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

DỰ ÁN CUỐI KỲ MÔN NHẬP MÔN HỌC MÁY

FINAL PROJECT 2023

Người hướng dẫn: **Thầy Lê Anh Cường**

Người thực hiện: **Lê Đình Khôi – 520H0374**

Vũ Lê Tấn Phát - 520H0120

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

DỰ ÁN CUỐI KỲ MÔN NHẬP MÔN HỌC MÁY

FINAL PROJECT 2023

Người hướng dẫn: **Thầy Lê Anh Cường**

Người thực hiện: **Lê Đình Khôi – 520H0374**

Vũ Lê Tấn Phát - 520H0120

Khoá**:** **24**

THÀNH PHỐ HỒ CHÍ MINH

NĂM 2023

LỜI CẢM ƠN

Chúng em xin chân thành gửi lời cảm ơn này đến thầy Lê Anh Cường giảng viên phụ trách giảng dạy bộ môn Nhập môn Học máy. Nhờ có sự tận tình giảng dạy, truyền đạt kiến thức của quý thầy mà chúng em mới đủ kiến thức để hoàn thành đồ án cuối kỳ này.

Bên cạnh đó, chúng em cũng xin gửi lời cảm ơn đến Khoa Công Nghệ Thông Tin, trường Đại học Tôn Đức Thắng vì đã tạo điều kiện cho chúng em học tập, nghiên cứu trong suốt quá trình học tập môn học này nói riêng và cả quá trình học tại môi trường Đại học nói chung. Chúng em xin cảm ơn.

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

Chúng tôi xin cam đoan đây là sản phẩm đồ án của chúng tôi. 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 tháng năm

Tác giả

Lê Đình Khôi

Vũ Lê Tấn Phát

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm

(ký và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm

(ký và ghi họ tên)

TÓM TẮT

Bài báo cáo cuối kỳ học kì I năm học 2023 - 2024 của môn Nhập môn Học máy nhằm báo cáo lại những vấn đề trong quá trình học và thực hành trên lớp. Bài báo cáo nghiên cứu về xác định và vận dụng các mô hình học máy để giải quyết nhiều bài toán khác nhau, chúng em đã vận dụng các kiến thức đã học trên lớp để giải những yêu cầu đưa ra. Trong quá trình làm bài báo cáo, chúng em cũng đã có những khó khăn trong việc tìm hiểu đề, nhưng chúng em vẫn đã làm được. Kết quả là chúng em đã hoàn thành bài báo cáo tốt nhất đối với chúng em.

MỤC LỤC

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

I. BÀI LÀM

1. Bài làm riêng từng người

1.1. 520H0374 - Lê Đình Khôi

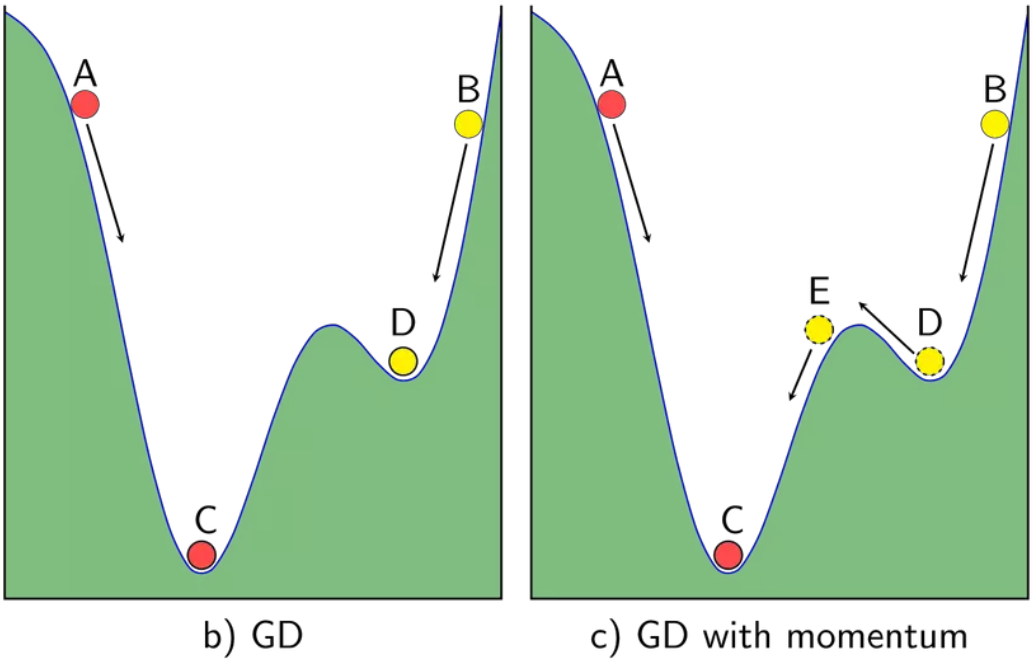
1.1.1. Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy

- Các phương pháp Optimizer trong huấn luyện học máy được trình bày trong báo cáo này bao gồm:
    - Adagrad (Adaptive Gradient Algorithm)
    - RMSprop (Root Mean Square Propagation)
    - Momentum
    - Adam

- **Adagrad (Adaptive Gradient Algorithm)** là một thuật toán tối ưu hóa được sử dụng trong học máy, đặc biệt là trong huấn luyện mạng nơ-ron sâu (deep neural networks). Nó là một biến thể của thuật toán Gradient Descent, nhằm khắc phục một số nhược điểm của Gradient Descent và cải thiện hiệu quả huấn luyện.
- Cách thức hoạt động của Adagrad: giống như Gradient Descent, Adagrad cập nhật các tham số của mô hình theo hướng giảm dần của hàm mục tiêu. Tuy nhiên, Adagrad điều chỉnh tốc độ học (learning rate) cho từng tham số của mô hình, dựa trên độ lớn của gradient của tham số đó.
- Cụ thể, Adagrad tính tổng bình phương của gradient của từng tham số trong quá trình huấn luyện. Tốc độ học của tham số sẽ được điều chỉnh theo công thức sau:
    - **learning_rate = learning_rate_base / sqrt(sum_squared_gradients)**
- Trong đó:
    - learning_rate_base là tốc độ học ban đầu.
    - sum_squared_gradients là tổng bình phương của gradient của tham số.

- **RMSprop (Root Mean Square Propagation)** là một thuật toán tối ưu hóa được sử dụng trong học máy, đặc biệt là trong huấn luyện mạng nơ-ron sâu (deep neural networks). Về cơ bản, nó là một biến thể của thuật toán Adagrad, nhằm khắc phục một số nhược điểm của Adagrad và cải thiện hiệu quả huấn luyện.
- Cách thức hoạt động của RMSprop: giống như Adagrad, RMSprop điều chỉnh tốc độ học (learning rate) cho từng tham số của mô hình, nhưng theo một cách khác:
    - Thay vì chỉ tính mức độ dốc trung bình bình phương từng tham số như Adagrad, RMSprop tính cả gốc bình phương trung bình (root mean square) của gradient.
    - Gốc bình phương trung bình giúp "giảm dần" ảnh hưởng của các gradient lớn trong quá khứ, do đó các tham số có gradient lớn ban đầu sẽ không bị giảm tốc độ học quá nhanh.
    - Điều này cải thiện tính ổn định của quá trình huấn luyện, nhất là đối với những mô hình có nhiều tham số và dữ liệu nhiễu.
- Công thức của RMSprop:
    - **E_t = (1 - beta^2) * E_{t-1} + beta^2 * g_t^2**
    - **x_t = x_{t-1} - learning_rate * sqrt(E_t + epsilon) * g_t**
- Trong đó:
    - E_t là trung bình động của bình phương gradient tại thời điểm t.
    - E_{t-1} là trung bình động của bình phương gradient tại thời điểm t-1.
    - g_t là gradient tại thời điểm t.
    - x_t là giá trị của tham số tại thời điểm t.
    - x_{t-1} là giá trị của tham số tại thời điểm t-1.
    - learning_rate là tốc độ học.
    - beta là một hyperparameter xác định tốc độ suy giảm của trung bình động.
    - epsilon là một giá trị nhỏ để tránh chia cho 0.

- Để giải thích được **Gradient with Momentum** thì trước tiên ta nên nhìn dưới góc độ vật lí: Ở hình 3b), ta có thể thấy, khi thả 2 viên bi ở 2 điểm A và B khác nhau, Viên bi ở điểm A sẽ dừng tại điểm C, trong khi viên bị xuất phát từ điểm B sẽ dừng tại D. Tuy nhiên, ta không muốn viên bi tại B dừng ở điểm D (local minimum) mà sẽ tiếp tục tiến tới và dừng tại C (global minimum). Để thực hiện được điều này, ở hình 3c), ta phải cho viên bi tại B 1 vận tốc đủ lớn để viên bi có thể vượt qua điểm D đến E và dừng ở C. Dựa trên hiện tượng này, một thuật toán được ra đời nhằm khắc phục việc nghiệm của GD rơi vào một điểm local minimum không mong muốn. Thuật toán đó có tên là **Momentum** (tức theo đà trong tiếng Việt).



