Áp dụng phương pháp Ensemble trong bài toán dự đoán mức lương

CS116.N11 HKI 2022-2023

GVHD
TS Nguyễn Vinh Tiệp

SV
Đặng Anh Tiến
20520800

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
THÀNH PHỐ HỒ CHÍ MINH 2023

Mục lục

1 Giới thiệu

1.1 Động lực

Khi tham gia thị trường lao động, định giá giá trị của bản thân là vô cùng quan trọng. Không những giúp người lao động hiểu được vị trí của bản thân trong thị trường, mà còn giúp họ không bị thiệt thòi khi thỏa thuận mức lương trước các nhà tuyển dụng. Mức thu nhập hàng năm cũng là một trong những thước đo giúp đánh giá được giá trị của ứng viên có thể mang lại cho doanh nghiệp và thị trường. Thông thường, mức lương càng cao càng cho thấy được giá trị cũng như phản ánh được chất lượng, trình độ của người lao động vô cùng tốt.

1.2 Giới thiệu bài toán

Tuy nhiên, để dự đoán mức lương thực sự là việc không hề đơn giản, bởi mức lương phụ thuộc vào rất nhiều yếu tố khác nhau. Ví dụ như quốc gia của ứng viên mong muốn làm việc cũng ảnh hưởng rất nhiều, bởi các quốc gia đã phát triển sẽ có mức thu nhập cao hơn. Hay số năm kinh nghiệm cũng là một trong các thước đo đánh giá năng lực làm việc. Ngoài ra còn rất nhiều yếu tố khác như trình độ học vấn, vị trí tuyển dụng, các công cụ đã từng làm việc,…

Đồ án sẽ trình bày giải pháp vận dụng Machine Learning giải quyết bài toán dự đoán mức lương lập trình viên, vấn đề thuộc nhóm bài toán Hồi quy (regression). Đối với lập trình viên, mức lương được đánh giá bởi nhiều yếu tố khác nhau, chẳng hạn các lĩnh vực xu hướng như AI, Blockchain sẽ có mức thu nhập cao hơn các ngành khác. Hay các quốc gia kì lân về công nghệ như Hoa Kỳ, Anh Quốc sẽ có nhu cầu tuyển dụng rất cao. Các mô hình Machine Learning sẽ học các đặc trưng này để đưa ra các dự đoán khách quan nhất.

1.3 Phương pháp đánh giá

Các độ đo phổ biến như MSE, MAE thường sử dụng để đánh giá hiệu năng của mô hình Hồi quy, chúng đều đo mức độ sai khác giữa kết quả dự đoán và kết quả thực tế, phản ánh chất lượng của mô hình.

MSE (Mean Square Error) là bình phương độ sai khác của giá trị dự đoán và thực tế, đây là độ đo vô cùng phổ biến để đánh giá bài toán Hồi quy. MSE càng thấp cho thấy mức độ sai khác càng thấp, cho thấy hiệu năng của mô hình càng tốt:

$$MSE = \frac{1}{N}\sum (Y_{TRUE} - Y_{PRED})^2$$

MAE (Mean Absolute Error) là trị tuyệt đối sai khác của giá trị dự đoán và thực tế, cùng với MSE là các độ đo phổ biến trong bài toán Hồi quy, tuy nhiên, MAE không bị ảnh hưởng quá nhiều bởi các outlier so với MSE:

$$MAE = \frac{1}{N}\sum |Y_{TRUE} - Y_{PRED}|$$

R^2-score cũng là độ đo thường sử dụng để đánh giá bài toán hồi quy, R^2-score có giá trị trong khoảng [0,1], R^2-score tốt nhất khi có giá trị bằng 1. Độ đo này cho biết lượng thông tin của output được dự đoán bởi các thuộc tính input:

$$R^2 = 1 - \frac{\sum (Y_{TRUE} - Y_{PRED})^2}{\sum (Y_{TRUE} - Y_{MEAN})^2}$$

R^2-score giúp so sánh được hiệu năng của nhiều mô hình với nhau mô hình với nhau, còn MSE và MAE cho biết hiệu năng của mô hình tốt như thế nào.

2 Dữ liệu

2.1 Nguồn dữ liệu.

Sử dụng **Stack Overflow Annual Developer Survey** là nguồn dữ liệu chính trong đồ án này. Đây là kết quả khảo sát lập trình viên trên nền tảng **StackOverflow** - là một trong những nền tảng hàng đầu để các lập trình viên trên thế giới đặt ra các câu hỏi, tìm giải pháp cho vấn đề của mình. Chủ yếu khảo sát về các vấn đề xung quanh công nghệ cũng như về lĩnh vực Khoa học máy tính, bài khảo sát này được thực hiện hàng năm từ 2011 cho đến nay.

Đồ án này sử dụng khảo sát năm 2022 để phân tích, xử lí và huấn luyện mô hình. Với hơn 70.000 kết quả khảo sát đến từ hơn 180 quốc gia. Bài khảo sát chủ yếu hỏi về các vấn đề xung quanh lập trình viên, từ những khía cạnh xung quanh những người chỉ mới bắt đầu code, từ những ngôn ngữ lập trình yêu thích cho đến những công nghệ quản lí phiên bản cũng như là các nền tảng lập trình yêu thích của những lập trình viên chuyên nghiệp.

Dữ liệu bao gồm 78 thuộc tính với hơn 73000 mẫu. Các feature chủ yếu là các câu hỏi về các khía cạnh nền tảng giáo dục, công việc, ngôn ngữ lập trình, các nền tảng,..

2.2 Làm sạch dữ liệu.

Dữ liệu là yếu tố quan trọng nhất để huấn luyện mô hình, vì vậy việc đảm bảo dữ liệu đã được chuẩn bị sạch sẽ và đầy đủ trước khi phân tích và huấn luyện là một trong những bước tốn nhiều thời gian nhất nhưng lại vô cùng quan trọng.

Dữ liệu đa dạng nhóm đối tượng khảo sát, từ những người mới học code, code vì sở thích đến những lập trình viên chuyên nghiệp. Bài toán hướng tới dự đoán mức lương của các lập trình viên nói chung nên sẽ chỉ trích chọn những mẫu dữ liệu có đối tượng là lập trình viên chuyên nghiệp. Ngoài ra, mong muốn của việc dự đoán mức lương là để tham khảo từ những người làm việc thực tế nên sẽ trích chọn các mẫu dự liệu có tình trạng việc làm thuộc nhóm: Được thuê, freelancer, làm việc theo hợp đồng và tự làm chủ.

Cuối cùng là sẽ bỏ đi các mẫu dữ liệu có các thuộc tính bị lỗi, bỏ trống,… Vì bộ dữ liệu rất lớn nên việc loại bỏ các sample lỗi không làm ảnh hưởng đến số lượng dữ liệu.

2.3 Chọn lọc thuộc tính.

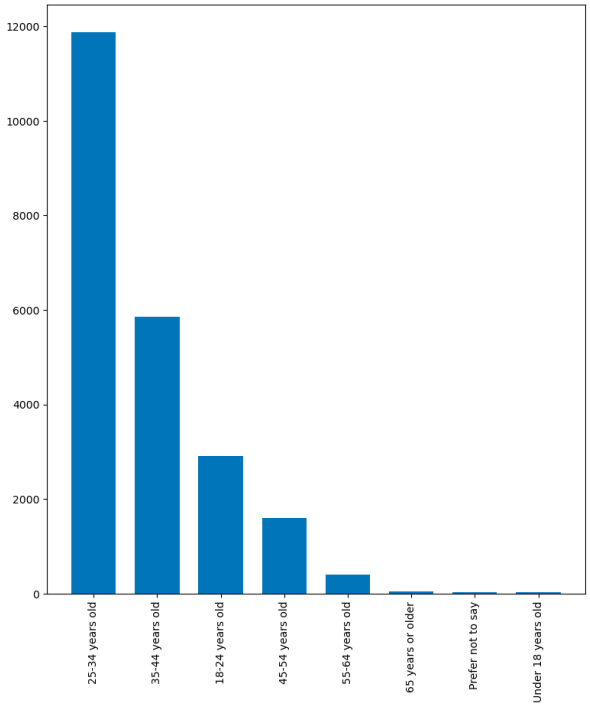
Với 78 feature khác nhau, trích chọn 9 feature làm input và output sẽ dự đoán 1 feature là mức lương theo năm đã được chuyển đổi theo đồng dollar.

