

MỤC LỤC

- Giới thiệu
 - 1. Tại sao cần chiến lược Học Máy
 - 2. Cách sử dụng cuốn sách khi làm việc nhóm
 - 3. Kiến thức tiền đề và Ký hiệu
 - 4. Quy mô là động lực phát triển học máy
- Phần 1: Chuẩn bị tập phát triển và tập kiểm tra
 - 5. Tập phát triển và tập kiểm tra
 - 6. Tập phát triển và tập kiểm tra nên có cùng phân phối
 - 7. Tập phát triển/kiểm tra cần lớn đến mức nào?
 - 8. Thiết lập một phép đo đơn trị làm mục tiêu tối ưu
 - 9. Phép đo tối ưu và phép đo thỏa mãn
 - 10. Xây dựng một tập phát triển và một phép đo sẽ tăng tốc quá trình làm việc
 - 11. Khi nào cần thay đổi tập phát triển/kiểm tra và các phép đo
 - 12. Điều cần nhớ: Thiết lập các tập phát triển và kiểm tra
- Phần 2: Phân tích lỗi cơ bản
 - 13. Bạn mong muốn xây dựng một hệ thống phòng chống email rác mới. Nhóm của bạn có rất nhiều ý tưởng:
 - 14. Phân tích lỗi: đánh giá ý tưởng dựa trên tập phát triển
 - 15. Đánh giá song song các ý tưởng trong quá trình phân tích lỗi
 - 16. Dọn dẹp những mẫu bị gán nhãn nhầm trong tập phát triển và tập kiểm tra
 - 17. Nếu bạn có một tập phát triển lớn, chia nó thành hai tập con và chỉ phân tích trên một tập
 - 18. Tập phát triển Eyeball và Blackbox nên lớn như thế nào?
 - 19. Điều cần nhớ: Phân tích lỗi cơ bản
- Phần 3: Độ chêch và Phương sai
 - 20. Độ chêch và Phương sai: Hai nguồn lớn của lỗi
 - 21. Những ví dụ về Độ chêch và Phương sai
 - 22. So sánh với tỉ lệ lỗi tối ưu
 - 23. Xử lý Độ chêch và Phương sai
 - 24. Sự đánh đổi giữa Độ chêch và Phương sai
 - 25. Các kĩ thuật để giảm độ chêch có thể tránh được
 - 26. Phân tích lỗi trên tập huấn luyện
 - 27. Các kĩ thuật làm giảm phương sai
- Phần 4: Đồ thị quá trình học
 - 28. Chẩn đoán độ chêch và phương sai: Đồ thị quá trình học
 - 29. Vẽ đồ thị sai số huấn luyện
 - 30. Diễn giải đồ thị quá trình học: Độ chêch cao
 - 31. Giải nghĩa các đồ thị quá trình học: Những trường hợp khác
 - 32. Vẽ đồ thị quá trình học
- Phần 5: So sánh với chất lượng mức con người
 - 33. Tại sao chúng ta so sánh với chất lượng mức con người?
 - 34. Cách xác định chất lượng mức con người
 - 35. Vượt qua chất lượng mức con người
- Phần 6: Huấn luyện và kiểm tra trên các phân phối khác nhau
 - 36. Khi nào bạn nên huấn luyện và kiểm tra trên những phân phối khác nhau
 - 37. Làm sao để quyết định có nên sử dụng toàn bộ dữ liệu?
 - 38. Làm thế nào để quyết định có nên bao gồm dữ liệu không nhất quán
 - 39. Đánh trọng số dữ liệu
 - 40. Tổng quát hóa từ tập huấn luyện đến tập phát triển
 - 41. Xác định những lỗi về độ chêch, phương sai, và dữ liệu không tương đồng
 - 42. Xử lý dữ liệu không tương đồng
 - 43. Tổng hợp dữ liệu nhân tạo

- Phần 7: Gỡ lỗi các Thuật toán suy luận
 - 44. Bài kiểm tra xác minh tối ưu
 - 45. Dạng tổng quát của bài kiểm tra xác minh tối ưu
 - 46. Ví dụ về Học tăng cường
- Phần 8: Học sâu đầu-cuối
 - 47. Sự trỗi dậy của học đầu-cuối
 - 48. Thêm những ví dụ về học đầu-cuối.
 - 49. Ưu nhược điểm của học đầu-cuối
 - 50. Lựa chọn các thành phần cho pipeline: Tính sẵn có của dữ liệu
 - 51. Lựa chọn các thành phần cho pipeline: tính đơn giản của tác vụ
 - 52. Trực tiếp học những đầu ra phức tạp
- Phần 9: Phân tích lỗi từng phần
 - 53. Phân tích lỗi từng phần
 - 54. Quy lỗi cho một thành phần
 - 55. Trường hợp tổng quát của việc quy lỗi
 - 56. Phân tích lỗi từng phần và so sánh với chất lượng mức con người
 - 57. Phát hiện một pipeline học máy bị lỗi
- Phần 10: Tổng kết
 - 58. Xây dựng một biệt đội siêu anh hùng - Hãy để đồng đội của bạn đọc điều này

Giới thiệu

1. Tại sao cần chiến lược Học Máy

Học Máy là nền tảng cho hàng loạt ứng dụng quan trọng như tìm kiếm trang web, lọc thư điện tử spam, nhận dạng giọng nói, gợi ý sản phẩm, và nhiều ứng dụng khác nữa. Nếu bạn cùng các thành viên trong nhóm đang làm một dự án học máy và rất muốn tiến triển nhanh chóng, thì quyển sách này là dành cho bạn.

Ví dụ: Xây dựng Startup về ảnh mèo

Giả sử bạn xây dựng công ty khởi nghiệp cung cấp không giới hạn ảnh mèo cho những người yêu thích.



Bạn dùng mạng nơ-ron cho hệ thống thị giác máy nhằm phát hiện mèo trong ảnh. Nhưng dở một cái là thuật toán bạn dùng chưa đủ độ chính xác. Bạn đang chịu rất nhiều áp lực để tăng chất lượng bộ phát hiện mèo. Bạn sẽ làm thế nào?

Nhóm bạn có thể đưa ra rất nhiều ý tưởng như:

- Lấy thêm dữ liệu: Sưu tầm thêm nhiều ảnh mèo.
- Lấy tập huấn luyện đa dạng hơn. Ví dụ như: ảnh mèo ở vị trí độc lạ, ảnh mèo với màu sắc khác thường, ảnh mèo được chụp với cấu hình máy ảnh khác nhau .v.v.
- Huấn luyện thuật toán lâu hơn bằng cách chạy thêm nhiều vòng lặp hạ gradient.
- Thủ nghiệm mạng nơ-ron lớn hơn với nhiều tầng/nút ẩn/tham số hơn.
- Thủ nghiệm mạng nơ-ron nhỏ hơn.
- Thủ nghiệm kỹ thuật regularization (ví dụ như L2 regularization)
- Thay đổi kiến trúc mạng nơ-ron (ví dụ: hàm kích hoạt, số lượng nút ẩn, .v.v)
- ...