Hình 1. So sánh GD khi có và không có momentum

- Cách thức hoạt động của Momentum: trong mỗi bước cập nhật, momentum tính toán một giá trị mới dựa trên gradient của bước hiện tại và giá trị momentum từ bước trước đó. Công thức cập nhật momentum thường được mô tả như sau:
  - **v_t = beta * v_{t-1} + g_t**
  - **x_t = x_{t-1} - learning_rate * v_t**

- Trong đó:
    - v_t là momentum tại thời điểm t.
    - v_{t-1} là momentum tại thời điểm t-1.
    - g_t là gradient tại thời điểm t.
    - x_t là giá trị của tham số tại thời điểm t.
    - x_{t-1} là giá trị của tham số tại thời điểm t-1.
    - learning_rate là tốc độ học.
    - beta là một hyperparameter xác định mức độ ảnh hưởng của momentum từ các bước trước đó.

- **Adam (Adaptive Moment Estimation)** là một thuật toán tối ưu hóa phổ biến trong học máy, đặc biệt là trong huấn luyện mạng nơ-ron sâu (deep neural networks). Về cơ bản, nó kết hợp ưu điểm của các thuật toán Gradient Descent, Momentum và RMSprop, mang lại hiệu quả và tính ổn định cao trong quá trình huấn luyện.
- Cách thức hoạt động của Adam: Adam sử dụng hai thành phần chính để cập nhật các tham số của mô hình:
    - Estimated first moment (m): Tương tự như Momentum, Adam ước tính momentum của gradient, cung cấp thông tin về hướng di chuyển chung của các tham số.
    - Estimated squared second moment (v): Tương tự như RMSprop, Adam ước tính độ biến thiên trung bình của gradient, nhằm điều chỉnh tốc độ học cho từng tham số.
- Sau mỗi bước di chuyển, Adam cập nhật cả hai thành phần này và sử dụng chúng để tính toán tốc độ học cho từng tham số theo công thức sau:
    - **m_t = beta1 * m_{t-1} + (1 - beta1) * g_t**
    - **v_t = beta2 * v_{t-1} + (1 - beta2) * g_t^2**
    - **x_t = x_{t-1} - learning_rate * m_t / sqrt(v_t + epsilon)**
- Trong đó:
    - m_t là moment (momentum) tại thời điểm t.
    - v_t là squared moment (moment bình phương) tại thời điểm t.

- g_t là gradient tại thời điểm t.
- x_t là giá trị của tham số tại thời điểm t.
- x_{t-1} là giá trị của tham số tại thời điểm t-1.
- learning_rate là tốc độ học.
- beta1 và beta2 là hyperparameter xác định tốc độ suy giảm của moment và squared moment.
- epsilon là một giá trị nhỏ để tránh chia cho 0.

- So sánh giữa Adagrad, RMSprop, Momentum và Adam:

Phương pháp	**Ưu điểm**	**Nhược điểm**
Adagrad	- Tự động điều chỉnh tỉ lệ học cho từng tham số dựa trên lịch sử của gradient. - Hiệu quả đối với dữ liệu có độ dốc biến động lớn.	- Có thể dẫn đến việc giảm tỉ lệ học quá nhanh và dừng lại sớm nếu tổng bình phương của gradient trở nên quá lớn.
RMSprop	- Giảm vấn đề giảm tỉ lệ học quá nhanh của Adagrad bằng cách sử dụng trung bình động của bình phương gradient. - Hiệu quả đối với các bài toán deep learning và dữ liệu có độ dốc biến động.	- Có thể vẫn có vấn đề với tỉ lệ học quá nhỏ sau một số lượng lớn các bước huấn luyện.
Momentum	- Giúp tránh được sự dao động của thuật toán và	- Có thể bắt đầu quá nhanh và vượt qua điểm

	giúp vượt qua các điểm local minimum. - Hiệu quả đối với các bài toán có nhiều độ dốc biến động và không đều.	tối ưu.
Adam	- Kết hợp cả momentum và RMSprop, kết hợp những ưu điểm của cả hai. - Hiệu quả trong nhiều loại mô hình và dữ liệu.	- Cần điều chỉnh nhiều hyperparameters, điều này có thể làm tăng chi phí của việc đào tạo.

- Tùy thuộc vào loại mô hình, dữ liệu, và yêu cầu cụ thể của vấn đề, mỗi thuật toán có thể hoạt động tốt hơn trong một số trường hợp. Adam thường được xem là một lựa chọn đa dụng và phổ biến, nhưng có thể cần được điều chỉnh cẩn thận để đạt được hiệu suất tốt nhất.

1.1.2. Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.

- **Continual learning** (hoặc còn được gọi là lifelong learning, incremental learning, hay online learning) là một phương pháp trong lĩnh vực học máy mà mô hình được cập nhật liên tục khi có dữ liệu mới, mà không làm mất đi kiến thức đã học trước đó. Mục tiêu chính của continual learning là giúp mô hình thích ứng và học từ dữ liệu động theo thời gian mà không cần phải huấn luyện lại toàn bộ mô hình từ đầu.
- Các thách thức của Continual Learning:
    - **Catastrophic forgetting:** Đây là vấn đề chính của Continual Learning, xảy ra khi mô hình quên kiến thức cũ khi học kiến thức mới.

    - **Hiệu suất đối với dữ liệu mới:** Làm thế nào để đảm bảo rằng mô hình có thể học hiệu quả từ dữ liệu mới mà không ảnh hưởng đến kiến thức đã học trước đó.
- Một số phương pháp và thuật toán được phát triển để giải quyết các thách thức này trong Continual Learning, bao gồm:
    - **Replay Methods:** Lưu trữ một số mẫu hoặc thông tin quan trọng từ dữ liệu cũ để sử dụng lại trong quá trình huấn luyện với dữ liệu mới.
    - **Regularization:** Sử dụng các kỹ thuật như Elastic Weight Consolidation (EWC) hoặc Synaptic Intelligence để giữ cho các trọng số liên quan đến kiến thức cũ không thay đổi quá nhanh.
    - **Architectures Designed for Continual Learning:** Phát triển kiến trúc mô hình đặc biệt để hỗ trợ Continual Learning, như Progressive Neural Networks (PNN) hoặc Memory-Augmented Networks.
    - **Dynamic Expansion:** Tăng kích thước của mô hình khi có dữ liệu mới để đảm bảo nó có đủ khả năng học.
- Continual learning chủ yếu được áp dụng trong các tình huống khi mô hình phải xử lý dữ liệu động liên tục, như trong ứng dụng Internet of Things (IoT), học máy trực tuyến, và các bài toán liên quan đến dữ liệu nguồn mở.

- **Test Production** rất quan trọng trong việc đảm bảo chất lượng của các mô hình học máy. Trong môi trường thực tế, các mô hình học máy phải đối mặt với nhiều thách thức hơn so với trong môi trường thử nghiệm, chẳng hạn như:
    - Dữ liệu thực tế có thể không đồng đều và chứa nhiều nhiễu.
    - Các yêu cầu của người dùng có thể thay đổi theo thời gian.
- Test Production giúp các mô hình học máy thích ứng với những thách thức này và đảm bảo rằng chúng hoạt động tốt trong điều kiện thực tế.
- Bằng cách tích hợp cả Continual Learning và Test Production, ta có thể xây dựng một giải pháp học máy linh hoạt, liên tục và đáng tin cậy, đáp ứng nhanh chóng với sự thay đổi trong dữ liệu và môi trường sản xuất.
- Vd:

- Chia dữ liệu thành tập huấn luyện, tập kiểm thử và tập kiểm định để kiểm thử mô hình.
- Sử dụng dữ liệu thực tế trong quá trình kiểm thử để đảm bảo chất lượng trước khi được triển khai thực tế.

1.2. 520H0120 - Vũ Lê Tấn Phát

1.2.1. Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy

1.2.1.1. Optimizer là gì?

Optimizer là thành phần chính của quá trình huấn luyện mô hình học máy. Nó có trách nhiệm điều chỉnh các trọng số của mô hình để giảm thiểu hàm mất mát, thông qua việc sử dụng gradient của hàm mất mát.

1.2.1.2. So Sánh Các Phương Pháp Optimizer

Gradient Descent:

Định nghĩa:

Là phương pháp cơ bản, hoạt động bằng cách cập nhật trọng số theo hướng ngược với đạo hàm của hàm mất mát.

Hoạt Động:

Bắt đầu với một bộ trọng số ngẫu nhiên, Gradient Descent tính toán gradient của hàm mất mát theo từng trọng số.

Cập nhật trọng số theo hướng ngược với gradient để giảm thiểu giá trị của hàm mất mát.

Quá trình này lặp lại cho đến khi đạt được sự hội tụ hoặc đạt đến số lần lặp đã định.

- Công thức cập nhật trọng số:

  $\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t)$

  Trong đó:

  - θ_t là vector trọng số tại bước thời gian t,
  - η là tốc độ học (learning rate),
  - $\nabla J(\theta_t)$ là gradient của hàm mất mát J tại θ_t.

Điểm mạnh	Điểm yếu
Dễ triển khai và hiểu. Hoạt động tốt trên dữ liệu lớn và có ít biến động. Tính ổn định: Gradient Descent thường ổn định và không nhạy cảm với các tham số cụ thể.	Tốc Độ Học Chậm: Gradient Descent có thể hội tụ chậm đối với một số bài toán, đặc biệt là khi có nhiều cực tiểu cục bộ. Đặc trưng Stochastic: Trong môi trường có nhiều nhiễu, có thể gặp khó khăn trong việc hội tụ.

Stochastic Gradient Descent (SGD):

Định Nghĩa:

Là biến thể của Gradient Descent, thực hiện cập nhật trọng số dựa trên một mini-batch ngẫu nhiên của dữ liệu.