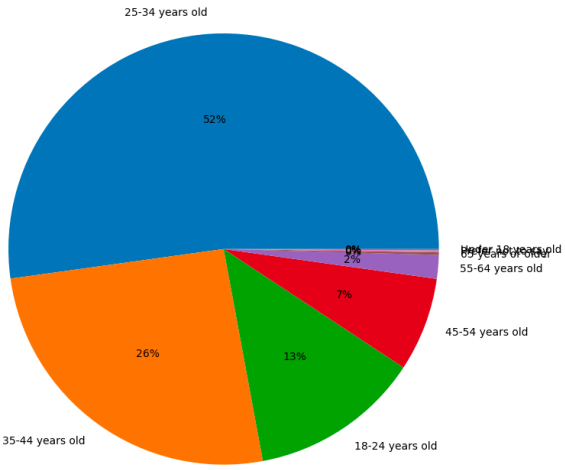
Các thuộc tính bao gồm:

- **RemoteWork**: trạng thái làm việc: On-site, Hybrid hay Remote.
- **EdLevel**: trình độ học vấn: Đại học, Thạc sĩ, Tiến sĩ, …
- **YearsCodePro**: số năm kinh nghiệm làm việc.
- **DevType**: danh mục lập trình viên, là các chức vụ mà người làm khảo sát đang đảm nhiệm trong doanh nghiệp. (Frontend, Backend, DevOps, Data Scientist, …).
- **LanguageHaveWorkedWith**: ngôn ngữ lập trình (C++, Java, Python, …).
- **PlatformHaveWorkWith**: nền tảng điện toán đám mây (AWS, Google Cloud,…).
- **ToolsTechHaveWorkWith**: các công cụ mở rộng thêm (Docker, Homebrew, npm).
- **Country**: quốc gia của người làm khảo sát.
- **Age**: nhóm tuổi của người làm khảo sát.
- **ConvertedCompYearly**: mức lương theo năm đã được đổi sang đơn vị dollar. Mô hình sẽ dựa vào các feature trên để đưa ra output là mức lương.

2.4 Visualize dữ liệu.

2.4.1 Thuộc tính Age

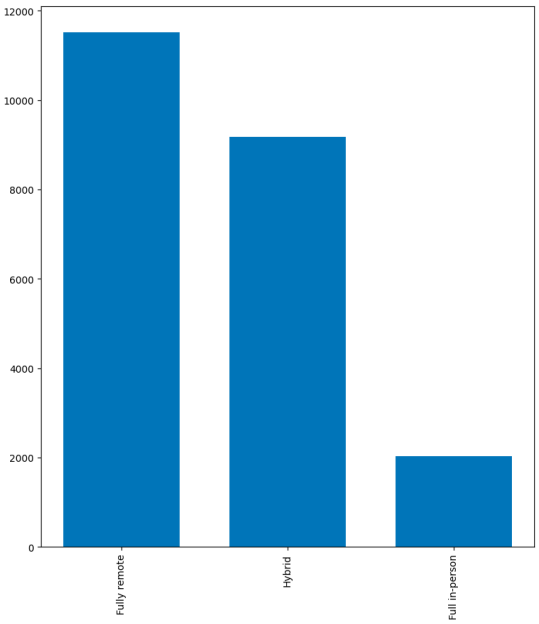


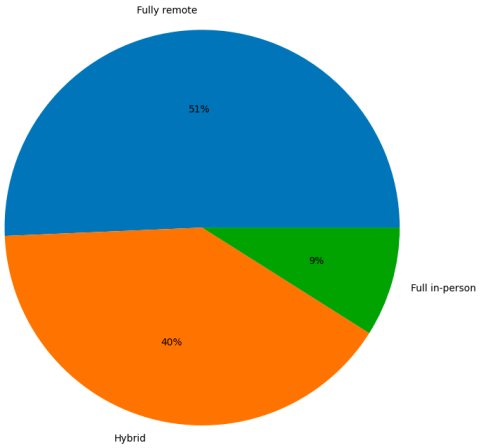


Đa số các đối tượng khảo sát có độ tuổi trong khoảng từ 18 đến 45 tuổi và sau đó giảm dần đều. Cho thấy tuổi nghề của các lập trình viên tương đối dài, tuổi nghề càng lớn họ sẽ có những lựa chọn công việc khác thay vì tiếp tục nghề lập trình.

Vì 3 nhóm tuổi cuối có tần suất xuất hiện rất thấp, loại bỏ chúng không làm ảnh hưởng đến độ đa dạng dữ liệu. Gộp 2 nhóm tuổi "45-54" và "55 – 64" thành nhóm mới "Over 45".

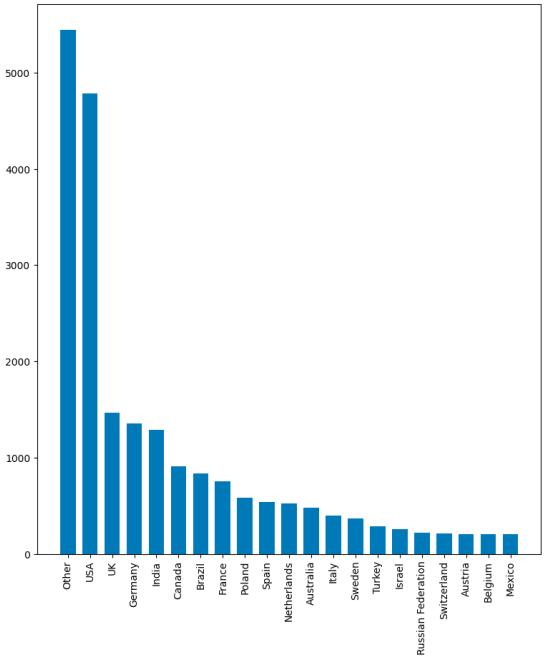
2.4.2 Thuộc tính RemoteWork

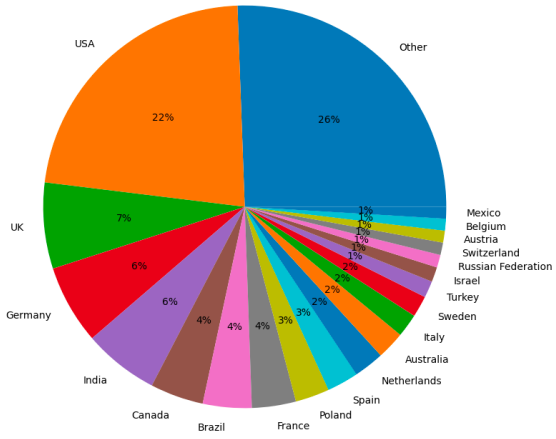




Có vẻ như sau đợt dịch vừa qua, xu thế làm việc dần chuyển qua làm việc tại nhà, đặc biệt với công việc lập trình. Đa số các nhóm đối tượng làm việc tại nhà hoặc Hybrid, số đối tượng làm việc trực tiếp rất ít, chỉ chiếm 9%.

2.4.3 Thuộc tính Country

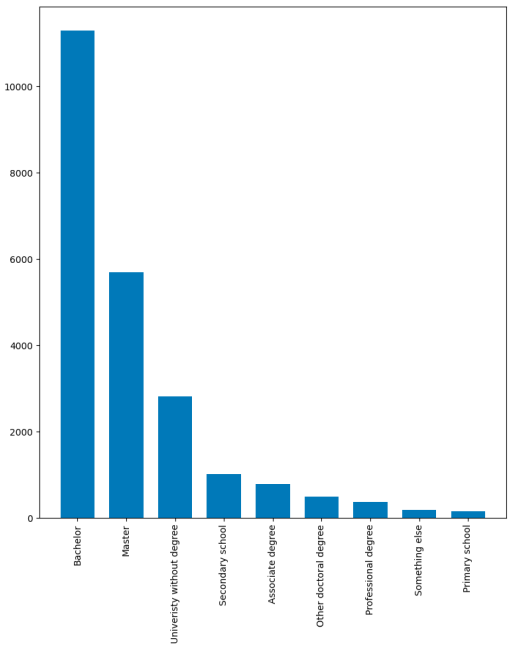


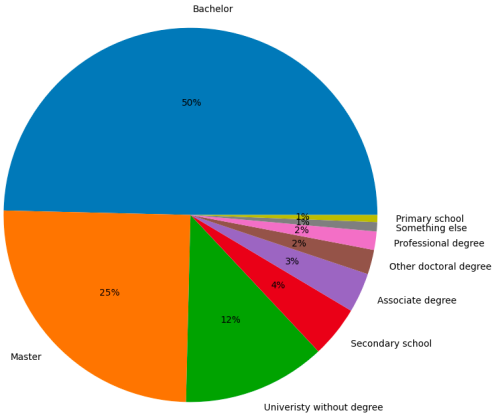


Đa số các lập trình viên tham gia khảo sát đến từ các quốc gia châu Âu, châu Mĩ. Đặc biệt Hoa Kỳ chiếm đa số các lập trình viên, cho thấy quốc gia này là lựa chọn hàng đầu trong lĩnh vực công nghệ thông tin, kế đến là Vương quốc Anh, Ấn độ, và các nước châu Âu.