Nếu chọn đúng một trong những hướng kể trên, có thể bạn sẽ xây dựng nên một nền tảng ảnh mèo và startup thành công. Ngược lại, nếu chọn nhầm hướng, bạn có thể đánh mất cả tháng trời. Vậy phải làm như thế nào?

Cuốn sách này sẽ giúp bạn trả lời câu hỏi đó. Phần lớn các vấn đề về học máy đều có những dấu hiệu riêng ẩn chứa gợi ý về phương hướng giải quyết. Việc học để phát hiện ra những dấu hiệu đó sẽ giúp bạn tiết kiệm hàng tháng hay thậm chí hàng năm trời phát triển sản phẩm.

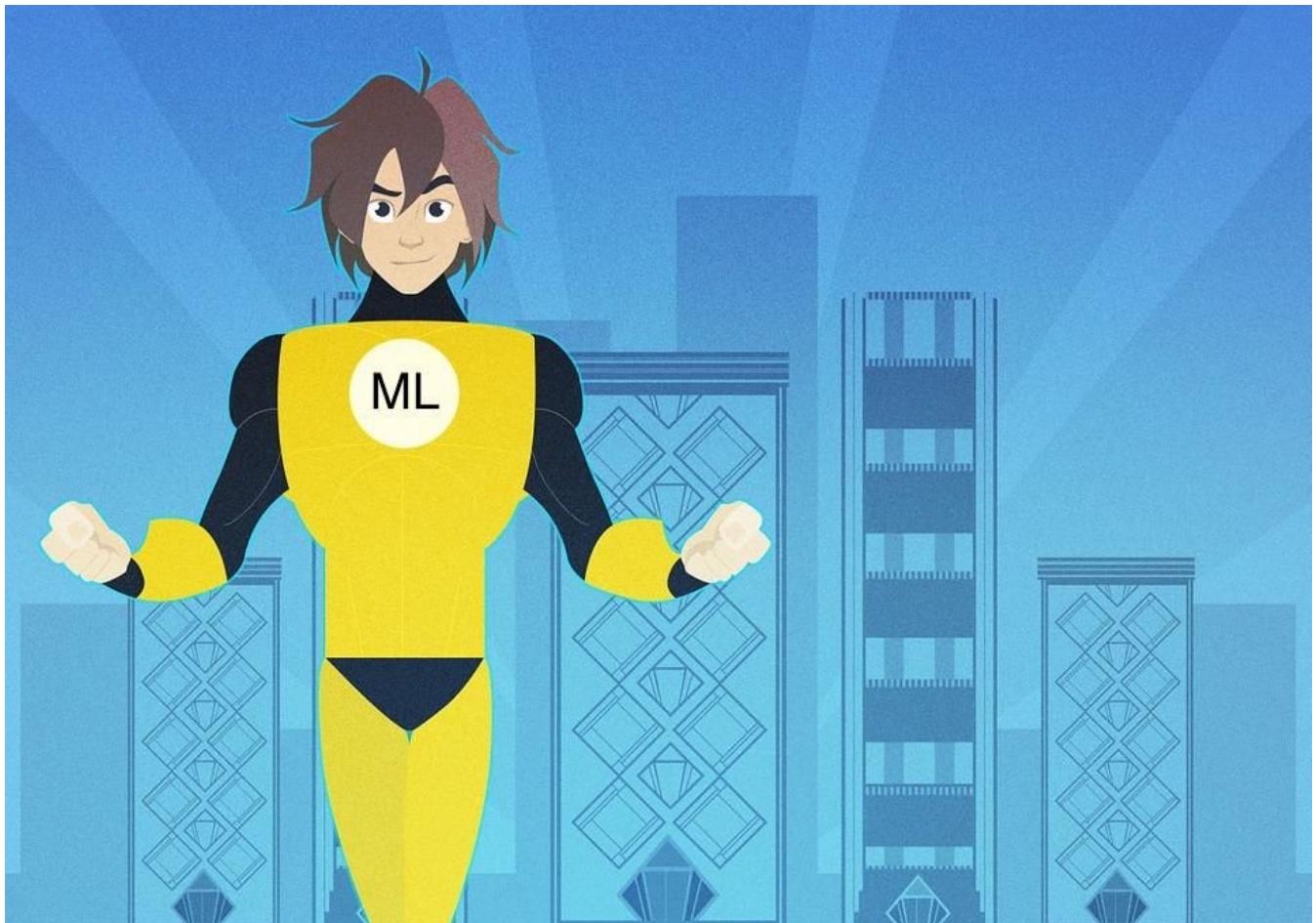
2. Cách sử dụng cuốn sách khi làm việc nhóm

Sau khi đọc xong cuốn sách này, bạn sẽ hiểu sâu hơn về cách lựa chọn hướng giải quyết kỹ thuật cho đề tài học máy.

Nhưng có thể cộng sự chưa rõ tại sao bạn lại chọn hướng đi như vậy. Ví dụ bạn muốn cả đội xác định và dùng một phép đo đơn trị, nhưng nếu mọi người không đồng tình, thì bạn sẽ làm gì để thuyết phục họ?

Đó là lý do tôi chủ tâm viết những chương rất ngắn. Bạn có thể dễ dàng thuyết phục quý đồng nghiệp bằng cách chia sẻ 1-2 trang của chương liên quan.

Chỉ với một vài thay đổi nhỏ về thứ tự ưu tiên có thể tác động lớn tới năng suất công việc của cả nhóm. Và bằng những thay đổi đó, tôi hi vọng bạn sẽ sớm trở thành siêu nhân Học Máy của cả đội!



3. Kiến thức tiền đề và Ký hiệu

Nếu bạn đã từng học một lớp về Học Máy, ví dụ như lớp MOOC của tôi trên Coursera, hoặc bạn có kinh nghiệm áp dụng học có giám sát thì cuốn sách này sẽ dễ hiểu đối với bạn.

Tôi giả định rằng bạn đã quen thuộc với **học có giám sát** học một hàm ánh xạ từ x tới y , sử dụng các cặp dữ liệu có nhãn (x, y). Các thuật toán học có giám sát bao gồm hồi quy tuyến tính, hồi quy logistic và mạng nơ-ron. Học Máy có rất nhiều dạng tuy nhiên phần lớn các giá trị thực tiễn của nó hiện nay đến từ học có giám sát.