Hoạt Động:

Chia dữ liệu thành các mini-batch nhỏ.

Đối với mỗi mini-batch, tính toán gradient của hàm mất mát và cập nhật trọng số.

Lặp lại quá trình trên cho đến khi đạt được sự hội tụ hoặc đạt đến số lần lặp đã định.

Điểm mạnh	Điểm yếu
+ Giảm thiểu thời gian huấn luyện bằng cách sử dụng mini-batch. + Thích ứng tốt với dữ liệu lớn. + Hiệu Quả trên Dữ Liệu Có Điểm Nhiễu: SGD thường hiệu quả hơn trên dữ liệu có độ nhiễu cao.	+ Khả Năng Nhảy Lên Xuống: SGD có thể nhảy lên xuống nhiều do sự ngẫu nhiên của mini-batch. + Độ Chậm và Phức Tạp: Có thể cần nhiều lần lặp và thêm các siêu tham số để đạt được hiệu suất tốt.

Stochastic Gradient Descent (SGD):

- Công thức cập nhật trọng số:

  $\theta_{t+1} = \theta_t - \eta \nabla J_i(\theta_t)$

  Trong đó:

  - i là một mẫu ngẫu nhiên được chọn từ tập dữ liệu,
  - $\nabla J_i(\theta_t)$ là gradient của hàm mất mát chỉ tính toán trên mẫu i.

Mini-Batch Gradient Descent:

- Công thức cập nhật trọng số:

  $\theta_{t+1} = \theta_t - \eta \nabla J_{\text{mini-batch}}(\theta_t)$

  Trong đó:

  - $\nabla J_{\text{mini-batch}}(\theta_t)$ là gradient được tính toán trên một mini-batch ngẫu nhiên từ dữ liệu.

Adam Optimizer:

Định Nghĩa:

Là một trong những phương pháp tiên tiến, Adam kết hợp giữa ưu điểm của Gradient Descent và SGD.

Hoạt Động:

Adam tự điều chỉnh tốc độ học cho từng tham số dựa trên độ lớn của gradient và mức độ không chắc chắn của chúng.

Nó tích hợp cả yếu tố động (momentum) và yếu tố hiệu quả của RMSprop.

Điểm mạnh	Điểm yếu
+ Kết hợp ưu điểm của cả Gradient Descent và SGD. + Hiệu quả trên nhiều loại hàm mất mát và kiến trúc mô hình. + Hiệu Quả trên Dữ Liệu Có Điểm Nhiễu: Adam thường hiệu quả hơn trên dữ liệu có độ nhiễu cao.	+ Cấu Hình Tham Số Phức Tạp: Yêu cầu cấu hình tham số phức tạp hơn. Khả Năng Quá Tinh Tế: Có thể quá tinh tế và không hiệu quả trên một số tác vụ đơn giản. + Overfitting: Có thể dẫn đến overfitting trên một số tập dữ liệu nhỏ.

Adam Optimizer:

- Công thức cập nhật trọng số:

$$m_{t+1} = \beta_1 \cdot m_t + (1 - \beta_1) \cdot \nabla J(\theta_t)$$
$$v_{t+1} = \beta_2 \cdot v_t + (1 - \beta_2) \cdot (\nabla J(\theta_t))^2$$
$$\hat{m}_{t+1} = \frac{m_{t+1}}{1-\beta_1^{t+1}}$$
$$\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^{t+1}}$$
$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}}+\epsilon}$$

Trong đó:

- m_t và v_t là các moment của gradient và bình phương gradient,
- β_1 và β_2 là các hệ số giảm dần moment,
- ϵ là một số nhỏ để tránh chia cho 0.

TỔNG QUAN:

Optimizer	Điểm Mạnh	Điểm Yếu
Gradient Descent	Dễ triển khai và hiểu. Hoạt động tốt trên dữ liệu lớn và có ít biến động. Tính ổn định và không nhạy cảm với các tham số cụ thể.	Tốc độ học chậm, đặc biệt là khi có nhiều cực tiểu cục bộ. Khó khăn trong môi trường có nhiều nhiễu.
Stochastic Gradient Descent (SGD)	Giảm thiểu thời gian huấn luyện bằng cách sử dụng mini-batch. Thích ứng tốt với dữ liệu lớn. Hiệu quả trên dữ liệu có độ nhiễu cao.	Khả năng nhảy lên xuống nhiều do sự ngẫu nhiên của mini-batch. Độ chậm và phức tạp, có thể cần nhiều lần lặp và thêm các siêu tham số để đạt hiệu suất tốt.

Adam Optimizer	Kết hợp ưu điểm của cả Gradient Descent và SGD. Hiệu quả trên nhiều loại hàm mất mát và kiến trúc mô hình. Hiệu quả trên dữ liệu có độ nhiễu cao.	Cấu hình tham số phức tạp hơn. Quá tinh tế và không hiệu quả trên một số tác vụ đơn giản. Dẫn đến overfitting trên tập dữ liệu nhỏ.

1.2.2. Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.

1.2.2.1 Continual Learning

Đặc điểm:

Khả Năng Tự Học và Linh Hoạt:

+ Tự Học (Self-learning): Mô hình có khả năng học từ dữ liệu mới mà không cần phải được huấn luyện lại từ đầu.
+ Linh Hoạt (Flexibility): Mô hình có thể linh hoạt thích ứng với sự thay đổi trong dữ liệu đầu vào hoặc khi chuyển từ một nhiệm vụ sang nhiệm vụ khác.

Quản Lý Kiến Thức Cũ và Mới:

+ Giữ Thông Tin Cũ: Mô hình không nên quên kiến thức đã học từ các tác vụ trước đó.
+ Cập Nhật Thông Tin Mới: Đồng thời, nó cũng cần có khả năng tích hợp và cập nhật kiến thức mới mà không làm mất đi hiệu suất đã đạt được.

Ứng dụng:

Học Liên Tục trong Hệ Thống Nhận Dạng:

+ Nhận Dạng Đối Tượng (Object Recognition): Hệ thống nhận dạng đối tượng có thể tiếp tục học nhận dạng các đối tượng mới mà không làm suy giảm khả năng nhận dạng đối tượng cũ.

Ứng Dụng trong Ngôn Ngữ Tự Nhiên:

+ Bài toán Sentiment Analysis: Mô hình có thể duy trì khả năng phân loại cảm xúc trên dữ liệu đánh giá mới.

Học Liên Tục trong Robot và Tương Tác Người-Máy:

+ Kiểm Soát Robot: Robot có thể học cách thực hiện nhiều nhiệm vụ khác nhau trong thời gian dài mà không cần phải được lập trình lại.

Thách Thức và Giải Pháp:

Quên (Catastrophic Forgetting):

+ Thách Thức: Mô hình có thể quên thông tin cũ khi học từ dữ liệu mới.
+ Giải Pháp: Sử dụng các phương pháp như Regularization hoặc sử dụng mô hình với kiến trúc chuyên dụng cho Continual Learning.

Nguy Cơ Overfitting:

+ Thách Thức: Nguy cơ tăng cường quá mức trên dữ liệu cũ, làm giảm khả năng tổng quát hóa.
+ Giải Pháp: Sử dụng kỹ thuật như Elastic Weight Consolidation (EWC) để ổn định các trọng số quan trọng cho các tác vụ trước.

Các Phương Pháp Continual Learning Phổ Biến:

Elastic Weight Consolidation (EWC):

+ Giữ các trọng số quan trọng của mô hình để giảm thiểu quên thông tin quan trọng.

Synaptic Intelligence:

+ Sử dụng một độ đo thông tin synaptic để ổn định các trọng số khi học từ dữ liệu mới.

Replay Memory:

+ Lưu giữ và sử dụng lại mẫu dữ liệu cũ để giúp mô hình giữ thông tin cũ.

1.2.2.2 Test Production

Đặc điểm:

Kiểm Thử Toàn Diện và Tự Động:

+ Kiểm Thử Toàn Diện (End-to-End Testing): Bao gồm việc kiểm thử toàn bộ hệ thống hoặc ứng dụng từ đầu đến cuối để đảm bảo tất cả các phần hoạt động đúng đắn và tương tác đúng đắn.
+ Kiểm Thử Tự Động (Automated Testing): Sử dụng kịch bản thử nghiệm tự động để giảm thời gian và đảm bảo nhất quán trong quá trình kiểm thử.

Phân Loại Các Kịch Bản Thử Nghiệm:

+ Kịch Bản Thử Nghiệm Dựa Trên Chức Năng (Functional Testing): Đảm bảo rằng mọi chức năng của ứng dụng hoặc hệ thống đều hoạt động đúng.
+ Kiểm Thử Hiệu Suất (Performance Testing): Đánh giá hiệu suất của ứng dụng trong các điều kiện tải cao.
+ Kịch Bản Thử Nghiệm Bảo Mật (Security Testing): Đảm bảo rằng hệ thống không có lỗ hổng bảo mật.

Ứng dụng:

Hệ Thống Tìm Kiếm và Phân Loại Dữ Liệu:

+ Mục Tiêu: Đảm bảo rằng hệ thống tìm kiếm và phân loại dữ liệu hoạt động đúng cách trong nhiều điều kiện khác nhau.
+ Kịch Bản Thử Nghiệm: Tìm kiếm các loại dữ liệu khác nhau và kiểm tra khả năng phân loại.