Gồm có 180 quốc gia với tần suất xuất hiện dao động khác nhau từ chỉ vài trăm đến vài chục nghìn. Gom nhóm các quốc gia có tần suất xuất hiện thấp hơn 200 về một nhóm "Other".

2.4.4 Thuộc tính EdLevel

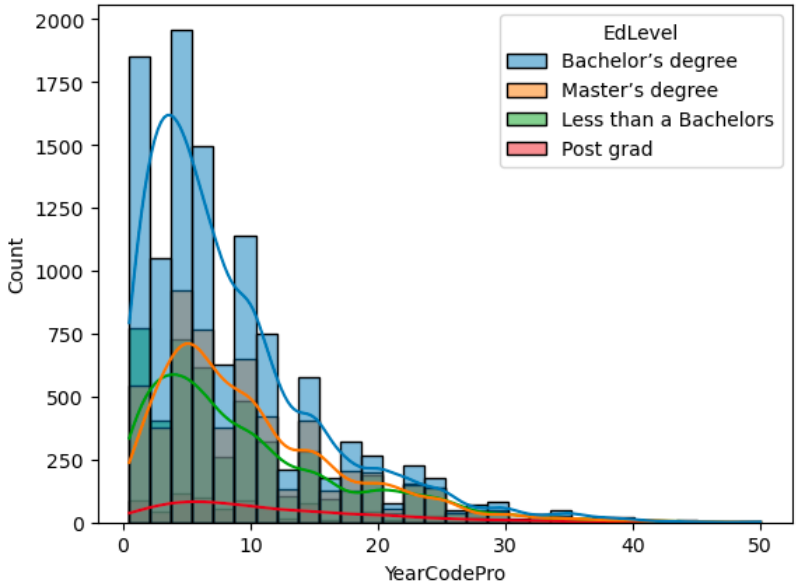




Đa số các lập trình viên có bằng Đại học, chiếm đến 50%. Tiếp theo là loại bằng Thạc sĩ, chiếm 25%. Kế đến là nhóm đối tượng có trải qua đại học/cao đẳng nhưng không có bằng , chiếm đến 12%. Nhóm đối tượng có trình độ học vấn thấp hơn đại học, và hay trình độ tiến sĩ trở lên rất ít. Chúng ta thường biết đến câu nói "bằng cấp không quan trọng, quan trọng là kỹ năng", quan điểm này có vẻ đúng với lĩnh vực lập trình, tuy không có bằng cấp nhưng vẫn có việc làm, 12% cho thấy đây là một tỉ lệ tương đối lớn trong nhóm các đối tượng khảo sát

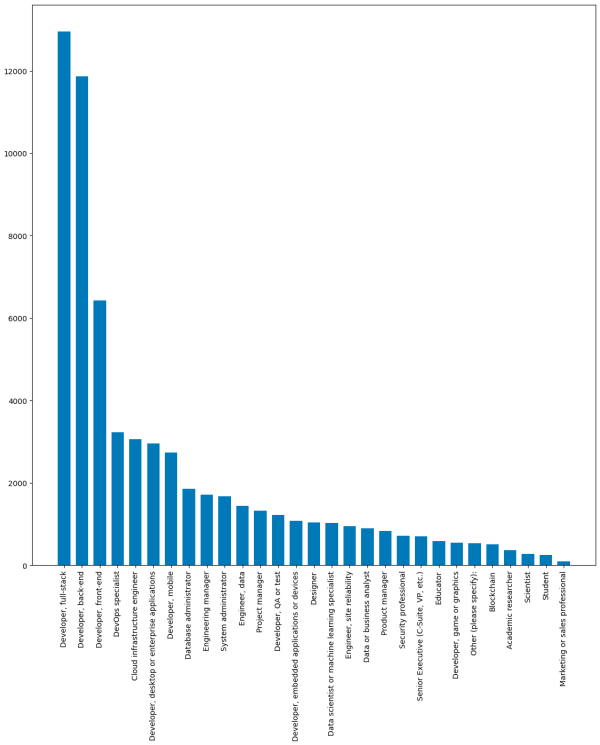
Vì "Professional degree" và "Other doctoral" có tần suất xuất hiện thấp và giống nhau nên được gộp chung thành "Post grad". Nhóm có bằng cấp là Đại học và Thạc sĩ có tần suất xuất hiện cao nên được giữ lại. Các nhóm còn lại được gộp chung thành "Less than a Bachelor".

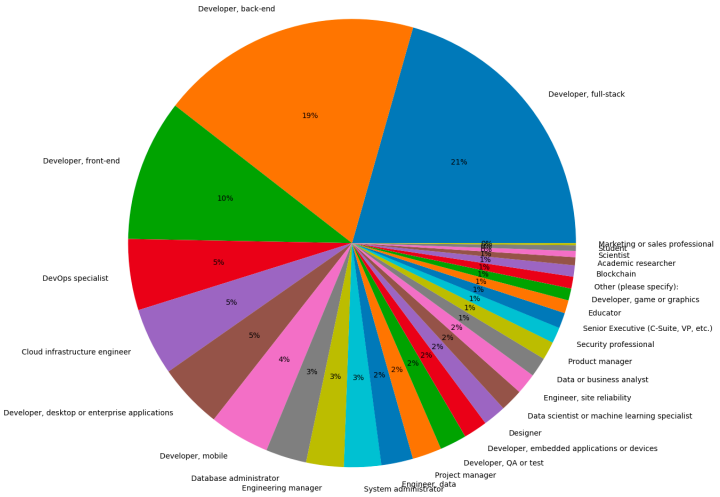
2.4.5 Thuộc tính YearsCodePro



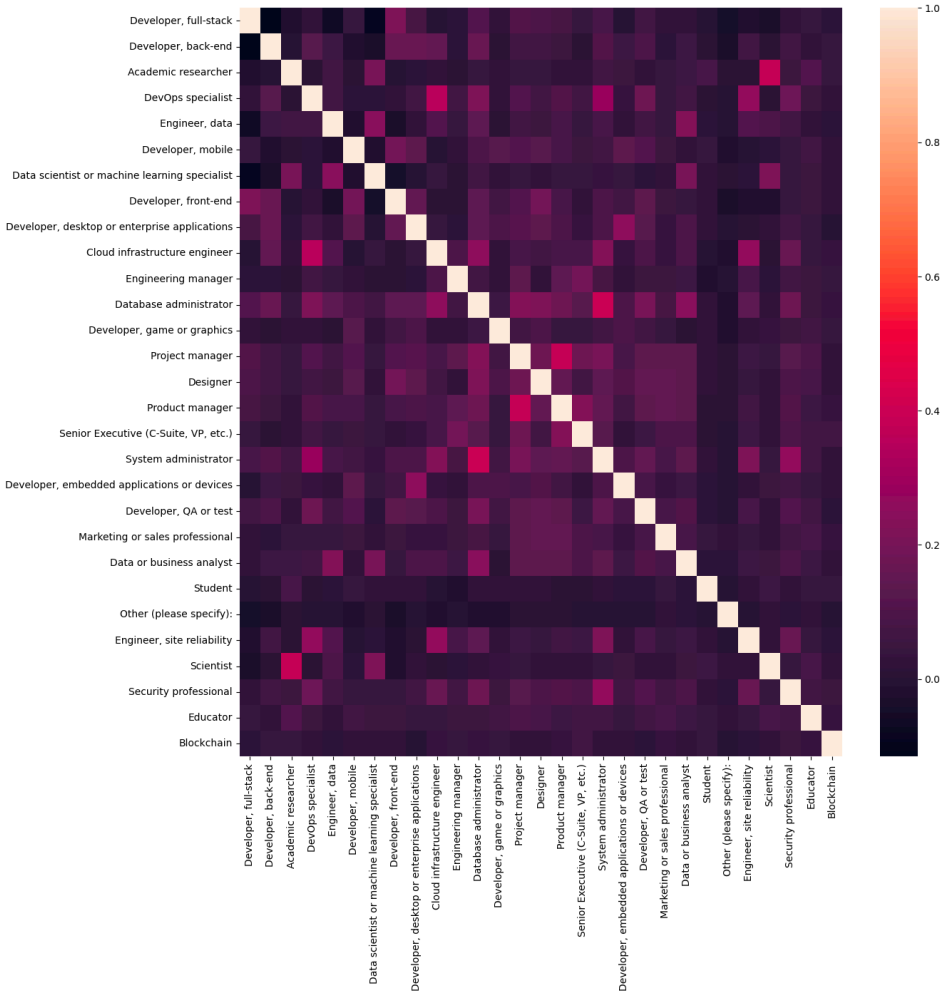
Thuộc tính cho biết số năm kinh nghiệm làm việc không bao gồm những năm học. Đa số nằm trong khoảng từ 0-25 năm, cho thấy đây là độ tuổi kinh nghiệm phổ biến của lập trình viên. Nhóm người làm nghề lập trình đa số có tuổi nghề rất trẻ, chỉ có một số ít là từ 20 năm trở lên.