Tôi sẽ thường xuyên đề cập đến mạng nơ-ron (còn được biết đến là "học sâu"). Bạn chỉ cần nắm được một số khái niệm cơ bản về mạng nơ-ron là có thể hiểu được nội dung cuốn sách.

Nếu những khái niệm new trên còn mới với bạn thì bạn hãy xem các video ba tuần đầu tiên của khóa học Machine Learning trên Coursera tại <http://ml-class.org>

The screenshot shows the Coursera website with the URL 'coursera.org' in the address bar. The main page header includes 'Explore', 'Log In', and 'Join for Free'. Below the header, there's a search bar with the placeholder 'What do you want to learn?'. The main content area displays the 'Machine Learning' course by Stanford. The course title is 'Machine Learning', it has a rating of 4.9 from 116,957 ratings, and 28,701 reviews. It is offered for free starting on Oct 28. There are 2,589,174 already enrolled. Below the course summary, there are links for 'About', 'Syllabus', 'Reviews', 'Instructors', 'Enrollment Options', and 'FAQ'. To the right, there's a sidebar titled 'Learners taking this Course are' which lists roles: Technical Leads, Software Engineers, Machine Learning Engineers, Risk Managers, and Chief Technology Officers (CTOs). There are also sections for '100% online' and 'Flexible deadlines'.

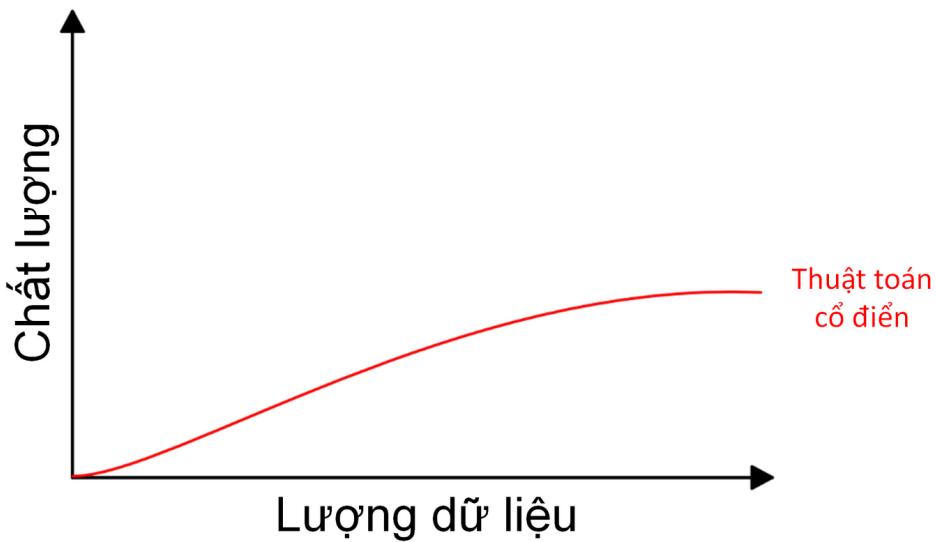
4. Quy mô là động lực phát triển học máy

Rất nhiều những ý tưởng của học sâu (mạng nơ-ron) đã xuất hiện từ hàng thập kỷ trước. Vậy tại sao tới bây giờ chúng mới bùng nổ như vậy?

Hai nguyên nhân chính là:

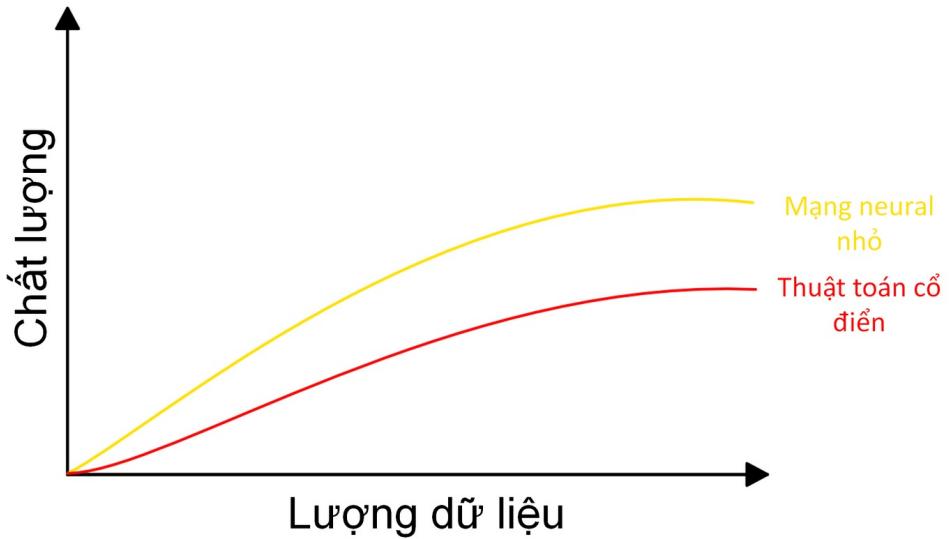
- **Sự sẵn có của dữ liệu.** Ngày nay, mọi người dành nhiều thời gian hơn bên những thiết bị số như máy tính xách tay, thiết bị di động, .v.v. Việc này tạo ra nguồn dữ liệu cực lớn dùng cho những thuật toán học máy.
- **Quy mô năng lực tính toán.** Chỉ tới một vài năm gần đây ta mới có thể huấn luyện mạng nơ-ron đủ lớn để tận dụng những bộ dữ liệu khổng lồ này.