Ứng Dụng Di Động và Tương Tác Người-Máy:

+ Mục Tiêu: Đảm bảo ứng dụng di động và giao diện người-máy hoạt động mượt mà và phản hồi đúng đắn.
+ Kịch Bản Thử Nghiệm: Kiểm tra tương tác trên nhiều loại thiết bị và đảm bảo tính nhất quán.

Thách Thức và Giải Pháp:

Thời Gian và Nguyên Tắc 80/20:

+ Thách Thức: Thử nghiệm toàn diện có thể tốn nhiều thời gian và nguồn lực.

+ Giải Pháp: Áp dụng nguyên tắc 80/20, tập trung kiểm thử vào những phần quan trọng nhất và có thể tạo ra 80% giá trị.

Tự Động Hóa và Điều Chỉnh Liên Tục:

+ Thách Thức: Sự thay đổi liên tục trong ứng dụng yêu cầu sự điều chỉnh liên tục của kịch bản thử nghiệm.
+ Giải Pháp: Tự động hóa kịch bản và cập nhật chúng định kỳ để đảm bảo tính nhất quán.

Các Phương Pháp Test Production Phổ Biến:

Selenium và Appium:

+ Công cụ tự động hóa kiểm thử cho ứng dụng web và di động.

Jenkins và CI/CD:

+ Sử dụng Continuous Integration/Continuous Deployment để tự động hóa quá trình kiểm thử và triển khai.

Jira và TestRail:

+ Công cụ quản lý dự án và quản lý kiểm thử.

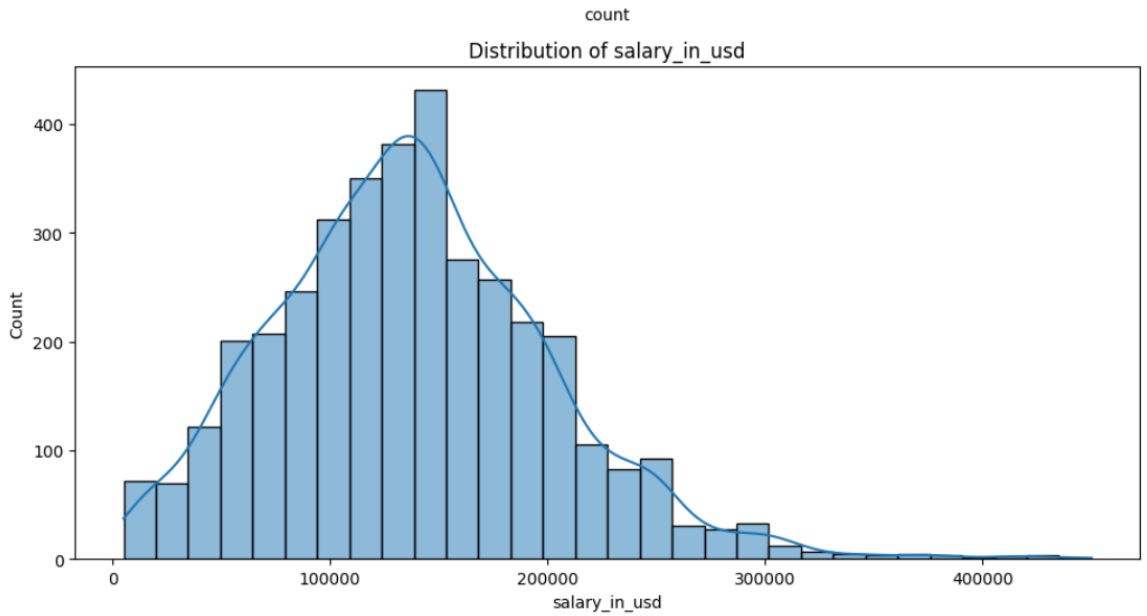
2. Bài làm chung trong nhóm

2.1. Phân tích dữ liệu

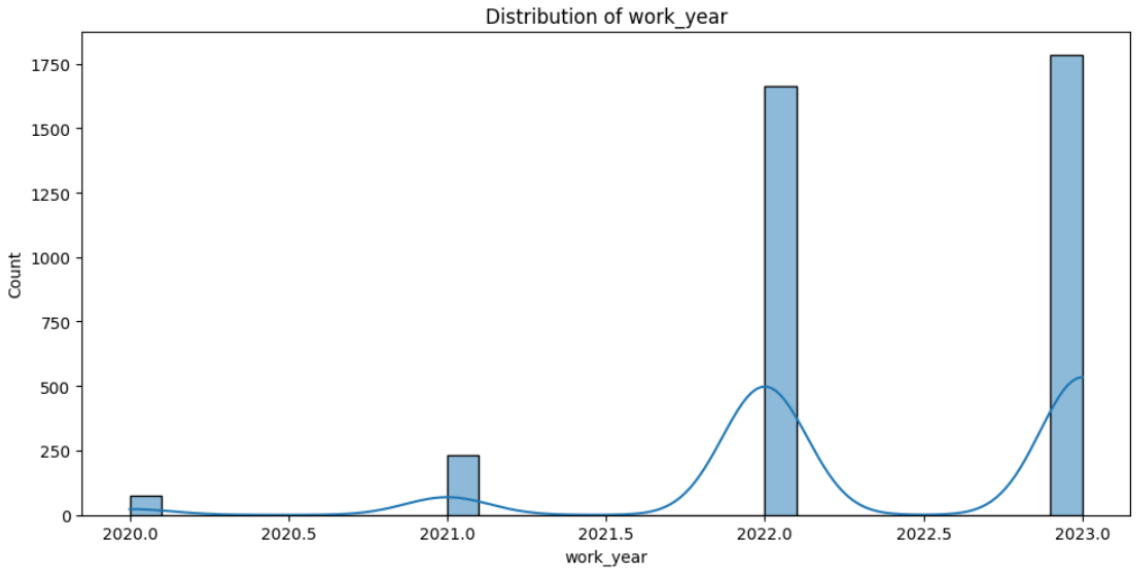
- Dữ liệu trong tệp ds_salaries.csv bao gồm thông tin về mức lương trong ngành khoa học dữ liệu và có các đặc điểm sau:
- Các Đặc Trưng/Tính Năng:
  + work_year: Năm của dữ liệu lương (Numerical).
  + experience_level: Cấp độ kinh nghiệm (Categorical).
  + employment_type: Loại hình làm việc (Categorical).
  + job_title: Chức danh công việc (Categorical).
  + salary: Mức lương (Numerical).
  + salary_currency: Đơn vị tiền tệ của lương (Categorical).
  + salary_in_usd: Mức lương tính bằng USD (Numerical).
  + employee_residence: Nơi cư trú của nhân viên (Categorical).
  + remote_ratio: Tỷ lệ làm việc từ xa (Numerical).

+ company_location: Vị trí của công ty (Categorical).
+ company_size: Kích thước của công ty (Categorical).

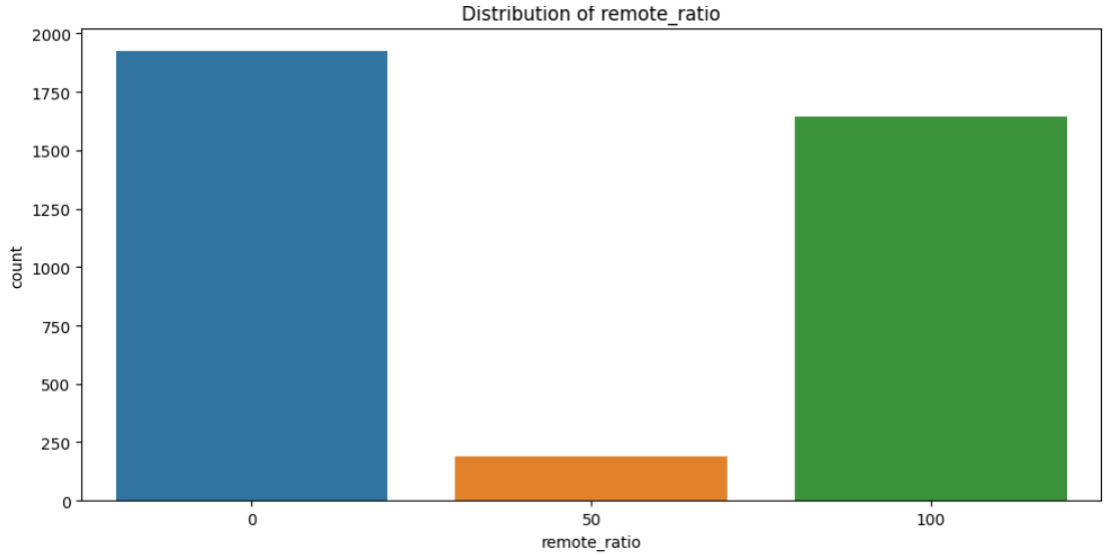
- Tóm tắt thống kê của các đặc trưng số:
  + work_year: Năm từ 2020 đến 2023.
  + salary: Dao động rộng, với trung bình khoảng 190,695 và tối đa là 30,400,000 theo đơn vị tiền tệ gốc.
  + salary_in_usd: Dao động từ 5,132 đến 450,000 USD, với trung bình khoảng 137,570 USD.
  + remote_ratio: Dao động từ 0% đến 100%, cho thấy sự đa dạng trong các hình thức làm việc từ xa.