2.4.6 Thuộc tính DevType





Dựa vào các biểu đồ trên có thể thấy lĩnh vực Web, Mobile, DevOps và Cloud là các lĩnh vực vô cùng phổ biến, chiếm hơn 50% số lượng mẫu khảo sát. Tuy đây là các lĩnh vực đã có từ rất lâu, nhưng vẫn không hề có dấu hiệu bị lỗi thời, ít nhất ở thời điểm năm 2022.

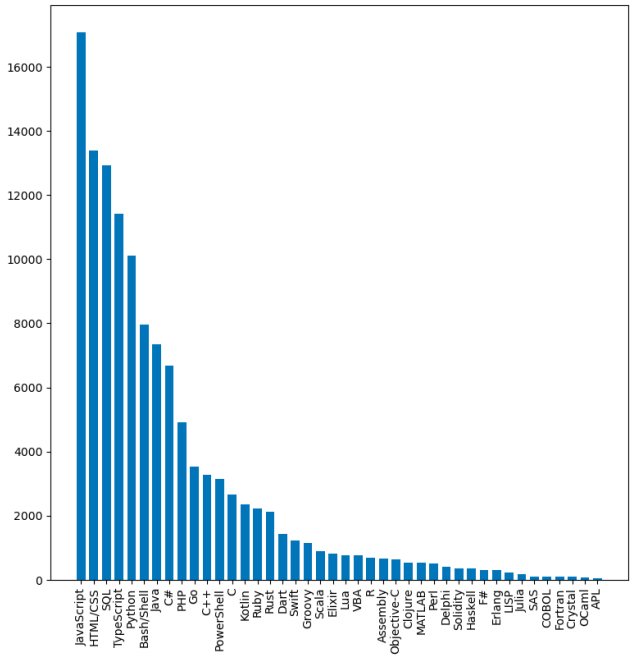


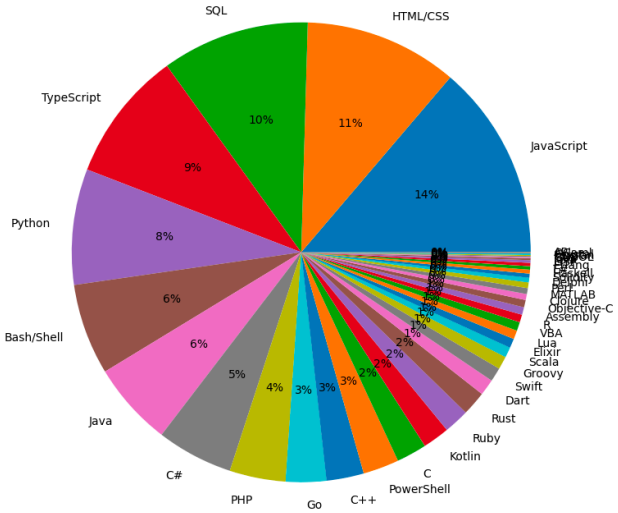
Dựa vào heatmap trên, dữ liệu cho ta thấy được có một số lĩnh vực có sự tương đồng cao:

- Cloud Infrastruture với DevOps
- Academic Researcher với Scientist
- System administator với Database administator
- Product Manager với Project Manager

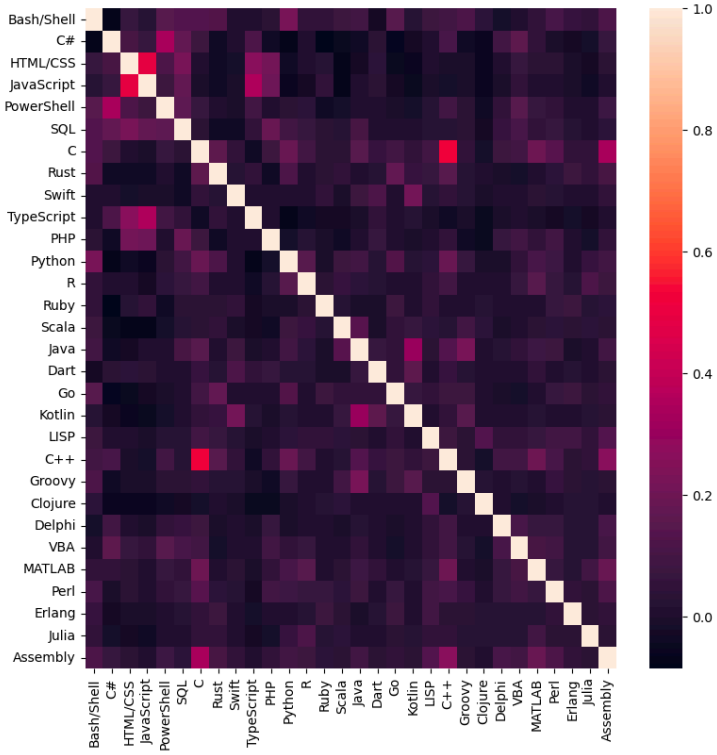
Chỉ giữ lại những mẫu dữ liệu có tần suất xuất hiện trên 900.

2.4.7 Thuộc tính LanguageHaveWorkedWith





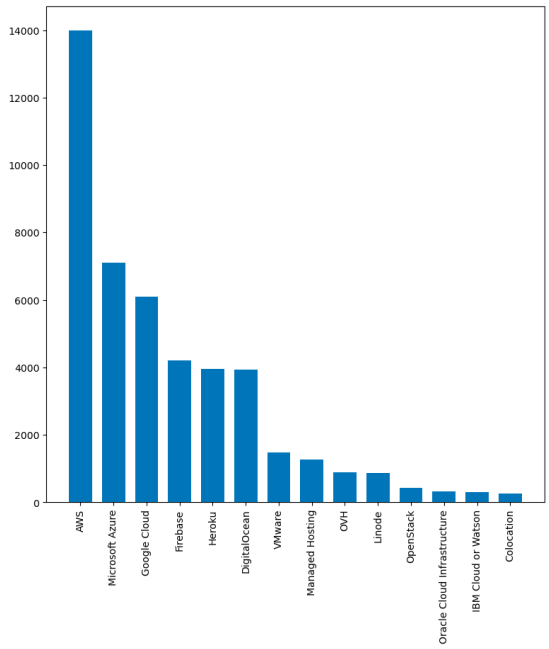
Lĩnh vực web, mobile và DevOp là các lĩnh vực việc làm phổ biến, thế nên không khó hiểu khi các ngôn ngữ phổ biến cho các lĩnh vực này phổ biến. Các ngôn ngữ như C/C++ tuy ra đời đã lâu, nhưng vẫn còn phổ biến hiện nay.

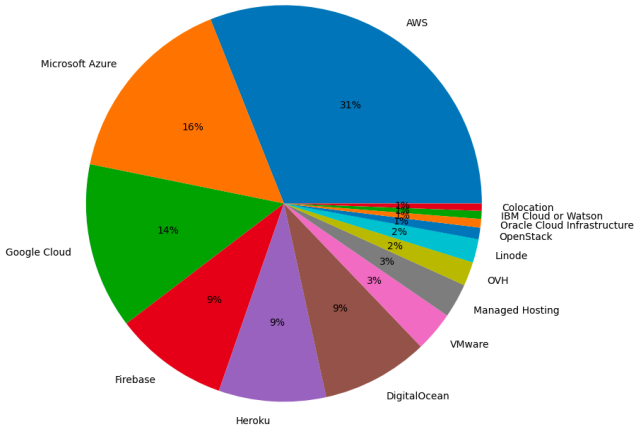


Từ heatmap trên cho ta thấy nhóm các ngôn ngữ C và C++, HTML/CSS và JavaScript có sự tương đồng cao. Có thể những lập trình viên làm việc với C thì sẽ có sự hiểu biết về C++, nhóm lập trình viên biết đến HTML/CSS thì sẽ biết thêm JavaScript.

Chỉ giữ lại các ngôn ngữ lập trình có tần suất xuất hiện trên 2000.