Cho dù có thêm nhiều dữ liệu nữa, thường thì chất lượng của các thuật toán học máy cổ điển, như hồi quy logistic, cũng không tốt hơn. Nghĩa là đồ thị quá trình học chững lại và thuật toán ngừng cải thiện ngay cả khi có thêm dữ liệu:

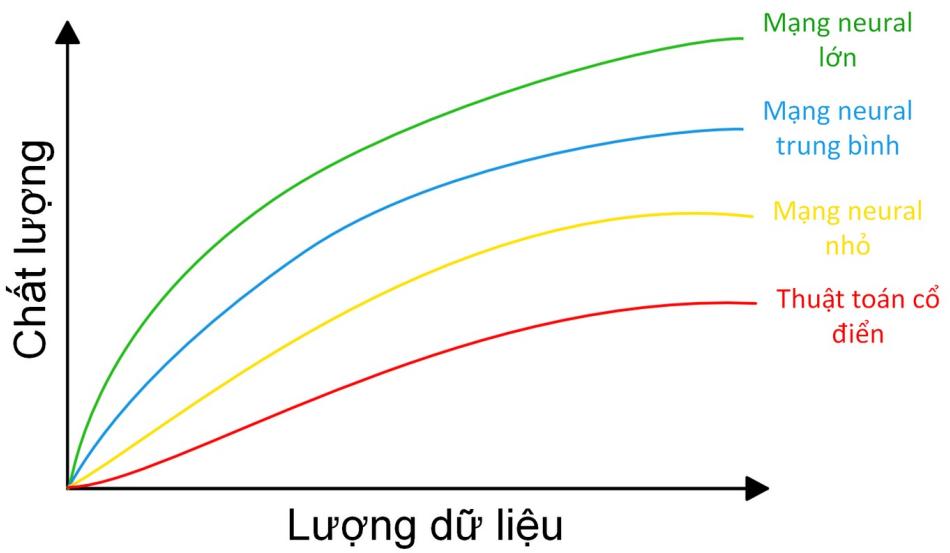


Như thế thuật toán cổ điển không biết xử lý thế nào với tất cả lượng dữ liệu ta đang có.

Nếu bạn huấn luyện một mạng nơ-ron nhỏ cho cùng một tác vụ học có giám sát, bạn có thể đạt chất lượng cao hơn một chút:



"Mạng neural nhỏ" ở đây có nghĩa là mạng nơ-ron với ít nút ẩn/tầng/tham số. Sau cùng, bạn có thể cải thiện chất lượng thêm nữa nếu dùng các mạng nơ-ron lớn hơn [1]:



Như vậy bạn đạt được chất lượng tốt nhất khi (i) huấn luyện mạng nơ-ron rất lớn -- tương ứng với đường chất lượng màu xanh lục và (ii) có lượng dữ liệu lớn.

Nhiều chi tiết khác như kiến trúc mạng nơ-ron cũng rất quan trọng, và có nhiều phát kiến trong lĩnh vực này. Tuy nhiên, một trong những cách đáng tin cậy hơn để tăng chất lượng thuật toán vẫn là (i) huấn luyện mạng lớn hơn và (ii) lấy thêm dữ liệu.

CHÚ THÍCH:

[1] Mặc dù hình vẽ thể hiện mạng nơ-ron cho kết quả tốt hơn với tập dữ liệu nhỏ, hiện tượng này ít nhất quán so với việc mạng nơ-ron hoạt động tốt với dữ liệu lớn. Với dữ liệu nhỏ, chất lượng thuật toán cổ điển có thể tốt hoặc kém hơn mạng nơ-ron và phụ thuộc vào các đặc trưng thủ công. Nếu ta chỉ có 20 mẫu huấn luyện thì việc dùng hồi quy logistic hay mạng nơ-ron không khác biệt nhiều; chọn khéo các đặc trưng thủ công sẽ giúp ích nhiều hơn so với việc chọn thuật toán. Còn nếu có một triệu mẫu, thì tôi sẽ chọn dùng mạng nơ-ron.

Để đạt được (i) và (ii) là một quá trình đặc biệt phức tạp. Vấn đề này sẽ được thảo luận đầy đủ và chi tiết trong cuốn sách này. Ta sẽ bắt đầu với các chiến lược thông thường và hữu ích cho cả thuật toán truyền thống lẫn mạng nơ-ron, từ đó hình thành các chiến lược tân tiến nhất để xây dựng các hệ thống học sâu.

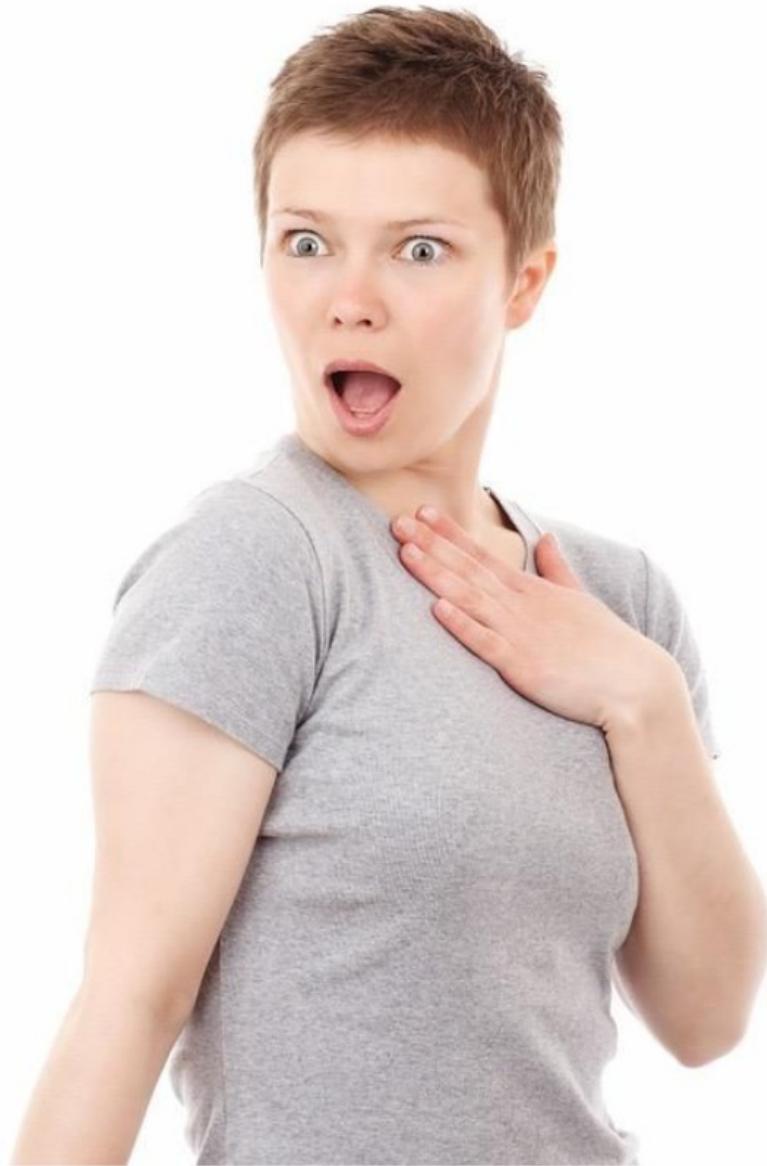
Phần 1: Chuẩn bị tập phát triển và tập kiểm tra

5. Tập phát triển và tập kiểm tra

Trở lại với ví dụ bức ảnh mèo ở phần trước: khi bạn có một ứng dụng di động, và người dùng đang tải nhiều loại ảnh lên ứng dụng của bạn. Bạn muốn tự động tìm ra đâu là các bức ảnh mèo.