Hình 2. Mức lương tính bằng USD

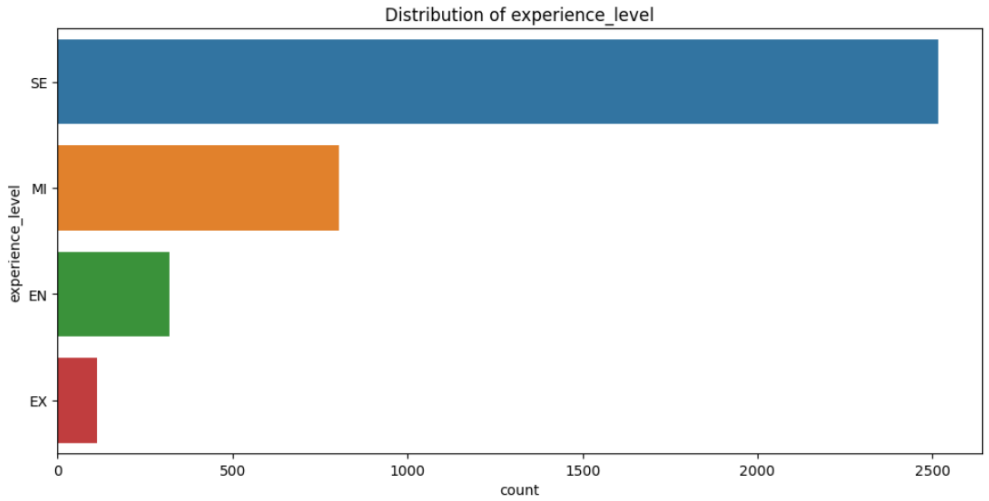


Hình 3. Năm từ 2020 đến 2023

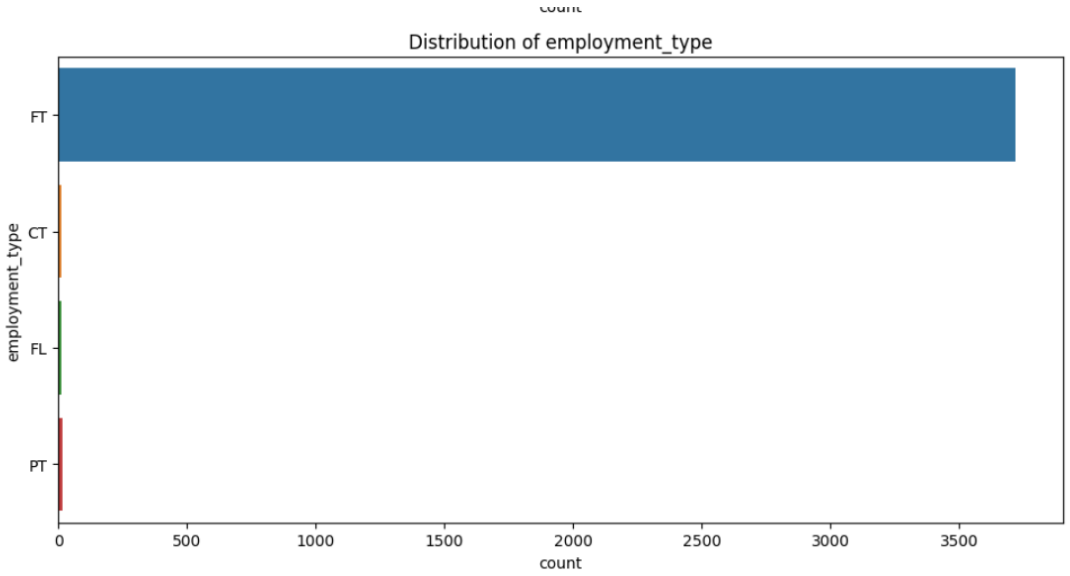


Hình 4. Tỷ lệ làm việc từ xa

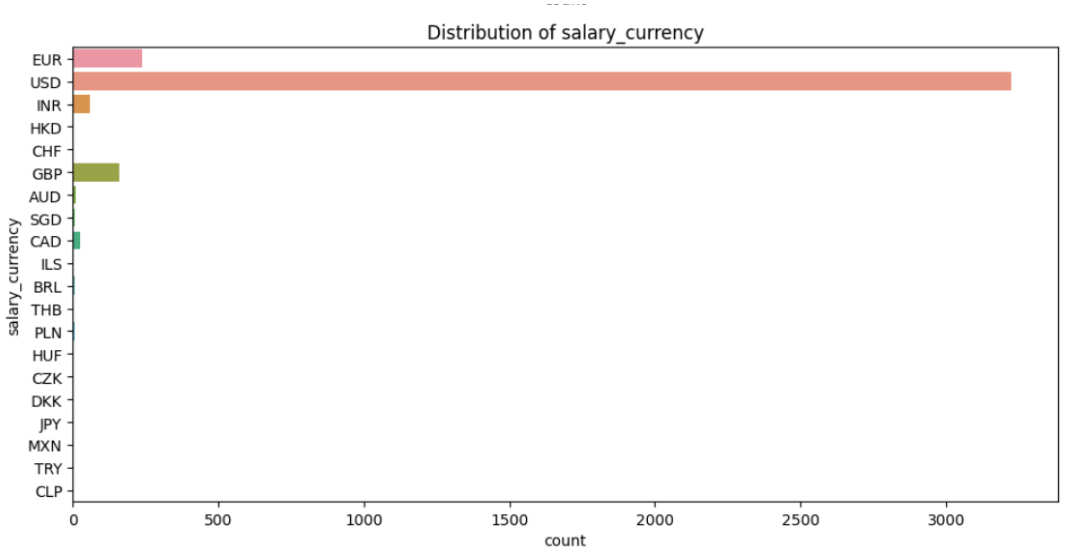
Ngoài ra còn có các biểu đồ về cấp độ kinh nghiệm làm việc và hình thức làm việc cho ta thấy chiếm đa số là nhân viên cấp cao và làm việc toàn thời gian. Đồng thời họ cũng xài đơn vị tiền tệ USD nhiều nhất ⇒ Điều này sẽ giúp cho phần salary_in_usd có nhiều biến động tốt vì kinh nghiệm lâu năm và hình thức toàn thời gian giúp họ tối ưu lương của mình ở mức cao nhất. Đồng thời USD là 1 trong những đồng tiền có giá trị nhất hiện nay :