2.4.8 Thuộc tính PlatformHaveWorkedWith

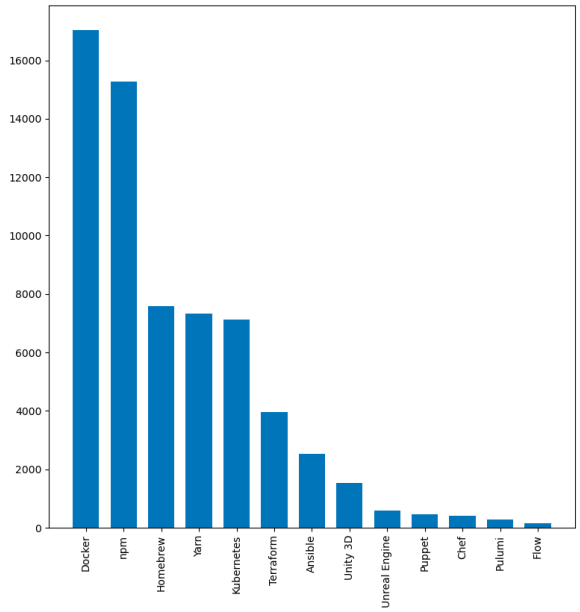


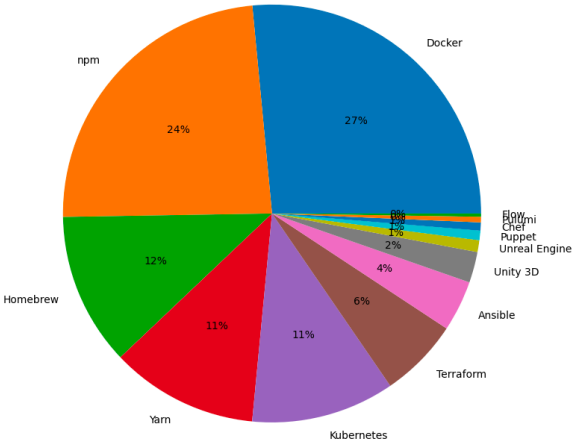


Nền tảng đám mây phổ biến nhất là AWS (Amazon), Azure (Microsoft), Google Cloud (Google), Firebase (Google).

Chỉ giữ lại các platform có tần suất xuất hiện trên 2000.

2.4.9 Thuộc tính ToolsTechHaveWorkedWith



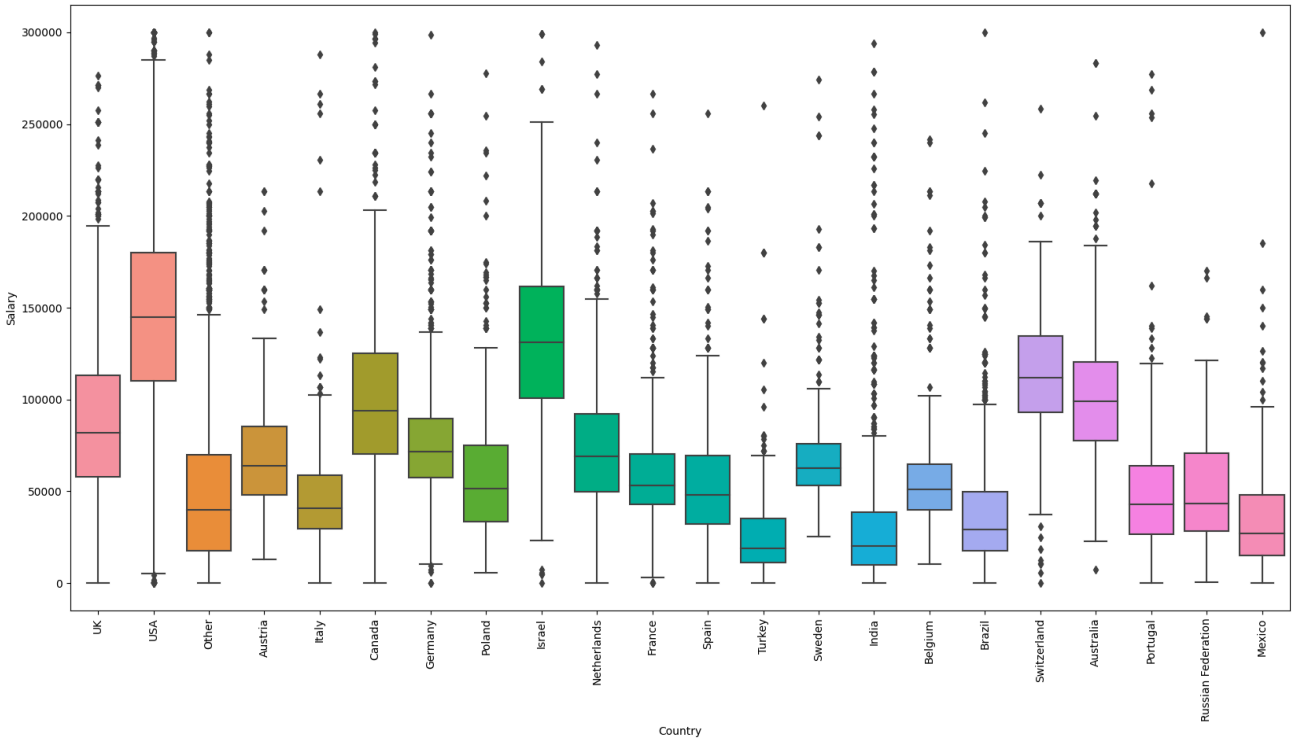


Các công cụ khác mà các lập trình viên thường sử dụng thêm là Docker, Npm, Homebrew, Yarn và Kubernetes.

Chỉ giữ lại các tool có tần suất xuất hiện trên 5000.

2.4.10 Thuộc tính Salary

Từ biểu đồ boxplot cho thấy có rất nhiều outlier, để loại bỏ các outlier, chỉ giữ lại các giá trị thấp hơn 0.3 * 1e6.



Các thị trường Hoa Kỳ, Isarel và Vương quốc Anh có mức lương trung bình cao nhất, kế đến là thị trường thuộc châu Âu.

3 Mô hình

Nhìn chung dữ liệu rất thưa (sparse data), đa số các feature có giá trị 0. Một số mô hình Machine Learning không được thiết kế để xử lí tốt trên các dạng dữ liệu như thế này, các mô hình rất dễ bị overfitting bởi vì có quá nhiều feature, vì đa số các feature có giá trị 0 nên có thể không mang quá nhiều thông tin.

Ensemble learning là giải pháp được sử dụng để giải quyết các vấn đề này. Khi nói về Ensemble, người ta thường nghĩ đến tập hợp nhiều mô hình kết hợp với nhau để đưa ra kết quả dự đoán cuối cùng. Bằng việc kết hợp nhiều "weak-learner", mỗi weak-learner được học trên một khía cạnh nhỏ nào đó của dữ liệu, kết quả cuối cùng sẽ là sự đóng góp của nhiều mô hình, phương pháp này không những giúp khắc phục vấn đề Overfitting mà còn cải thiện độ chính xác của mô hình đáng kể. Luôn có sự xuất hiện của Ensemble model giành chiến thắng ở các cuộc thi Machine Leanring lớn như Kaggle. Đồ án sử dụng phương pháp Ensemble kết hợp nhiều base model hình với nhau, Decision Tree được sử dụng để làm base model.

3.1 Decision Tree Regressor

Decision Tree Regressor về cơ bản là Decision Tree nhưng sử dụng cho bài toán regression, kết quả dự đoán là giá trị liên tục thay vì là giá trị rời rạc.

Trong Decision Tree, ta đã biết được rằng Tree đặt câu hỏi điều kiện tại các non-leaf node để đưa ra các dự đoán. Điều này thực hiện trong bài toán Classification bằng cách sử dụng 2 độ đo Entropy và Information Gain. Khi áp dụng cho bài toán Regression, độ đo Entropy không thể được sử dụng cho các giá trị liên tục. Độ đo thường dùng trong bài toán Regression, cho biết mức độ sai khác giữa giá trị dự đoán và giá trị thực tế đó chính là Mean Square Error (MSE):

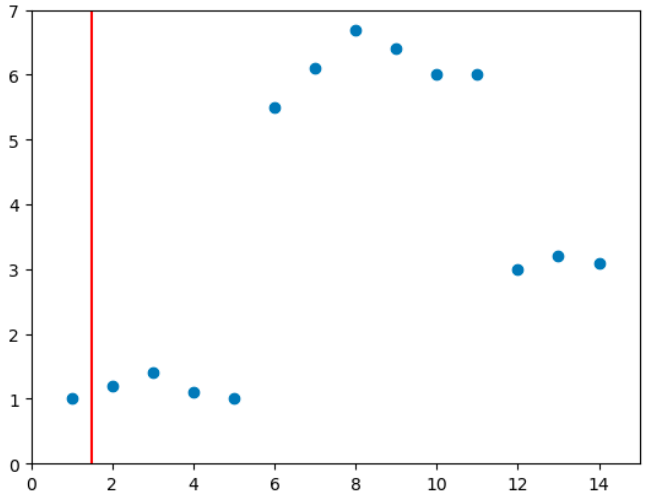
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_{true} - Y_{pred})^2$$

Ý tưởng chính của thuật toán là tìm điểm chia dữ liệu thành 2 phần tốt nhất sao cho MSE là nhỏ nhất, quá trình này được thực hiện liên tục trong quá trình tạo Tree.