Nhóm của bạn có một tập dữ liệu lớn bằng cách tải các bức ảnh mèo (các mẫu dương) và các bức ảnh không có mèo (các mẫu âm) từ nhiều nguồn khác nhau. Tập dữ liệu này sau đó được chia 70%/30% thành tập huấn luyện và tập kiểm tra. Sử dụng tập dữ liệu này, bạn tạo ra một bộ nhận dạng mèo có thể hoạt động tốt ở cả tập huấn luyện và tập kiểm tra.

Tuy nhiên, khi triển khai bộ nhận dạng mèo này lên một ứng dụng di động, bạn lại thấy rằng chất lượng rất tệ!



Điều gì đã xảy ra?

Bạn nhận ra rằng các bức ảnh được người dùng tải lên nhìn khác các bức ảnh mà bạn tải về từ trên mạng mà được dùng để xây dựng tập dữ liệu: do các bức ảnh được chụp bằng điện thoại thường có độ phân giải thấp hơn, bị nhòe (mờ) hoặc tối hơn. Do bộ nhận dạng được huấn luyện trên tập dữ liệu từ ảnh trên mạng nên nó không khai quát hóa tốt đến phân phối thực tế mà bạn cần nhắm đến: ảnh chụp từ điện thoại.

Trước kỷ nguyên big data, có một nguyên tắc chung trong machine learning là chia tập huấn luyện và kiểm tra ngẫu nhiên theo tỉ lệ 70%/30%. Cách chia này có thể hiệu quả, nhưng không phải là một ý tưởng tốt trong ngày càng nhiều ứng dụng nơi mà phân phối của tập huấn luyện (ảnh trên mạng trong ví dụ trên đây) khác với phân phối của dữ liệu bạn thực sự quan tâm (ảnh chụp từ điện thoại).

Chúng ta thường định nghĩa như sau:

- **Tập huấn luyện** — Là tập dữ liệu để chạy thuật toán học.

- **Tập phát triển** — Là tập dữ liệu được dùng để điều chỉnh lại các tham số, lựa chọn đặc trưng và quyết định các thay đổi liên quan đến thuật toán học. Đôi khi, nó còn được gọi là tập kiểm định chéo.
- **Tập kiểm tra** — Là tập dữ liệu dùng để đánh giá chất lượng của thuật toán học, nhưng không được dùng để quyết định các thay đổi liên quan đến thuật toán hay các tham số.

Sau khi định nghĩa tập phát triển và tập kiểm tra, nhóm của bạn có thể thử nhiều ý tưởng khác nhau, ví dụ như các tham số khác nhau cho thuật toán học, để tìm ra ý tưởng tốt nhất. Tập phát triển và tập kiểm tra cho phép nhóm của bạn có thể đánh giá khả năng hoạt động của thuật toán một cách nhanh chóng.

Nói cách khác, **mục đích của tập phát triển và tập kiểm tra là hướng nhóm bạn tới những thay đổi quan trọng nhất trong hệ thống học máy.**

Vậy nên, bạn nên làm những điều sau đây:

Lựa chọn tập phát triển và tập kiểm tra sao cho có thể phản ánh dữ liệu bạn gặp phải trong tương lai và muốn hoạt động tốt trên nó.

Nói cách khác, tập kiểm tra không nên chỉ đơn thuần là 30% dữ liệu hiện có, đặc biệt là khi bạn mong đợi dữ liệu tương lai (ảnh chụp từ điện thoại) về bản chất sẽ khác với dữ liệu trong tập huấn luyện (ảnh từ trên mạng).

Khi ứng dụng di động chưa được triển khai thì bạn có thể chưa có người dùng nào cả, nên việc có thể có được dữ liệu phản ánh chính xác dữ liệu tương lai là rất khó. Nhưng bạn vẫn có thể thử làm gần giống dữ liệu đó. Ví dụ, bạn có thể nhờ bạn bè chụp những bức ảnh mèo bằng điện thoại và gửi cho bạn. Một khi ứng dụng được triển khai, bạn có thể cập nhật tập phát triển/kiểm tra bằng dữ liệu người dùng thực tế.

Nếu bạn thực sự không có cách nào để có được dữ liệu gần giống với dữ liệu tương lai thì bạn có thể sử dụng ảnh từ các trang web. Nhưng bạn nên nhận thức nguy cơ dẫn đến một hệ thống khái quát hóa không tốt.

Ta cần thẩm định để quyết định được cần phải tập trung bao nhiêu cho việc phát triển tập phát triển và tập kiểm tra. Tuy nhiên đừng đánh đồng phân phối giữa tập huấn luyện và tập kiểm tra. Hãy chọn ra những mẫu kiểm tra phản ánh cái mà bạn muốn thực hiện tốt, hơn là bất kì dữ liệu nào bạn tình cờ có được cho tập huấn luyện.

6. Tập phát triển và tập kiểm tra nên có cùng phân phối



Bạn có thể chia tập dữ liệu của ứng dụng ảnh mèo dựa theo bốn thị trường chính: (i) Hoa Kỳ, (ii) Trung Quốc, (iii) Ấn Độ, và (iv) Khu vực khác. Chúng ta có thể lấy dữ liệu từ Hoa Kỳ và Ấn Độ làm tập phát triển trong khi lấy Trung Quốc và Khu vực khác làm tập kiểm tra. Hay nói theo cách khác, liệu việc chọn ngẫu nhiên dữ liệu ảnh từ hai trong bốn khu vực trên làm tập phát triển và hai khu vực còn lại làm tập kiểm tra có đúng hay không?

Một khi định nghĩa được tập phát triển và tập kiểm tra, nhóm của bạn sẽ tập trung cải thiện chất lượng trên tập phát triển. Bởi vậy, tập phát triển cần phản ánh tác vụ bạn muốn cải thiện nhiều nhất đó là: hoạt động tốt trên không chỉ hai mà cả bốn thị trường.

Nếu tập phát triển và tập kiểm tra có phân phối khác nhau, bạn có thể đổi mặt vấn đề thứ hai: Có khả năng nhóm của bạn sẽ xây dựng ra thuật toán nào đó hoạt động tốt trên tập phát triển nhưng lại kém trên tập kiểm tra. Tôi đã từng thấy việc này dẫn đến những hệ quả gây thất vọng và lãng phí công sức. Hãy cố gắng tránh để điều này xảy ra.