Hình 5. Cấp độ kinh nghiệm

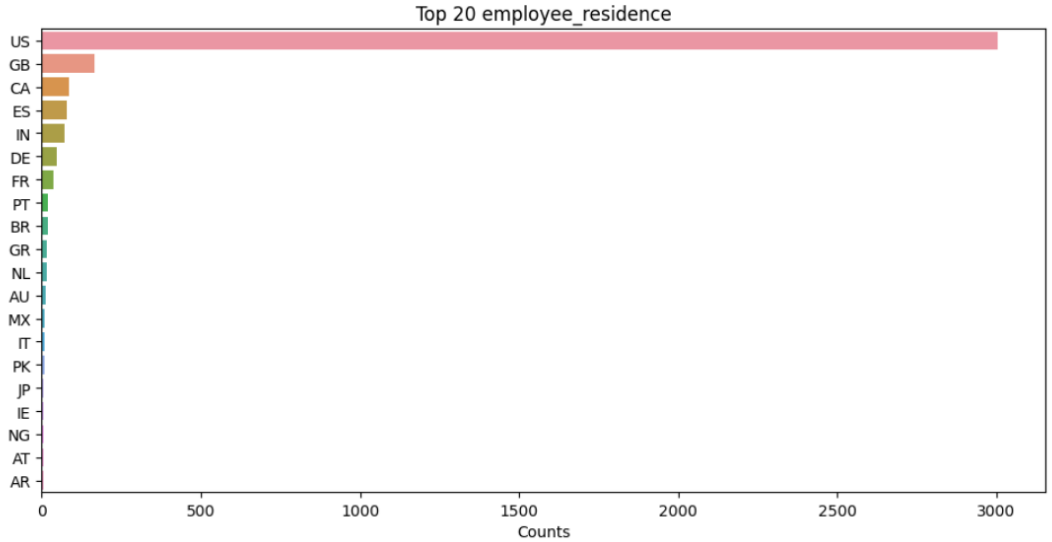


Hình 6. Loại hình làm việc

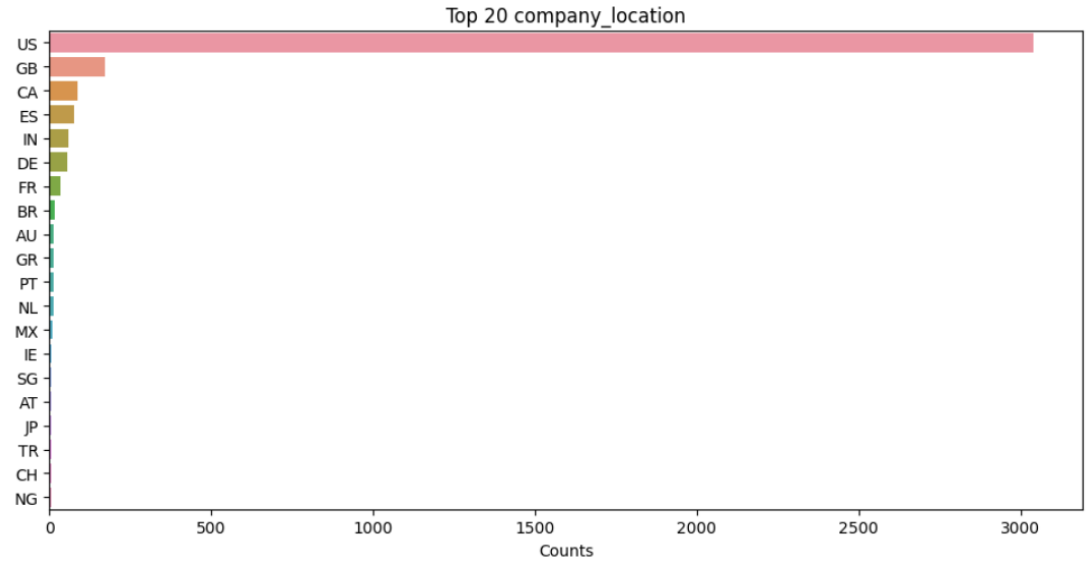


Hình 7. Đơn vị tiền tệ của lương

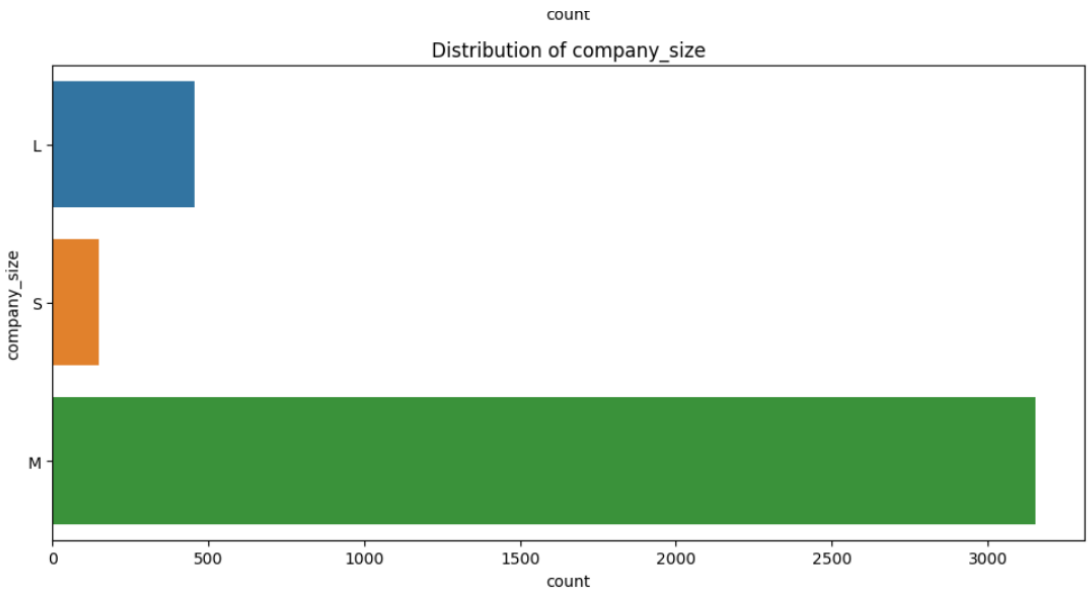
Cuối cùng là top 20 của khu vực của nhân viên sống và cũng top các vị trí công ty phân bố và kích thước của công ty. Vì dữ liệu có nhiều cột nên chỉ lấy ra top 20 để thống kê. Những điều này cũng cho ta thấy nơi cư trú của nhân viên lẫn các vị trí của công ty khá là đồng điệu với nhau khiến cho việc đi lại tới công ty cũng dễ dàng cho nhân viên hơn. Tuy nhiên kích thước công ty chiếm đa số là M nên điều này có thể làm ảnh hưởng 1 chút đến salary_in_usd nhưng có lẽ không đáng kể cho lắm. Đa phần sẽ là lợi thế nhờ sự đồng điệu của nơi cư trú nhân viên và vị trí công ty :



Hình 8. Nơi cư trú của nhân viên



Hình 9. Vị trí của công ty



Hình 10. Kích thước của công ty

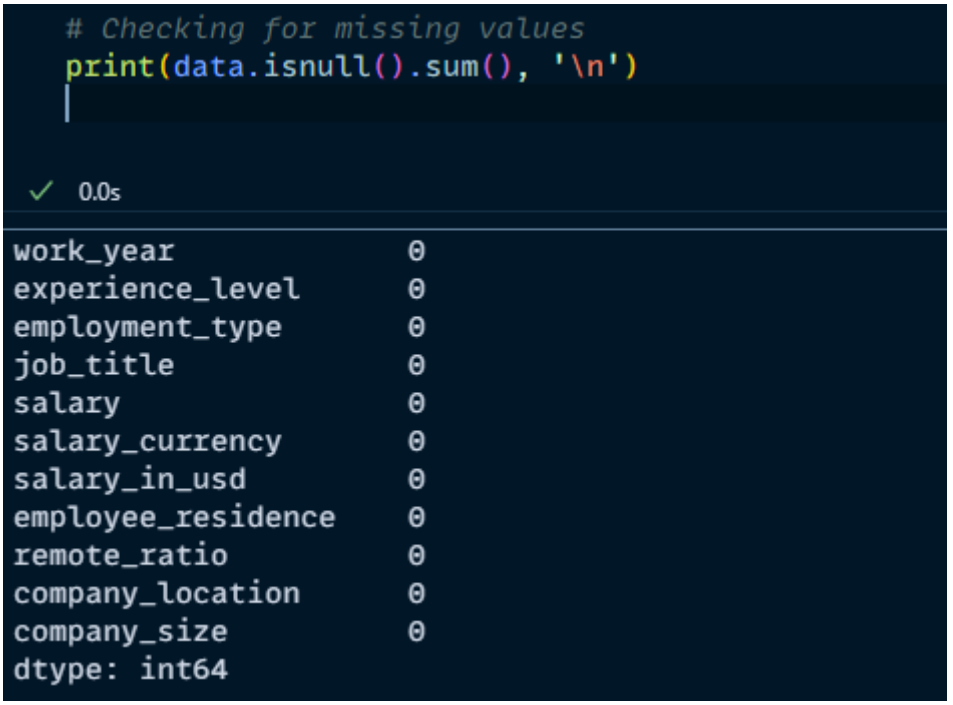
- Kế Hoạch Phân Tích và Trực Quan Hóa:
    + Phân Tích Thống Kê: Hiểu rõ về xu hướng trung tâm, sự phân tán và hình dạng của các thuộc tính số trong bộ dữ liệu.
    + Tầm Quan Trọng của Đặc Trưng: Phân tích cách các đặc trưng như cấp độ kinh nghiệm, loại hình làm việc, chức danh công việc và kích thước công ty ảnh hưởng đến mức lương.
    + Trực Quan Hóa Dữ Liệu:

- Biểu đồ histogram cho các đặc trưng số để hiểu rõ về phân phối và các ngoại lệ.
- Biểu đồ cột cho các đặc trưng phân loại để phân tích tần suất của các hạng mục khác nhau.
- Biểu đồ scatter hoặc các biểu đồ phù hợp khác để khảo sát mối quan hệ giữa các biến, đặc biệt là lương và các đặc trưng khác.

+ Nhận Định và Giải Thích: Rút ra những hiểu biết từ phân tích và trực quan hóa để hiểu rõ hơn về các yếu tố ảnh hưởng đến mức lương trong ngành khoa học dữ liệu và các xu hướng trong ngành.

2.2. Ứng dụng mô hình học máy

- Các mô hình học máy được sử dụng trong phần này bao gồm: Linear Regression, GaussianNB, Decision Tree, KNN, Random Forest, AdaBoost.
- Sau khi đọc dữ liệu, ta sẽ tiến hành kiểm tra các giá trị rỗng và loại bỏ nó. Trong tập dữ liệu hiện tại, không có dữ liệu nào bị rỗng.



Hình 11. Kiểm tra giá trị rỗng

- Để dễ phân loại, ta sẽ tự phân chia lương của nhân viên theo hình thức sau.

```python
def categorise_data(data):
    salary = []
    for i in range(len(data)):
        if data['salary_in_usd'].iloc[i] >= 100000:
            salary.append('>=100K')
        else:
            salary.append('<100K')
    data['salary_in_usd'] = salary

categorise_data(data)

print('\n', data['salary_in_usd'].value_counts())
```

Hình 12. Phân chia lương nhân viên thành 2 loại

```
 salary_in_usd
>=100K    2764
<100K      991
```

Hình 13. Kết quả sau khi phân loại

- Vì dữ liệu bao gồm cả dữ liệu phân loại và dữ liệu số, nên ta sẽ sử dụng ‘LabelEncoder’ để chuyển đổi các giá trị phân loại thành số nguyên.

```python
# Preprocessing data
types = data.dtypes
names = data.columns

le = LabelEncoder()

for i in range(len(types)):
    if types.iloc[i] == 'object':
        le.fit_transform(data[names[i]])
        data[names[i]] = le.transform(data[names[i]])

print(data.dtypes)
data.head()
```

```
97] ✓ 0.0s
..
employee_residence    0
remote_ratio          0
company_location      0
company_size          0
dtype: int64

work_year             int64
experience_level      int32
employment_type       int32
job_title             int32
salary                int64
salary_currency       int32
salary_in_usd         int64
employee_residence    int32
remote_ratio          int64
company_location      int32
company_size          int32
dtype: object
work_year             int64
experience_level      int32
employment_type       int32
job_title             int32
salary                int64
salary_currency       int32
salary_in_usd         int64
employee_residence    int32
remote_ratio          int64
company_location      int32
```

Hình 14. Tiền xử lý dữ liệu bằng LabelEncoder

	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	2023	3	2	84	80000	7	0	26	100	25	0
1	2023	2	0	66	30000	19	0	75	100	70	2
2	2023	2	0	66	25500	19	0	75	100	70	2
3	2023	3	2	47	175000	19	0	11	100	12	1
4	2023	3	2	47	120000	19	0	11	100	12	1

Hình 15. Kết quả sau khi tiền xử lý

- Sau khi tiền xử lý, ta sẽ tiến hành và phân chia dữ liệu thành 2 biến X (features) và y (target) cũng như chuẩn hóa dữ liệu. Sau khi chuẩn hóa, ta sẽ phân chia các biến trên thành các tập training và testing.

```
# Splitting data into target and features
X = data.drop(['salary_in_usd', 'salary', 'salary_currency'], axis=1)
y = data['salary_in_usd']

print(X.shape)
print(y.shape)

# Scaling the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print(X_scaled)

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

Hình 16. Phân chia và chuẩn hóa dữ liệu

- Tiếp đến, ta sẽ fit các dữ liệu vào các mô hình học máy tương ứng và dự đoán kết quả đầu ra.