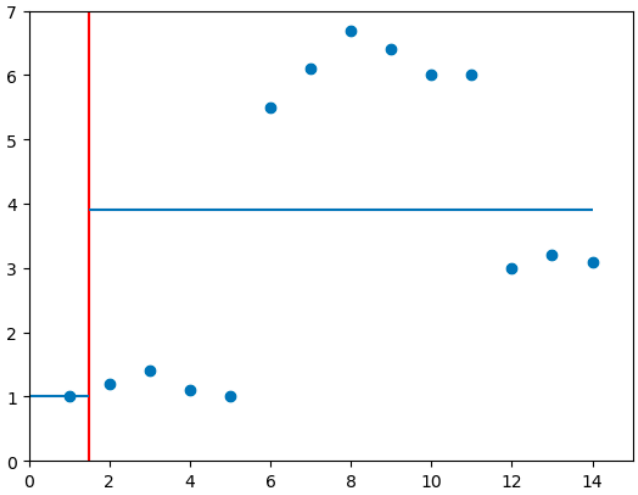
Giả sử ta có bộ dữ liệu sau:

	X	*Y*
0	1	1.0
1	2	1.2
2	3	1.4
3	4	1.1
4	5	1.0
5	6	5.5
6	7	6.1
7	8	6.7
8	9	6.4
9	10	6.0
10	11	6.0
11	12	3.0
12	13	3.2
13	14	3.1

- Bước 1: Sort dữ liệu theo X, sau đó tính giá trị trung bình của 2 điểm dữ liệu đầu tiên dựa theo X. Lúc này là (1 + 2) / 2 bằng 1.5, sau đó chia dữ liệu làm 2 phần dựa vào điểm trung bình đó.



- Bước 2: Tính giá trị trung bình của Y trong 2 phần đã được tách biệt. Hai giá trị trung bình này sẽ là giá trị dự đoán của Decision Tree tương ứng với $X < 1.5$ và $X \geq 1.5$. MSE được tính dựa trên giá trị trung bình và giá trị thực tế trong 2 phần dữ liệu. Tổng các MSE cho ta biết được loss của tree khi tách dữ liệu tại X = 1.5. Lặp lại quá trình trên tương tự với cặp điểm dữ liệu tiếp theo, ghi lại MSE từng cặp điểm dữ liệu.



- Bước 3: Giá trị X split mà tại đó có MSE nhỏ nhất sẽ là giá trị sẽ được sử dụng làm Root node. Trong trường hợp có nhiều feature thì sẽ chọn feature có MSE nhỏ nhất.

3.2 Ensemble Method

Ensemble Method là một cách tiếp cận phổ biến trong lĩnh vực Machine Learning nhằm cải thiện độ chính xác bằng cách kết hợp các kết quả dự đoán từ nhiều mô hình. Với ý tưởng ấy, có vẻ như sẽ có rất nhiều hướng tiếp cận để kết hợp nhiều mô hình, nhưng chủ yếu thuộc ba nhóm chính: **Bagging**, **Boosting** và **Stack**. Mỗi nhóm sẽ có cách tiệp cận khác nhau để trả về kết quả có mức độ hiệu quả khác nhau tùy vào bài toán mà chúng ta gặp phải, nhưng kết quả cuối cùng là sự kết hợp của nhiều mô hình chính là điểm chung của các phương pháp này. Trong phạm vi đồ án sẽ sử dụng 2 nhóm phổ biến nhất là **Bagging** và **Boosting**.

3.3 Bagging

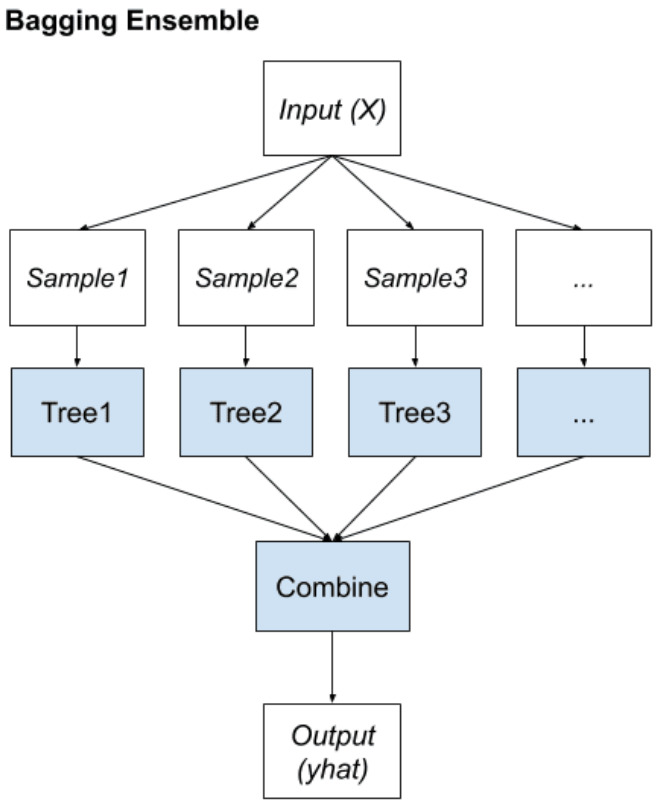
Boosting aggregation hay **Bagging** là phương pháp Ensemble tận dụng nhiều mô hình được huấn luyện trên các mẫu dữ liệu riêng trích xuất từ bộ dữ liệu huấn luyện.

Sử dụng một mô hình Machine Learning làm base model, thường là một Decision Tree. Huấn luyện mỗi mô hình trên các mẫu khác nhau của cùng một bộ dữ liệu. Kết quả dự đoán được tạo bởi các mô hình sau đó được kết hợp lại, bằng cách lấy giá trị trung bình.

Chìa khóa của Bagging là cách thức mà mỗi mẫu trong tập dữ liệu được chuẩn bị để huấn luyện các mô hình thành viên, mỗi mô hình được huấn luyện trên một mẫu khác nhau. Các mẫu dữ liệu được lấy sử dụng kĩ thuật "Sample with Replacement", khi một điểm dữ liệu đã được lấy, nó vẫn được trả về cho dữ liệu huấn luyện để chờ cơ hội tiếp theo được lấy. Có nghĩa là một điểm dữ liệu có thể không hoặc được lặp lại nhiều lần trong một lần lấy mẫu. Mục đích của phương pháp lấy mẫu này để hạn chế trường hợp các mô hình thành viên không gặp các mẫu huấn luyện giống nhau.

Trong thư viện scikit-learn, **BaggingRegressor** được cung cấp để huấn luyện theo ý tưởng trên, kết quả cuối cùng là giá trị trung bình của các mô hình thành viên.

Ngoài ra, có một mô hình cải tiến dựa trên cách lấy mẫu, các bước lấy mẫu hầu như tương tự, chỉ khác khi ở mỗi mẫu dữ liệu được lấy có n thuộc tính, chỉ huấn luyện mô hình ngẫu nhiên trên $k < n$ thuộc tính. Nhờ sự ngẫu nhiên này nên mô hình thường được biến đến với tên gọi là RandomForest, scikit learn có cung cấp **RandomForestRegressor** để huấn luyện mô hình này.