Ví dụ như nhóm của bạn phát triển một hệ thống hoạt động tốt trên tập phát triển nhưng kém trên tập kiểm tra. Nếu tập phát triển và tập kiểm tra có cùng một phân phối, thì bạn có thể xác định ngay vấn đề: Mô hình của bạn đã overfit tập phát triển. Cách xử lý hiển nhiên nhất đó là bổ sung thêm dữ liệu cho tập phát triển.

Nhưng nếu tập phát triển và tập kiểm tra có phân phối khác nhau, thì việc xác định vấn đề sẽ phức tạp hơn. Rất nhiều vấn đề có thể xảy ra như:

- a. Bạn đã overfit tập phát triển.
- b. Tập kiểm tra khó hơn tập phát triển. Vì thế thuật toán của bạn có thể đã hoạt động tốt hết mức có thể và không thể có thiện thêm nữa.
- c. Tập kiểm tra không nhất thiết khó hơn, nhưng lại khác biệt so với tập phát triển. Do đó, việc thuật toán hoạt động tốt trên tập phát triển và kém trên tập kiểm tra là dễ hiểu. Trong trường hợp này, việc cố gắng cải thiện hiệu quả trên tập phát triển có thể trở nên vô nghĩa.

Làm việc với các ứng dụng học máy vốn dĩ đã khó. Việc không nhất quán giữa tập phát triển và kiểm tra khiến bạn càng khó chắc chắn về "liệu cải thiện chất lượng trên tập phát triển có đồng nghĩa với tăng chất lượng trên tập kiểm tra hay không?". Việc không đồng nhất giữa tập phát triển và tập kiểm tra khiến việc xác định những kỹ thuật giúp cải tiến chất lượng khó khăn hơn từ đó khó xếp thứ tự ưu tiên của tác vụ.

Nếu bạn đang làm việc thông qua một đánh giá xếp hạng của bên thứ ba, họ có thể đã chỉ ra là tập phát triển và tập kiểm tra không có cùng phân phối. So sánh với bài toán có tập phát triển và tập kiểm tra có cùng phân phối, thì chất lượng thuật toán của bạn trên tập đánh giá xếp hạng kia phụ thuộc nhiều vào may mắn hơn là kỹ năng. Việc phát triển thuật toán học mà được huấn luyện trên một phân phối này mà có khái quát hóa tốt trên một phân phối khác là một chủ đề nghiên cứu quan trọng. Tuy nhiên, nếu mục tiêu của bạn là cải tiến một ứng dụng học máy cụ thể thay vì làm nghiên cứu, thì tôi khuyên bạn chọn tập phát triển và tập kiểm tra có cùng phân phối. Điều này sẽ khiến nhóm bạn làm việc hiệu quả hơn.

7. Tập phát triển/kiểm tra cần lớn đến mức nào?

Tập phát triển phải đủ lớn để nhận ra sự khác biệt giữa các thuật toán đang thử nghiệm. Ví dụ, nếu bộ phân loại A có độ chính xác 90,0% và bộ phân loại B có độ chính xác 90,1%, thì một tập phát triển có 100 mẫu sẽ không thể phát hiện sự khác biệt 0,1% này. So với các vấn đề khác trong học máy mà tôi đã thấy, một tập phát triển chỉ với 100 mẫu là nhỏ. Các tập phát triển thường có từ 1.000 tới 10.000 mẫu. Với 10.000 mẫu, bạn sẽ có thể thấy sự cải thiện ở mức 0,1%. [2]

Trong các ứng dụng quan trọng và đã đã đưa vào khai thác -- ví dụ như quảng cáo, tìm kiếm trên web và gợi ý sản phẩm -- tôi đã thấy nhiều nhóm rất muốn cải thiện chất lượng thuật toán dù chỉ là 0,01%, vì nó có ảnh hưởng trực tiếp đến lợi nhuận của công ty. Trong trường hợp này, tập phát triển có thể lớn hơn 10.000 mẫu rất nhiều để có thể phát hiện ra những cải tiến thậm chí nhỏ hơn.

Vậy còn kích thước của tập kiểm tra thì sao? Nó cần đủ lớn để mang lại độ tin cậy cao về chất lượng tổng thể của hệ thống. Một công thức thực nghiệm phổ biến là sử dụng 30% dữ liệu làm tập kiểm tra. Cách làm này hiệu quả với những tập dữ liệu với lượng mẫu khiêm tốn từ 100 tới 10.000. Tuy nhiên, trong kỷ nguyên big data với những bài toán học máy đòi hỏi có nhiều hơn một tỷ mẫu, tỉ lệ dữ liệu dùng cho tập phát triển và tập kiểm tra đã giảm xuống đáng kể, mặc dù số lượng mẫu trong hai tập này vẫn tăng lên. Thực sự không cần có tập phát triển/kiểm tra lớn quá mức để đánh giá chất lượng của các thuật toán.

CHÚ THÍCH:

[2] Trên lý thuyết, ta cũng có thể kiểm tra xem một thay đổi trong thuật toán có tạo ra sự khác biệt có ý nghĩa thống kê trên tập phát triển hay không. Trong thực tế, hầu hết mọi người đều không quan tâm đến điều này (trừ khi họ muốn công bố các bài báo khoa học). Tôi thường thấy các bài kiểm định thống kê không mấy hữu ích trong việc đánh giá tiến độ phát triển.

8. Thiết lập một phép đo đơn trị làm mục tiêu tối ưu

Độ chính xác trong phân loại là ví dụ của **phép đo đơn trị** -- phép đo được biểu diễn bằng chỉ một con số. Khi chạy bộ phân loại trên một tập phát triển (hoặc tập kiểm tra), độ chính xác được tính bằng chỉ số thể hiện tỉ lệ mẫu được phân loại chính xác trên tổng số mẫu trong tập đó. Theo phép đo này, nếu độ chính xác của bộ phân loại A là 97% và của bộ phân loại B là 90% thì ta kết luận rằng bộ phân loại A cho kết quả tốt hơn.

Ngược lại, Precision và Recall[3] không phải là một phép đo đơn trị: có hai chỉ số được sử dụng để đánh giá bộ phân loại. Việc so sánh các thuật toán với nhau sẽ trở nên khó hơn với những phép đo đa trị -- những phép đo được biểu diễn bằng nhiều hơn một số. Giả sử thuật toán trả về kết quả như sau:

Ở đây, không bộ phân loại nào tốt hơn một cách rõ ràng, vì vậy dựa vào kết quả trên ta không thể ngay lập tức chọn ra một bộ phân loại tốt hơn.

