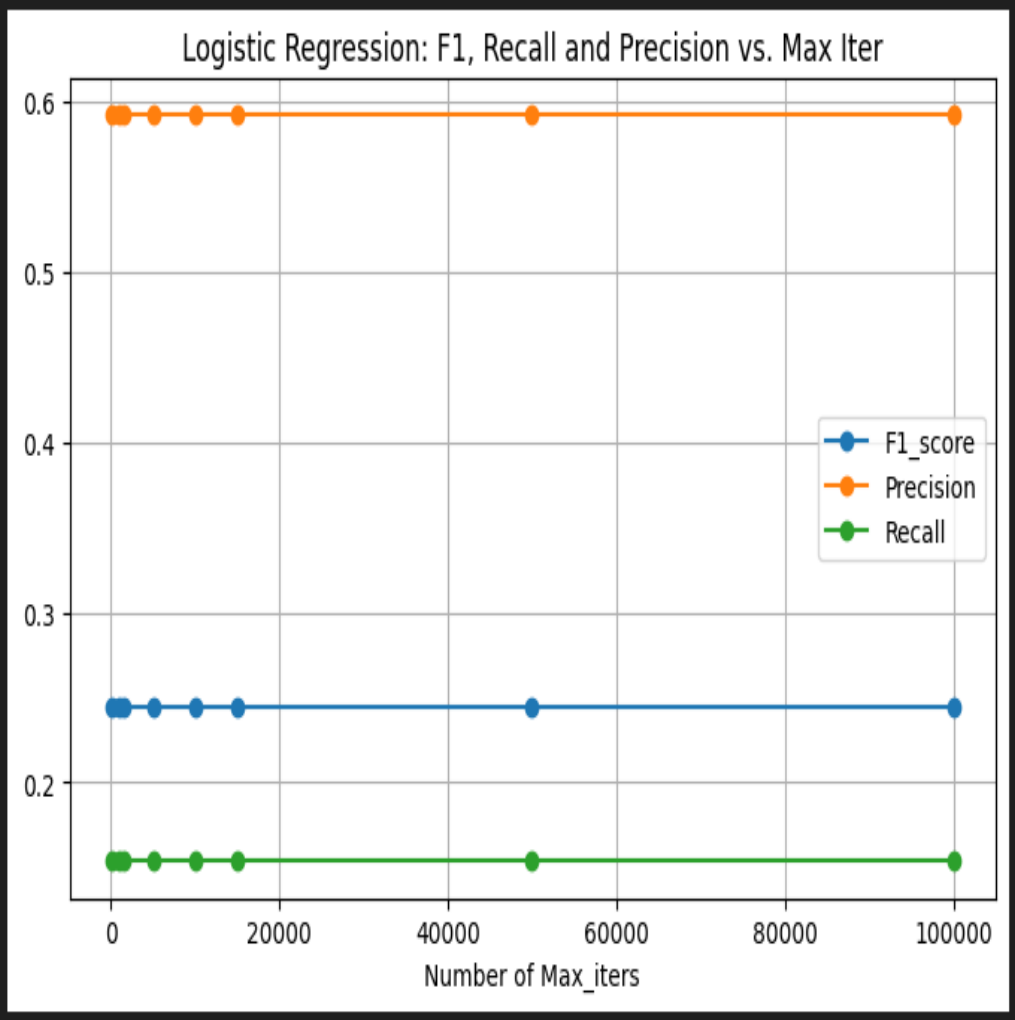
Hình 42: Kết quả sau khi thay đổi Feature(KNN)

```
  _warn_prf(average, modifier, msg_start, len(result))
depth = 1, F1 = 0.0, Precision = 0.0, , Recall = 0.0
depth = 2, F1 = 0.5, Precision = 0.6794117647058824, , Recall = 0.3955479452054795
depth = 3, F1 = 0.5217391304347827, Precision = 0.6852367688022284, , Recall = 0.4212328767123288
depth = 4, F1 = 0.4982373678025852, Precision = 0.7940074906367042, , Recall = 0.363013698630137
depth = 5, F1 = 0.48184019370460046, Precision = 0.8223140495867769, , Recall = 0.3407534246575342
depth = 6, F1 = 0.519721577726218, Precision = 0.8057553956834532, , Recall = 0.3835616438356164
depth = 7, F1 = 0.5397849462365591, Precision = 0.7254335260115607, , Recall = 0.4297945205479452
depth = 8, F1 = 0.5512820512820513, Precision = 0.7329545454545454, , Recall = 0.4417808219178082
depth = 9, F1 = 0.5408678102926338, Precision = 0.6584766584766585, , Recall = 0.4589041095890411
```