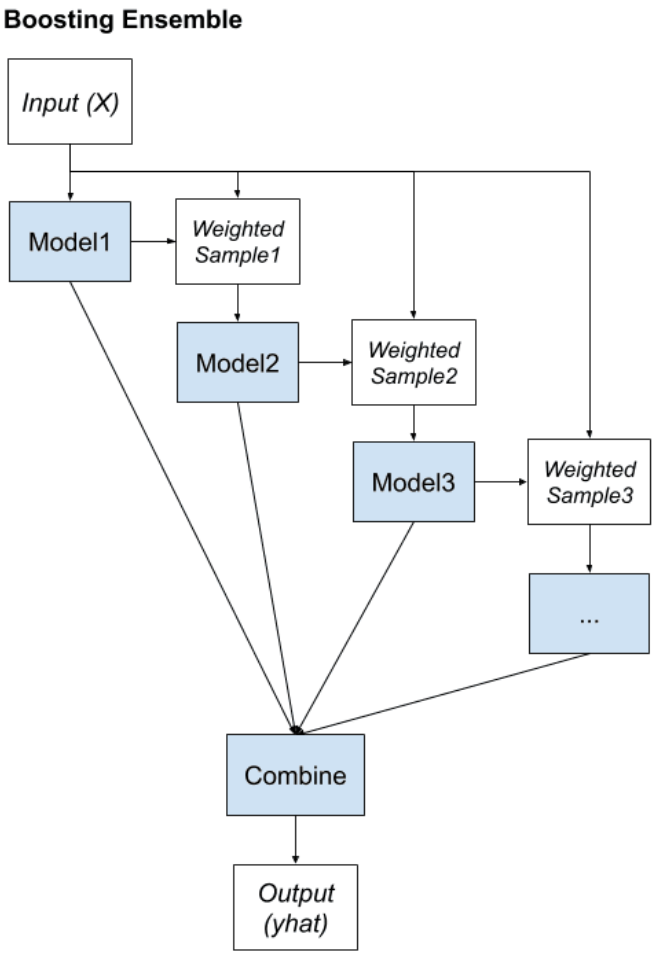


3.4 Boosting

Boosting là phương pháp Ensemble tìm cách thay đổi dữ liệu huấn luyện để mô hình tập trung vào các điểm dữ liệu mà trước đó đã dự đoán sai. Ý tưởng chính của Boosting là cố gắng khắc phục lỗi của các dự đoán sai. Các mô hình thành viên được huấn luyện tuần tự sao cho mô hình sau cố gắng khắc phục sai lầm của mô hình trước.

Base model là các Decision Tree thông thường rất đơn giản, đưa ra dự đoán từ rất ít các quyết định, gọi là các "weak learner", các dự đoán của "weak leaner" được kết hợp bằng cách lấy giá trị trung bình. Decision Trê có 1 node gốc và 2 node lá thường là "weak learner". Mục tiêu của phương pháp này là tạo ra một "strong learner" từ nhiều "weak learner".

Thông thường, dữ liệu huấn luyện không thay đổi, thay vào đó, các mô hình thành viên được huấn luyện để điều chỉnh tập trung nhiều hay ít vào các điểm dữ liệu dự đoán sai hay đúng bởi các mô hình thành viên trước. Mỗi điểm dữ liệu được thêm vào weight (trọng số), cho biết mức độ tập trung của mô hình trong quá trình học.



3.4.1 AdaBoost

AdaBoost là một trong những kĩ thuật Boosting đầu tiên, kể từ khi có AdaBoost, nhiều phương pháp Boosting được phát triển, có thể kể đến Gradient Boosting, XGBoost,… và là một trong những kĩ thuật hiệu quả nhất để cải thiện bài toán Classification và Regression.

Sử dụng **AdaBoostRegressor**, được cung cấp sẵn bởi scikit-learn để huấn luyện mô hình. Scikit-learn sử dụng AdaBoost.R2 – một biến thể của AdaBoost cho bài toán Regresion. Ý tưởng của thuật toán cũng giống trên, huấn luyện mô hình thành viên trong các mẫu dữ liệu, sau đó tính residual (lỗi giữa dự đoán và thực tế) để đánh giá chất lượng của "weak learner". Cập nhật weight dựa vào chất lượng (β^t) và độ chính xác (Ln) của "weak learner".

Sau quá trình học, chúng ta hình thành nhiều mô hình. Kết quả đự đoán sử dụng Weighted Median với trọng số $\log(\frac{1}{\beta^t})$ cho mỗi $t \in [1, T]$ với T là số lượng "weak learner".

Weighted Median

Với tập hợp v={v_1, v_2, …, v_N} và tập hợp trọng số w={w_1, w_2, …, w_N}. Đảm bảo weight đã được normalize để có tổng bằng 1. Để tính Weight Median, sort w và v theo thứ tự tăng dần của v, gọi v_{sorted} = {$v^{(1)}$, $v^{(2)}$, …, $v^{(N)}$} và w_{sorted} = {$w^{(1)}$, $w^{(2)}$, …, $w^{(N)}$}. Khi đó, Weighted Median là giá trị $v^{(n)}$ tương ứng với $w^{(n)}$ đầu tiên sao cho tổng tích lũy của các w lớn hơn hoặc bằng 0.5.

Với N là số lượng dữ liệu, T là số lượng các "weak-learner", các bước thực hiện:

- Bước 1: Khởi tạo trọng số ban đầu là $w_n=\frac{1}{N}$ với $n \in [1, N]$, có nghĩa là trước khi huấn luyện, các điểm dữ liệu có mức độ được tập trung như nhau.

- Bước 2: Với mỗi "weak learner" $t \in [1, T]$
  - Tạo mẫu dữ liệu với kĩ thuật "Sample with Replacement" và với xác suất w_n^t.
  - Huấn luyện mô hình thành viên, kết quả dự đoán là $f^t(x_n)$ với mỗi điểm x_n.
  - Tính lỗi sai khác giữa các điểm dữ liệu dự đoán và thực tế:

$$D^t = \max\{|y_n - f^t(x_n)|\}$$

$$L_n^t = \frac{|y_n - f^t(x_n)|}{D^t}$$

  - Tính lỗi của mô hình thành viên thứ t là L^t:

$$L^t = \sum_{n=1}^{N} L_n^t w_n^t$$

Nếu $L^t \geq 0.5$ thì dừng lại và đặt T = t – 1

  - Gọi $\beta^t = \frac{L^t}{1-L^t}$, β^t càng thấp cho biết mô hình càng tốt.
  - Cập nhật trọng số theo công thức:

$$w_n^{t+1} = \frac{w_n^t (\beta^t)^{1-L_n^t}}{\sum_{n=1}^{N} w_n^t (\beta^t)^{1-L_n^t}}$$

- Bước 3: Kết quả dự đoán là Weighted Median kết quả dự đoán của các mô hình thành viên với trọng số là $\log(\frac{1}{\beta^t})$ cho mỗi mô hình t.

3.4.2 Gradient Boosting

Gradient Boosting là một biến thể của kĩ thuật Boosting với mục tiêu là giảm thiểu loss của mô hình bằng cách thêm các "weak learner" sử dụng Gradient Descent. Gradient Descent là một thuật toán tối ưu hóa sử dụng gradient để tìm điểm cực tiểu cho các hàm số khả vi. Bởi vì Gradient Boosting dựa trên ý tưởng giảm thiểu hàm loss, nhiều loại hàm loss khác nhau có thể áp dụng được, dẫn đến đây là một kĩ thuật có thể sử dụng linh hoạt cho nhiều bài toán, có thể áp dụng cho bài toán regression, classification,…

Gradient Boosting được huấn luyện bằng cách học trên dựa trên sự sai khác giữa giá trị dự đoán và thực tế, điều này cũng tương tự như tập trung thêm những mẫu bị dữ đoán sai. Các "weak learner" sẽ được thêm vào để tập trung khắc phục những dự đoán sai do các mô hình thành viên trước mắc phải. Sự đóng góp của các "weak learner" vào kết quả cuối cùng dựa trên quá trình tối ưu gradient trên toàn bộ loss.

Gradient Boosting được thiết kế để giải quyết nhiều bài toán, đồ án sẽ trình bày cấu hình **GradientBoostingRegressor** của scikit-learn cho bài toán Regression sử dụng Decision Tree.

- Bước 1: Khởi tạo kết quả mô hình là giá trị trung bình của output:

$$y_{pred} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

- Bước 2: Cho mỗi cây *m* trong số lượng cây *M*:
  - Tính pseudo-residual (lỗi giữa giá trị dự đoán và thực tế):

$$r_m = y_{true} - y_{pred}$$

  - Huấn luyện Decision Tree Regressor kết quả dự đoán mới lúc này là r_t.

- Cập nhật giá trị dự đoán:

$$y_{pred}^{t} = y_{pred}^{t-1} + learning_{rate} \times r_t$$

Thay vì như các cách tiếp cận trước là mô hình dự đoán output, Gradient Boosting dự đoán residual – tức là lỗi giữa dự đoán và thực tế, nhiệm vụ mô hình cố gắng tối ưu residual bằng cách thêm các cây con kế tiếp. Tham số *learning_rate* để giảm mức độ overfitting cho mô hình, tránh quá khớp tập dữ liệu huấn luyện, để có khả năng generalize với các input mới.