Bộ Phân Loại	Precision	Recall
A	95%	90%
B	98%	85%

Trong quá trình phát triển, nhóm bạn sẽ thử rất nhiều ý tưởng liên quan đến cấu trúc thuật toán, tham số mô hình, lựa chọn các đặc trưng, v.v.. Việc có một **phép đo đơn trị** như độ chính xác sẽ giúp xếp hạng các mô hình dựa theo những chất lượng trả về qua phép đo đó, từ đó nhanh chóng quyết định mô hình nào hoạt động tốt nhất.

Nếu bạn thực sự quan tâm đến cả Precision lẫn Recall. Tôi gợi ý sử dụng một trong những cách tiêu chuẩn để kết hợp các chỉ số đó thành một chỉ số duy nhất. Ví dụ, một người có thể lấy giá trị trung bình của Precision và Recall rồi thu về một phép đo đơn trị. Hoặc thay vào đó, bạn có thể tính "chỉ số F1", một biến thể của trung bình cộng, thường hoạt động tốt hơn việc chỉ lấy giá trị trung bình.

Việc có một phép đo đơn trị sẽ giúp tăng tốc khả năng đưa ra quyết định của bạn khi bạn phải lựa chọn trong một số lượng lớn bộ phân loại. Phép đo đơn trị đưa ra ưu tiên rõ ràng trong việc phân hạng những thuật toán đó, tạo ra những đường hướng rõ ràng để phát triển.

Bộ Phân Loại	Precision	Recall	Chỉ số F1
A	95%	90%	92.4%
B	98%	85%	91.0%

Một ví dụ cuối cùng, giả sử bạn đang theo dõi riêng biệt về độ chính xác của bộ phân loại mèo trong bốn thị trường trọng điểm: (i) Mĩ, (ii) Trung Quốc, (iii) Ấn Độ, và (iv) những nước khác. Bạn sẽ thu về bốn phép đo. Bằng cách lấy giá trị trung bình hoặc giá trị trung bình có trọng số của bốn chỉ số này, bạn sẽ thu được một phép đo đơn trị. Tính toán giá trị trung bình hoặc giá trị trung bình có trọng số là một trong những cách phổ biến nhất để kết hợp nhiều phép đo thành một.

CHÚ THÍCH:

[3] Precision của một bộ phân loại mèo là tỉ lệ những ảnh được phân nhãn chính xác là mèo trong tập phát triển (hoặc tập kiểm tra) trên tổng số những ảnh được bộ phân loại phân nhãn mèo trong cùng tập đó. Recall của bộ phân loại đó là số phần trăm của tất cả ảnh mèo ở trong tập phát triển (hoặc tập kiểm tra) được phân loại chính xác là mèo trong cùng tập đó. Thường có một sự đánh đổi giữa việc có chỉ số precision cao và chỉ số recall cao.

[4] Nếu bạn muốn đọc thêm về chỉ số F1, xem https://en.wikipedia.org/wiki/F1_score. Chỉ số F1 là trung bình điều hoà của Precision và Recall, được tính bằng $2/(1/Precision) + (1/Recall)$

9. Phép đo tối ưu và phép đo thỏa mãn

Đây là một cách khác để kết hợp nhiều phép đo.

Giả sử bạn quan tâm đến cả độ chính xác lẫn thời gian chạy của một thuật toán học nào đó. Bạn cần phải chọn trong ba bộ phân loại sau:

Bộ phân loại	Độ chính xác	Thời gian chạy
A	90%	80ms
B	92%	95ms
C	95%	1,500ms

Việc tạo ra một phép đo đơn trị bằng cách đưa cả độ chính xác và thời gian chạy vào trong một công thức có vẻ không tự nhiên, ví dụ như:

Độ chính xác - 0.5*(Thời gian chạy)

Thay vào đó, bạn có thể làm như sau: Trước hết định nghĩa thế nào là một mốc thời gian chạy "chấp nhận được". Giả sử mốc dưới 100ms là chấp nhận được. Sau đó, hãy cực đại hóa độ chính xác, với ràng buộc là bộ phân loại đó vẫn đảm bảo yêu cầu về thời gian chạy. Ở đây, thời gian chạy là một "phép đo thỏa mãn" -- bộ phân loại của bạn chỉ cần "đủ tốt" về mặt này (thời gian), theo nghĩa nó chỉ được phép chạy trong thời gian ít hơn 100ms. Độ chính xác mới là "phép đo tối ưu".

Nếu bạn phải cân bằng giữa N tiêu chí khác nhau, ví dụ như kích thước file nhị phân của mô hình (điều này quan trọng với các ứng dụng di động, vì người dùng không muốn tải về những ứng dụng có kích thước lớn), thời gian chạy, và độ chính xác, bạn có thể cân nhắc đặt N-1 trong số các tiêu chí là các phép đo "thỏa mãn". Có nghĩa là bạn chỉ cần yêu cầu chúng đạt giá trị nào đó. Sau đó coi tiêu chí còn lại là phép đo "tối ưu". Ví dụ như đặt mức ngưỡng chấp nhận được cho kích thước file nhị phân và thời gian chạy, sau đó tối ưu độ chính xác với điều kiện các ràng buộc trên vẫn được thỏa mãn.