Hình 43: Kết quả sau khi thay đổi Feature(Decision Tree)

```
Max_iter = 100, F1 = 0.24456521739130432, Precision = 0.5921052631578947, , Recall = 0.1541095890410959
Max_iter = 1000, F1 = 0.24456521739130432, Precision = 0.5921052631578947, , Recall = 0.1541095890410959
Max_iter = 1500, F1 = 0.24456521739130432, Precision = 0.5921052631578947, , Recall = 0.1541095890410959
Max_iter = 5000, F1 = 0.24456521739130432, Precision = 0.5921052631578947, , Recall = 0.1541095890410959
Max_iter = 10000, F1 = 0.24456521739130432, Precision = 0.5921052631578947, , Recall = 0.1541095890410959
Max_iter = 15000, F1 = 0.24456521739130432, Precision = 0.5921052631578947, , Recall = 0.1541095890410959
Max_iter = 50000, F1 = 0.24456521739130432, Precision = 0.5921052631578947, , Recall = 0.1541095890410959
Max_iter = 100000, F1 = 0.24456521739130432, Precision = 0.5921052631578947, , Recall = 0.1541095890410959
```



Hình 44: Kết quả sau khi thay đổi Feature(Logistic Regression)

```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report

svm = SVC()

param_grid = {
    'C': [0.01, 0.1, 0.5],  # Giảm giá trị C
    'kernel': ['linear'],  # Sử dụng kernel đơn giản
}

grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Hyperparameters: ", best_params)
print("Best Cross-Validation Score: ", best_score)
```

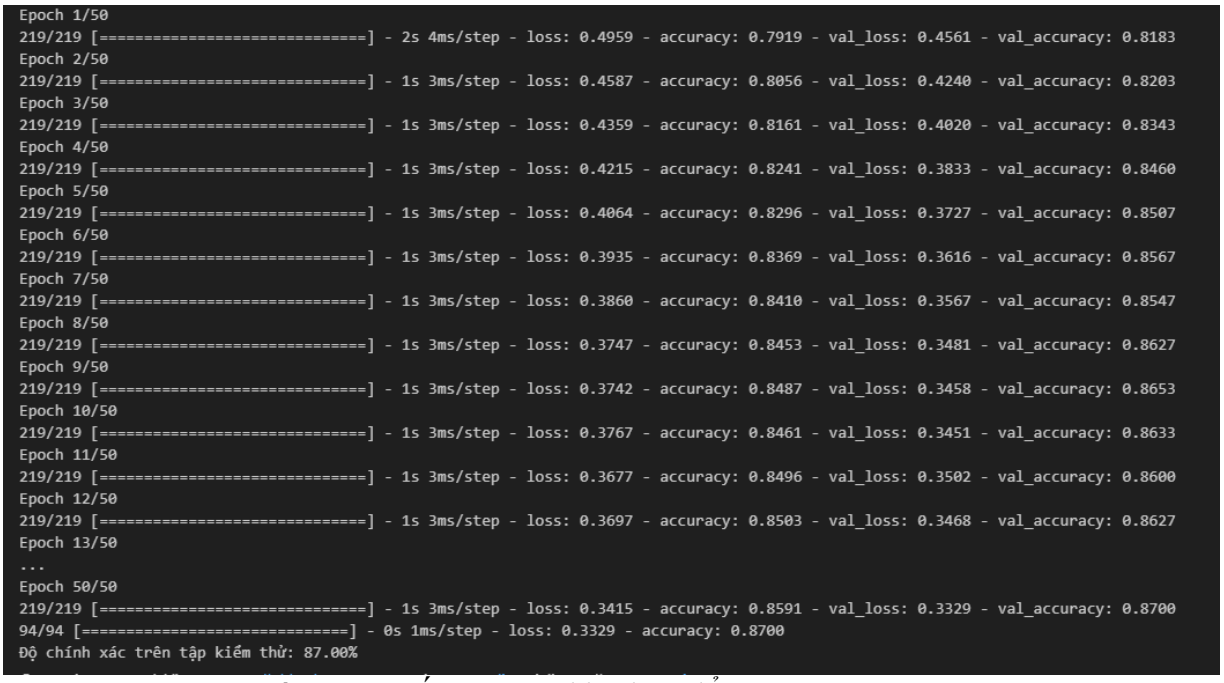
[170] ✓ 4.6s

```
Best Hyperparameters:  {'C': 0.01, 'kernel': 'linear'}
Best Cross-Validation Score:  0.7922857142857143
```