4 Các thực nghiệm

4.1 Chia dữ liệu huấn luyện – đánh giá

Huấn luyện mô hình và kiểm tra trên cùng một bộ dữ liệu là một sai lầm bởi vì: mô hình chỉ cố gắng lặp lại các nhãn của mẫu dữ liệu mà nó đã từng nhìn thấy sẽ có độ chính xác cao nhưng sẽ thiếu đi khả năng dự đoán trên các mẫu dữ liệu mới chưa từng nhìn thấy. Trường hợp này thường được biết đến với tên gọi là overfitting. Để tránh điều này xảy ra trong tác vụ supervised learning, thông thường một phần dữ liệu sẽ được giữ lại, gọi là Test set. Để đảm bảo đánh giá mô hình công bằng nhất, Test set được chia sẽ chỉ được đánh giá ở bước cuối cùng, để có thể thực sự phản ánh hiệu năng của mô hình nhất có thể.

Train – Test set được chia ra theo tỉ lệ 95 – 5, trong đó 95% lượng dữ liệu cho việc huấn luyện và chỉ 5% cho đánh giá hiệu năng mô hình. Vì số lượng mẫu dữ liệu khá lớn, lên đến hơn 20,000 mẫu nên chỉ trích ra rất ít để test (khoảng hơn 1000 mẫu), dành đa số dữ liệu để huấn luyện mô hình (xấp xỉ 20,300 mẫu).

Sau đó thực hiện scale các thuộc tính có giá trị liên tục và encode các giá trị rời rạc:

- **MaxAbsScaler** thuộc tính "YearsCodePro" vì thuộc tính này có các giá trị liên tục.
- **OnehotEncoder** cho "RemoteWork" để one-hot encoding vì thuộc tính này có các giá trị rời rạc và không có tính "thứ tự" giữa các giá trị của thuộc tính.
- **OrdinalEncoder** cho "EdLevel", "Country" và "Age" bởi vì các thuộc tính rời rạc này có tính "thứ tự" giữa các giá trị thuộc tính (VD: "Bachelor" < "Master").

4.2 Cross validation

Trong quá trình Hyperparameter tuning, vẫn sẽ có rủi ro về overfitting trên Test set vì các tham số được điều chỉnh để mô hình dự đoán tốt trên tập dữ liệu này. Khi ấy, những "tri thức" về của Test set vô tình để lộ cho mô hình, lúc đó sử dụng tập test để đánh giá khả năng nhìn thấy những dữ liệu mới sẽ không còn hiệu quả.

Để giải quyết vấn đề này, một phần của Train set được tách ra, gọi là Validation set: Huấn luyện mô hình trên Train set, đánh giá hiệu năng của mô hình trên Validation set, và sau khi các thực nghiệm có vẻ thành công thì đánh giá cuối cùng trên Test set. Tuy nhiên, do chia dữ liệu ra thành 3 phần có thể giảm số lượng mẫu để huấn luyện, ngoài ra kết quả của mô hình có thể bị dao động bởi sự lựa chọn ngẫu nhiên ấy.

Cross validation là một giải pháp được đề ra để khắc phục vấn đề này. Test set nên được tách ra để đánh giá cuối cùng, nhưng Validation set không cần thiết chuẩn bị để thực hiện Cross Validaiton. Cách tiếp cận của K-Fold Cross Validation thực hiện điều đó bằng cách, chia Train set thành K set nhỏ hơn, quá trình sau đây được thực hiện trên mỗi K fold:

- Mô hình được huấn luyện sử dụng k-1 fold làm Train set.
- Đánh giá kết quả trên fold còn lại, đóng vai trò là Validaiton set.

Kết quả đánh giá mô hình sau khi thực hiện K Fold Cross Validation sau đó được tính trung bình cộng. Quá trình này có thể tiêu tốn nhiều tài nguyên tính toán, nhưng nó giúp không phải tốn quá nhiều dữ liệu, ngoài ra có thể hạn chế được tình trạng ngẫu nhiên khi chọn mẫu.

4.3 Kết quả

Thực hiện đánh giá Cross Validation với K Fold với K bằng 5, thống nhất quá trình tiền xử lí giống nhau để đánh giá mô hình công bằng nhất. Đánh giá mô hình trên cả ba độ đo, lần lượt là Mean Square Error, Mean Absolute Error và R^2-score.

Các Ensemnble Model đều sử dụng Decision Tree làm base model, số lượng mô hình đều bằng 200 để công bằng so sánh các mô hình. Kết quả của các độ đo ở đây đều là tính trung bình trong 5 Fold. Các siêu tham số còn lại mặc định theo scikit-learn.

Model	**RMSE**	**MAE**	**R^2-score**
Decision Tree Regressor	56206.58	38946.41	0.14
BaggingRegressor	38911.11	27427.89	0.5897
RandomForest Regressor	38917.28	27427.93	0.5896
AdaBoostRegressor	38764.97	26913.78	0.5928
GradientBoostingRegressor	**37730.76**	**26009.15**	**0.6142**

Từ bảng kết quả có thể thấy dữ liệu nhìn chung khá phức tạp, hầu như các mô hình đều không có hiệu quả chính xác quá cao. Tuy nhiên, so sánh kết quả của Decision Tree với các mô hình Ensemble cho ta thấy được rằng sự kết hợp của nhiều mô hình lại với nhau rõ ràng có giúp tăng hiệu quả dự đoán rất nhiều, sự khác biệt ở đây rất lớn, lên đến **0.4** trên thang đo R^2-score.

Giữa 2 kĩ thuật Bagging và Boosting, với bộ dữ liệu này, kĩ thuật Boosting rõ ràng tỏ ra hiệu quả hơn. Khi so sánh R^2-score, Boosting giúp kết quả tăng lên **0.01** so với Bagging, thậm chí lên đến **0.03** khi sử dụng Gradient Boosting. Không những thế, Gradient Boosting thậm chí còn giúp làm giảm MSE, MAE hơn đáng kể.

Mô hình Gradient Boosting tỏ ra vượt trội hơn đáng kể so với các mô hình khác, sử dụng mô hình Gradient Boosting làm mô hình chính.

4.4 Điều chỉnh siêu tham số

Siêu tham số là các tham số không được học trong quá trình huấn luyện mô hình, trong scikit-learn thì nó chính là các tham số được thêm vào Contructor. Scikit-learn cung cấp Gradient Boosting với rất nhiều sự lựa chọn về tham số, chẳng hạn như Loss function, leanring rate, số lượng estimator,… Sử dụng **GridSearchCV** để tìm ra các siêu tham số có kết quả Cross Validate cao nhất.

Về cơ bản nhất, GridSearchCV sẽ tìm ra các siêu tham số sao cho đạt độ chính xác trên Cross Validation cao nhất. Nó sẽ Brute Force thử từng tham số, huấn luyện và đánh giá mô hình. Nhìn chung rất tốn kém tài nguyên tính toán, nhưng kết quả vô cùng ấn tượng.

Các siêu tham số được sử dụng để GridSearchCV:

- **Loss**: loss function được sử dụng để tối ưu bài toán regression, các lựa chọn bao gồm **squared_error**, **absolute_error**, **huber** và **quantile**.
- **Learning rate:** giúp thu hẹp đóng góp của các cây sau, giảm overfitting.
- **N estimator:** số lượng cây.
- **Criterion:** Hàm dùng để ước tính hiệu quả của node điều kiện đặt khi tách cây, các lựa chọn gồm có **friedman_mse**, **squaredd_error**.

Kết quả GridSearch đạt được cao nhất trả về các tham số:

- Loss: huber
- Learning rate: 0.1
- N estimator: 400
- Criterion: friedman_mse

Sử dụng các tham số trên để huấn luyện mô hình cuối trên toàn bộ Train set, sau đó đánh giá kết quả cuối cùng trên Test set đạt kết quả:

Metrics	**Values**
RMSE	37068.786
MAE	25121.021
R^2-score	0.619

Đây là kết quả cuối cùng tốt nhất đạt được trên bộ dữ liệu này, sử dụng mô hình Gradient Boosting. Sử dụng mô hình này để tạo web demo, chi tiết xem thêm tại github.

5 Nguồn tài liệu tham khảo

- https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/
- https://machinelearningcoban.com/2018/01/14/id3/
- https://medium.com/analytics-vidhya/regression-trees-decision-tree-for-regression-machine-learning-e4d7525d8047
- https://dafriedman97.github.io/mlbook/content/c6/s1/boosting.html
- https://maelfabien.github.io/machinelearning/GradientBoost/#full-pseudo-code
- https://www.mygreatlearning.com/blog/gradient-boosting/#extreme-gradient-boosting-xgboost
- https://blog.paperspace.com/adaboost-optimizer/
- https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502
- https://scikit-learn.org/stable/modules/ensemble.html

6 Source Code

Đồ án có thêm web demo để có thể dễ dàng tương tác với mô hình, cũng như có thêm phần visualize để có thể dễ dàng quan sát dữ liệu hơn.

Chi tiết cách thực hiện tại file README.md.

Github: https://github.com/tien02/salary_pred_stack_overflow_2022