```
# Applying Linear Regression
linear_regression = LinearRegression()
linear_regression.fit(X_train, y_train)
linear_regression_predictions = linear_regression.predict(X_test)

# Applying Naive Bayes
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)
naive_bayes_predictions = naive_bayes.predict(X_test)

# Applying Random Forests
random_forest = RandomForestClassifier()
random_forest.fit(X_train, y_train)
random_forest_predictions = random_forest.predict(X_test)

# Applying AdaBoost
gradient_boosting = AdaBoostClassifier()
gradient_boosting.fit(X_train, y_train)
gradient_boosting_predictions = gradient_boosting.predict(X_test)

# Applying Decision Trees
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
decision_tree_predictions = decision_tree.predict(X_test)

# Applying k-Nearest Neighbors
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
knn_predictions = knn.predict(X_test)
```

Hình 17. Sử dụng các mô hình học máy

- Cuối cùng, ta sẽ đánh giá và so sánh các mô hình học máy với nhau bằng cách sử dụng ‘accuracy_score’ và ‘mean_square_error’ (đối với Linear Regression).

```
# Evaluating the performance of each algorithm
linear_regression_mse = mean_squared_error(y_test, linear_regression_predictions)
naive_bayes_accuracy = accuracy_score(y_test, naive_bayes_predictions)
random_forest_accuracy = accuracy_score(y_test, random_forest_predictions)
ada_boost_accuracy = accuracy_score(y_test, gradient_boosting_predictions)
decision_tree_accuracy = accuracy_score(y_test, decision_tree_predictions)
knn_accuracy = accuracy_score(y_test, knn_predictions)

# Printing the accuracies
print("Linear Regression MSE:", linear_regression_mse)
print("Naive Bayes Accuracy:", naive_bayes_accuracy)
print("Random Forests Accuracy:", random_forest_accuracy)
print("AdaBoost Accuracy:", ada_boost_accuracy)
print("Decision Tree Accuracy:", decision_tree_accuracy)
print("k-Nearest Neighbors Accuracy:", knn_accuracy)
```

Hình 18. Đánh giá các mô hình học máy.

```
Linear Regression MSE: 0.13804893154249998
Naive Bayes Accuracy: 0.8175765645805593
Random Forests Accuracy: 0.8375499334221038
AdaBoost Accuracy: 0.8388814913448736
Decision Tree Accuracy: 0.8255659121171771
k-Nearest Neighbors Accuracy: 0.829560585885486
```

Hình 19. Kết quả đánh giá các mô hình học máy

2.3. Sử dụng Feed Forward Neural Network và Recurrent Neural Network

2.3.1 Feed Forward Neural Network

Trước tiên ta lựa chọn các thuộc tính nhằm mục đích hỗ trợ cho việc dự đoán và mục tiêu ta dự đoán ở đây sẽ là salary_in_usd

```
# Selecting relevant features and the target variable
features = ['experience_level', 'employment_type', 'company_size', 'remote_ratio']
target = 'salary_in_usd'
```

Hình 20. Lựa chọn thuộc tính và kết quả

Tiếp đến ta phải xử lý các tính năng:

Categorical Features: Các tính năng phân loại (như loại hình công việc, kích thước công ty) được xử lý bằng OneHotEncoder.

Numerical Features: Các tính năng số (như tỷ lệ làm việc từ xa) được chuẩn hóa bằng StandardScaler.

```
# Create transformers for categorical and numerical features
categorical_transformer = OneHotEncoder(handle_unknown='ignore')
numerical_transformer = StandardScaler()

# Column Transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

Hình 21. Phân chia Numerical và Categorical

Chia dữ liệu:

```
# Splitting 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=0)

# Apply transformations
X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)
```

Hình 22. Phân chia train và test

Mô hình FFNN được xây dựng với các lớp 'Dense':

+ Các lớp ẩn sử dụng hàm kích hoạt relu.
+ Lớp đầu ra cho bài toán hồi quy không sử dụng hàm kích hoạt.

```
# Step 3: Define the Neural Network Model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1)  # Output layer for regression
])

model.compile(optimizer='adam', loss='mean_squared_error')
```

Hình 23. Mô hình FFNN

Huấn luyện mô hình bằng dữ liệu với số lần lặp (epochs) và kích thước mẻ (batch_size):

Sau khi huấn luyện, mô hình được đánh giá trên tập kiểm tra để xác định hiệu suất dự đoán. Cuối cùng sử dụng nó để dự đoán salary_in_usd trên dữ liệu mới.

```
# Step 4: Train the Model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2)
```

```
# Step 5: Evaluate the Model
test_loss = model.evaluate(X_test, y_test)
print(f"Test Loss: {test_loss}")

# Step 6: Make Predictions (Optional)
predictions = model.predict(X_test)
```

Hình 24. In ra kết quả FFNN

Kết quả:

```
Epoch 95/100
76/76 [==============================] - 0s 2ms/step - loss: 3032327424.0000 - val_loss: 3194412032.0000
Epoch 96/100
76/76 [==============================] - 0s 2ms/step - loss: 3034167040.0000 - val_loss: 3174517248.0000
Epoch 97/100
76/76 [==============================] - 0s 2ms/step - loss: 3034700544.0000 - val_loss: 3177668096.0000
Epoch 98/100
76/76 [==============================] - 0s 2ms/step - loss: 3031356416.0000 - val_loss: 3167005184.0000
Epoch 99/100
76/76 [==============================] - 0s 2ms/step - loss: 3039113984.0000 - val_loss: 3173361408.0000
Epoch 100/100
76/76 [==============================] - 0s 2ms/step - loss: 3027433216.0000 - val_loss: 3168348160.0000
24/24 [==============================] - 0s 1ms/step - loss: 3320945664.0000
Test Loss: 3320945664.0
24/24 [==============================] - 0s 1ms/step
```

Hình 25. Kết quả FFNN

2.3.2 Recurrent Neural Network

Bước đầu tiên tương tự như FFNN:

Dữ liệu được tải từ file CSV. Các tính năng (features) và biến mục tiêu (target) được chọn. Ở đây, features bao gồm các thông tin như cấp độ kinh nghiệm, loại hình công việc, kích thước công ty và tỷ lệ làm việc từ xa. target là salary_in_usd (lương tính bằng USD):

Tiếp đến

Dữ liệu được chia thành hai phần: huấn luyện và kiểm tra, với tỷ lệ thông thường là 80/20

Dữ liệu được chuyển đổi thành hình dạng phù hợp với RNN, tức là dữ liệu phải có hình dạng 3 chiều (num_samples, num_timesteps, num_features). Trong trường hợp này, giả định mỗi mẫu dữ liệu chỉ có 1 bước thời gian.

```
# Step 3: Split the Data
X_train, X_test, y_train, y_test = train_test_split(X_processed, y, test_size=0.2, random_state=0)

# Step 4: Reshape Data for RNN
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

Hình 26. Chia dữ liệu

Cuối cùng xây dựng mô hình RNN và xuất kết quả tương tự FFNN:

```
# Step 5: Build the RNN Model
model = Sequential([
    SimpleRNN(50, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')

# Step 6: Train the Model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2)

# Step 7: Evaluate the Model
test_loss = model.evaluate(X_test, y_test)
print(f"Test Loss: {test_loss}")

# Step 8: Make Predictions (Optional)
predictions = model.predict(X_test)
```

Hình 27. Mô hình RNN

Kết quả:

```
Epoch 94/100
76/76 [==============================] - 0s 2ms/step - loss: 19616247808.0000 - val_loss: 18934581248.0000
Epoch 95/100
76/76 [==============================] - 0s 2ms/step - loss: 19562663936.0000 - val_loss: 18882314240.0000
Epoch 96/100
76/76 [==============================] - 0s 2ms/step - loss: 19509200896.0000 - val_loss: 18829938688.0000
Epoch 97/100
76/76 [==============================] - 0s 2ms/step - loss: 19454984192.0000 - val_loss: 18776686592.0000
Epoch 98/100
76/76 [==============================] - 0s 2ms/step - loss: 19400697856.0000 - val_loss: 18723758080.0000
Epoch 99/100
76/76 [==============================] - 0s 2ms/step - loss: 19346122752.0000 - val_loss: 18670010368.0000
Epoch 100/100
76/76 [==============================] - 0s 2ms/step - loss: 19291312128.0000 - val_loss: 18616637440.0000
24/24 [==============================] - 0s 1ms/step - loss: 20984082432.0000
Test Loss: 20984082432.0
24/24 [==============================] - 0s 1ms/step
```

Hình 28. Kết quả RNN

2.4. Sử dụng kỹ thuật tránh Overfitting cho các mô hình học máy trên

- Các kỹ thuật tránh overfitting thường được sử dụng là: Regularization, Cross-Validation, Early Stopping, …
- Đầu tiên là Regularization, ở phần này ta sẽ áp dụng kỹ thuật này vào mô hình Feed Forward Neural Network ở trên. Ta sẽ thêm lớp regularization (L1 hoặc L2) vào các lớp để kiểm soát trọng số để giảm overfitting.

- L2 regularization thêm một thành phần vào hàm loss, bằng cách thêm giá trị bình phương của trọng số vào hàm loss. Điều này giúp làm giảm độ lớn của các hệ số mô hình, khiến mô hình đơn giản hơn và ít có khả năng overfitting.