Hình 45 : Kết quả sau khi thay đổi Feature(SVM)

```
Epoch 1/10
219/219 [==============================] - 6s 14ms/step - loss: 0.5235 - accuracy: 0.7916 - val_loss: 0.4934 - val_accuracy: 0.8053
Epoch 2/10
219/219 [==============================] - 2s 10ms/step - loss: 0.5108 - accuracy: 0.7923 - val_loss: 0.4864 - val_accuracy: 0.8053
Epoch 3/10
219/219 [==============================] - 2s 9ms/step - loss: 0.5028 - accuracy: 0.7923 - val_loss: 0.4921 - val_accuracy: 0.8053
Epoch 4/10
219/219 [==============================] - 2s 9ms/step - loss: 0.4973 - accuracy: 0.7923 - val_loss: 0.4826 - val_accuracy: 0.8053
Epoch 5/10
219/219 [==============================] - 2s 9ms/step - loss: 0.4921 - accuracy: 0.7924 - val_loss: 0.4699 - val_accuracy: 0.8053
Epoch 6/10
219/219 [==============================] - 2s 9ms/step - loss: 0.4890 - accuracy: 0.7926 - val_loss: 0.4655 - val_accuracy: 0.8053
Epoch 7/10
219/219 [==============================] - 2s 10ms/step - loss: 0.4840 - accuracy: 0.7941 - val_loss: 0.4581 - val_accuracy: 0.8080
Epoch 8/10
219/219 [==============================] - 2s 10ms/step - loss: 0.4789 - accuracy: 0.7944 - val_loss: 0.4520 - val_accuracy: 0.8140
Epoch 9/10
219/219 [==============================] - 2s 10ms/step - loss: 0.4687 - accuracy: 0.7969 - val_loss: 0.4504 - val_accuracy: 0.8160
Epoch 10/10
219/219 [==============================] - 2s 9ms/step - loss: 0.4613 - accuracy: 0.8004 - val_loss: 0.4335 - val_accuracy: 0.8167
94/94 [==============================] - 0s 3ms/step - loss: 0.4335 - accuracy: 0.8167
Loss: 0.4334520697593689, Accuracy: 0.8166666626930237
```

Hình 46: Kết quả sau khi thay đổi Feature (RNN)



Hình 47: Kết quả sau khi thay đổi Feature (FFNN)

Kết luận: Có thể thấy việc mất cân bằng giữa các feature ảnh hưởng rất lớn đến độ chính xác của thuật toán, trong trường hợp này việc loại bỏ feature "complain" đã giúp model có thể đưa ra các kết quả bằng việc tổng hợp kết quả từ nhiều feature, nhờ đó trong thực tế độ chính xác sẽ tăng lên.

Trước khi áp dụng việc xác định độ quan trọng của các trọng số, nhóm đã áp dụng các phương pháp khác như tăng giảm độ lớn của dữ liệu, tăng giảm độ phân chia dữ liệu ở dữ liệu train và test, áp dụng các biện pháp xử lý Overfitting nhưng độ chính xác vẫn không tăng. Vấn đề chính ở đây là việc không ý thức được có những feature có thể có ảnh hưởng rất lớn đến model.

TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] T. T. Trực, “Optimizer - Hiểu sau về các thuật toán tối ưu,” 17 10 2020. [Trực tuyến]. Available: https://shorturl.at/dehyN. [Đã truy cập 22 12 2023].

[2] V. H. Tiệp, "Gradient Descent," 12 1 2017. [Online]. Available: https://shorturl.at/bLU15. [Accessed 20 12 2023].

Tiếng Anh

[3] A. Smith, "A complete Introduction to Continual Learning," 05 04 2023. [Online]. Available: https://shorturl.at/pvMN7. [Accessed 20 12 2023].

PHỤ LỤC

Phần này bao gồm những nội dung cần thiết nhằm minh họa hoặc hỗ trợ cho nội dung luận văn như số liệu, biểu mẫu, tranh ảnh. . . . nếu sử dụng những câu trả lời cho một *bảng câu hỏi thì bảng câu hỏi mẫu này phải được đưa vào phần Phụ lục ở dạng nguyên bản* đã dùng để điều tra, thăm dò ý kiến; không được tóm tắt hoặc sửa đổi. Các tính toán mẫu trình bày tóm tắt trong các biểu mẫu cũng cần nêu trong Phụ lục của luận văn. Phụ lục không được dày hơn phần chính của luận văn