```
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],), kernel_regularizer=l2(0.01)),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1)
])
```

Hình 29. Thêm L2 Regularization vào 1 lớp của mô hình

- Kế tiếp là Cross-Validation, lần này ta sẽ sử dụng mô hình Naive Bayes Gaussian để ứng dụng.
- Đối với mô hình Naive Bayes Gaussian, cross-validation có thể giúp:
    + Đánh giá hiệu suất của mô hình trên tập dữ liệu test. Cross-validation giúp đánh giá hiệu suất của mô hình trên các tập dữ liệu ngẫu nhiên khác nhau, giúp chúng ta có được đánh giá khách quan hơn về hiệu suất của mô hình.
    + Phát hiện overfitting. Nếu điểm số cross-validation thấp hơn đáng kể so với điểm số training, điều đó có thể cho thấy overfitting.
    + Chọn tham số tối ưu cho mô hình. Cross-validation có thể được sử dụng để tìm các tham số tối ưu cho mô hình, chẳng hạn như tham số Laplace smoothing.

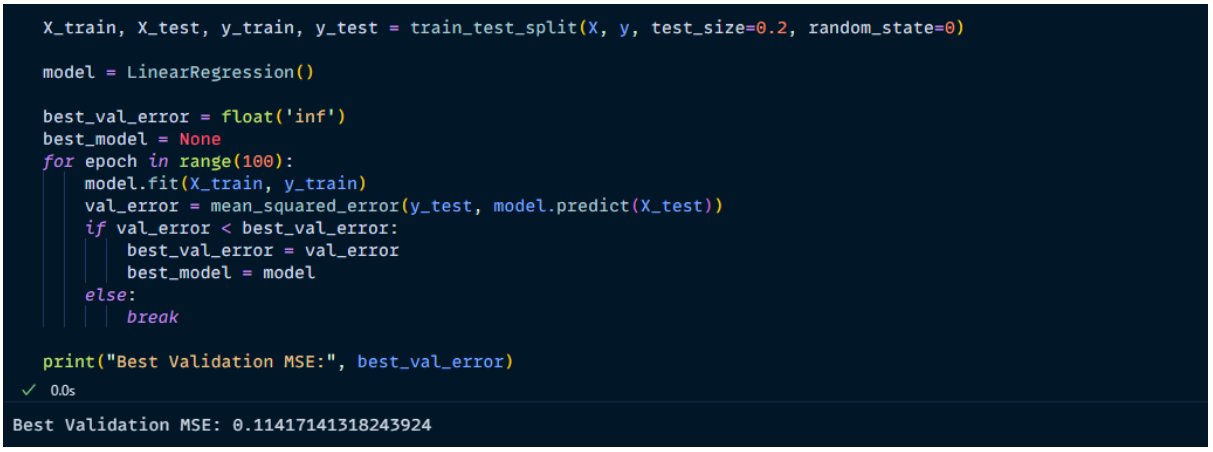
```
from sklearn.model_selection import cross_val_score

model = GaussianNB(var_smoothing=0.9)
score = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')

print("Accuracy:", score.mean())
✓ 0.0s
Accuracy: 0.7323566278424847
```

Hình 30. Cross-Validation sử dụng cross_val_score

- Một số tham số tùy chọn của phương thức cross_val_score():
    + n_splits: Số lần cross-validation. Giá trị mặc định là 5.
    + cv: Lớp cross-validation. Giá trị mặc định là KFold().

+ scoring: Hàm đánh giá được sử dụng. Giá trị mặc định là accuracy_score().
+ fit_params: Các tham số bổ sung được sử dụng để đào tạo mô hình.
+ verbose: Mức độ chi tiết của đầu ra. Giá trị mặc định là 0.

- Kỹ thuật Early Stopping là một kỹ thuật giúp ngăn chặn quá trình đào tạo mô hình quá lâu, dẫn đến overfitting. Kỹ thuật này hoạt động bằng cách theo dõi hiệu suất của mô hình trên tập dữ liệu kiểm tra và ngừng đào tạo khi hiệu suất trên tập dữ liệu kiểm tra không còn cải thiện.
- Việc sử dụng kỹ thuật Early Stopping có thể mang lại một số lợi ích cho mô hình Linear Regression, bao gồm:
  + Giảm nguy cơ overfitting: Kỹ thuật Early Stopping có thể giúp giảm nguy cơ overfitting bằng cách ngừng đào tạo mô hình khi mô hình bắt đầu học thuộc lòng dữ liệu đào tạo.
  + Tăng hiệu suất trên tập dữ liệu kiểm tra.
  + Giảm thời gian đào tạo.



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

model = LinearRegression()

best_val_error = float('inf')
best_model = None
for epoch in range(100):
    model.fit(X_train, y_train)
    val_error = mean_squared_error(y_test, model.predict(X_test))
    if val_error < best_val_error:
        best_val_error = val_error
        best_model = model
    else:
        break

print("Best Validation MSE:", best_val_error)
```

✓ 0.0s

```
Best Validation MSE: 0.11417141318243924
```

Hình 31. Early Stopping

2.5. Phân tích và cải thiện

Vấn đề: Do sự chênh lệch khá lớn giữa các mức lương ở salary_in_usd nên dẫn đến việc khả năng dự đoán của các mô hình không có tỷ lệ chính xác cao. Vì vậy ta sẽ tìm những cách để cải thiện độ chính xác:

Đầu tiên đối với FFNN: Áp dụng biến đổi logarit để giảm thiểu ảnh hưởng các giá trị lượng lớn:

```
# Apply log transformation to the target variable
data['salary_in_usd'] = np.log1p(data['salary_in_usd'])
```

Hình 32. Áp dụng biến đổi logarit

Sử dụng thêm các lớp regularization, dropout và tăng số lượng Dense lên(tránh overfitting và cho học thêm lâu hơn):

```
# Step 3: Define the Neural Network Model with Regularization and Dropout
model = Sequential([
    Dense(256, activation='relu', input_shape=(X_train.shape[1],), kernel_regularizer=l2(0.001)),
    Dropout(0.5),
    Dense(128, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.5),
    Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
    Dense(1)  # Output layer for regression
])

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='mean_squared_error')
```

Hình 33. Thay đổi, cải tiến FFNN

Kết quả:

```
Epoch 143/150
38/38 [==============================] - 0s 3ms/step - loss: 0.3109 - val_loss: 0.3115
Epoch 144/150
38/38 [==============================] - 0s 3ms/step - loss: 0.3130 - val_loss: 0.3123
Epoch 145/150
38/38 [==============================] - 0s 3ms/step - loss: 0.3114 - val_loss: 0.3054
Epoch 146/150
38/38 [==============================] - 0s 3ms/step - loss: 0.3126 - val_loss: 0.3078
Epoch 147/150
38/38 [==============================] - 0s 3ms/step - loss: 0.3114 - val_loss: 0.3047
Epoch 148/150
38/38 [==============================] - 0s 3ms/step - loss: 0.3111 - val_loss: 0.3052
Epoch 149/150
38/38 [==============================] - 0s 3ms/step - loss: 0.3067 - val_loss: 0.3065
Epoch 150/150
38/38 [==============================] - 0s 3ms/step - loss: 0.3053 - val_loss: 0.3031
24/24 [==============================] - 0s 2ms/step - loss: 0.2916
Improved Test Loss: 0.2915867269039154
24/24 [==============================] - 0s 1ms/step
```

Hình 34. Kết quả mô hình sau khi cải tiến

So với kết quả FFNN cũ thì đã cải thiện hơn rất nhiều:

```
Test Loss: 3320945664.0
24/24 [==============================] - 0s 1ms/step
```

Còn mô hình RNN thường được sử dụng cho dữ liệu có tính chất tuần tự hoặc thời gian, như xử lý ngôn ngữ tự nhiên hoặc dự báo chuỗi thời gian. Trong trường hợp của bộ dữ liệu về lương, nếu không có yếu tố thời gian hoặc tuần tự rõ ràng, việc sử dụng RNN có thể không phải là lựa chọn tối ưu. Nên đó sẽ là điểm yếu mạnh của RNN không nên cố cải thiện thay vào đó ta sẽ dụng FFNN dự đoán lương sẽ tối ưu.

THAM KHẢO

1. https://viblo.asia/p/optimizer-hieu-sau-ve-cac-thuat-toan-toi-uu-gdsgdadam-Qbq5QQ9E5D8
2. https://machinelearningcoban.com/2017/01/16/gradientdescent2/