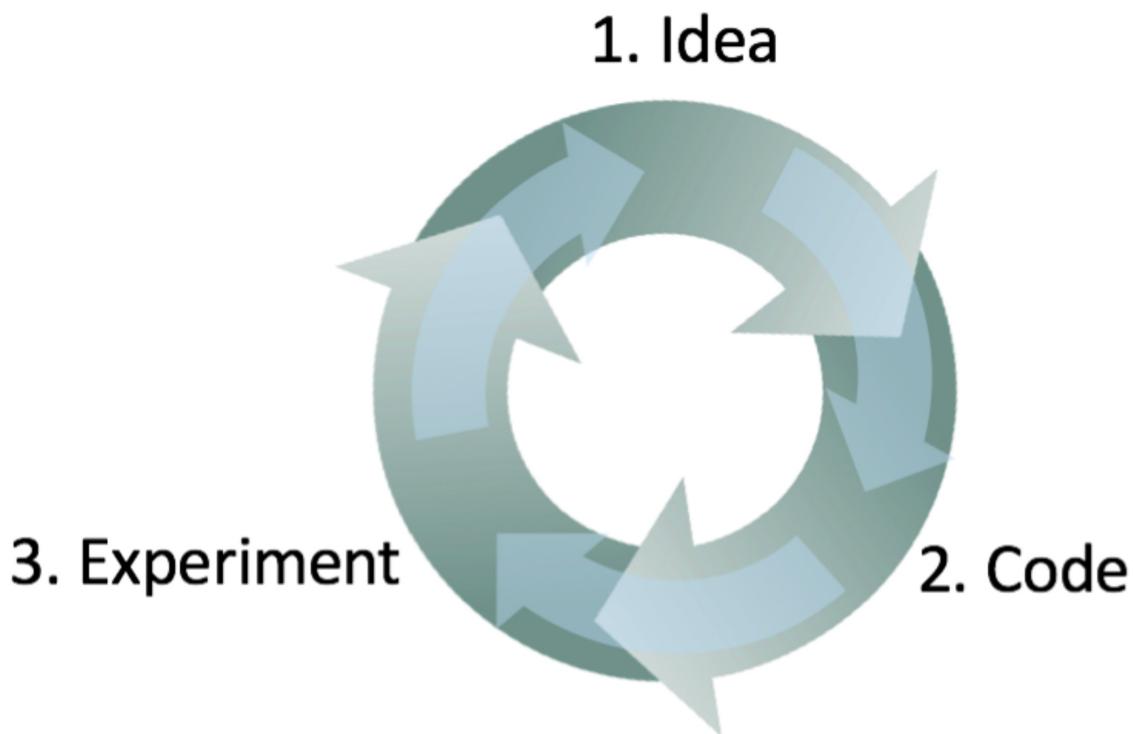
Ví dụ cuối cùng, giả sử bạn cần xây dựng một thiết bị phần cứng có sử dụng microphone để nghe người dùng nói một từ "đánh thức" đặc biệt nào đó để đánh thức hệ thống. Ví dụ về từ đánh thức như Amazon Echo với "Alexa"; Apple Siri với "Hey Siri"; Android với "Hey Google" hay các ứng dụng của Baidu với "Hello Baidu". Bạn quan tâm đến cả tần suất dương tính giả (hay báo động nhầm) -- tần suất mà hệ thống thức dậy khi không ai nói cụm đánh thức -- cũng như tần suất âm tính giả (hay bỏ sót) -- tần suất hệ thống không thức dậy khi có người nói cụm đánh thức. Một mục tiêu khả dĩ cho hệ thống này là tối thiểu hóa tần suất âm tính giả (phép đo tối ưu), trong ràng buộc rằng không có nhiều hơn một báo động nhầm cho mỗi 24 giờ hoạt động (phép đo thỏa mãn).

Một khi nhóm của bạn thống nhất về việc phép đo nào cần được tối ưu, cả nhóm sẽ đạt tiến độ nhanh hơn.

10. Xây dựng một tập phát triển và một phép đo sẽ tăng tốc quá trình làm việc

Thật sự rất khó để đoán trước phương án tiếp cận nào tốt nhất cho một vấn đề mới. Kể cả những nhà nghiên cứu học máy dày dặn kinh nghiệm cũng thường thử nghiệm cả chục ý tưởng mới khám phá ra cái gì đó thỏa mãn. Khi xây dựng một hệ thống học máy, tôi thường:

- a. Bắt đầu với một vài ý tưởng xây dựng hệ thống.
- b. Hiện thực hóa ý tưởng dưới dạng code.
- c. Tiến hành một **thí nghiệm** cho đo tính khả thi của ý tưởng. (Thường thì một số ý tưởng đầu tiên sẽ không khả thi!) Từ những kết quả đó, chúng ta quay lại thử nghiệm thêm những ý tưởng mới và cứ thế lặp lại.



Đây là một quá trình lặp đi lặp lại. Hoàn thiện vòng lặp càng nhanh, thì càng sớm cải thiện kết quả. Đó là lý do tại sao có tập phát triển/thử nghiệm và một phép đo là rất quan trọng: Việc đánh giá chất lượng của mỗi ý tưởng trên tập phát triển giúp xác định liệu chúng ta có đi đúng hướng.

Ngược lại, giả sử bạn không có một tập phát triển và phép đo cụ thể. Như vậy mỗi khi nhóm của bạn phát triển một bộ phân loại mèo mới, bạn sẽ phải tích hợp nó vào ứng dụng, và ngõi thử nghiệm ứng dụng đó một vài tiếng để kiểm tra liệu bộ phân loại mới có cải thiện hay không. Quá trình này sẽ cực kì chậm! Đồng thời, nhóm của bạn sẽ rất khó nhận ra sự khác biệt nếu độ chính xác chỉ cải thiện từ 95.0% lên 95.1%, bạn sẽ không thể phát hiện sự cải thiện 0.1% đó chỉ qua việc ngõi thử nghiệm trên ứng dụng. Và hệ thống sau cùng là tích lũy của rất nhiều bước cải thiện nhỏ 0.1%. Có một tập phát triển và phép đo cho phép bạn nhanh chóng phát hiện ra ý tưởng nào sẽ đem lại những cải tiến nhỏ (hoặc lớn), và từ đó bạn có thể quyết định những ý tưởng nào cần hoàn thiện thêm hoặc loại bỏ.

11. Khi nào cần thay đổi tập phát triển/kiểm tra và các phép đo

Khi bắt đầu một dự án, tôi luôn cố gắng chọn tập phát triển/kiểm tra thật nhanh để tạo một mục tiêu rõ ràng cho cả nhóm.

Tôi thường yêu cầu các nhóm của tôi xác định tập phát triển/kiểm tra và một phép đo ban đầu trong ít hơn một tuần, rất hiếm khi lâu hơn. Tốt hơn hết là có được những điều này, kể cả chưa hoàn hảo, và bắt đầu nhanh chóng hơn là suy nghĩ quá nhiều về chúng. Tuy nhiên, thời hạn một tuần không áp dụng với các ứng dụng đã phát triển. Ví dụ, chống thư rác là một ứng dụng học sâu đã phát triển. Tôi từng thấy những nhóm làm việc với những hệ thống đã phát triển dành hàng tháng để tạo được những tập phát triển/kiểm tra tốt hơn.

Nếu sau đó bạn nhận ra rằng tập phát triển/kiểm tra hoặc phép đo ban đầu không phù hợp với mục tiêu đặt ra, bằng mọi giá hãy thay đổi chúng một cách nhanh chóng. Chẳng hạn, nếu bộ phân loại A được đánh giá tốt hơn bộ phân loại B theo phép đo trên tập phát triển ban đầu, nhưng nhóm nghĩ rằng bộ phân loại B thực ra cho kết quả tốt hơn nhiều trên sản phẩm, điều này có thể là dấu hiệu cho thấy bạn cần thay đổi tập phát triển/kiểm tra hoặc phép đo.

Có ba nguyên nhân chính khiến việc tập phát triển/phép đo sai sót trong việc đánh giá bộ phân loại A cao hơn:

1. Phân phối thực tế mà bạn cần làm tốt khác với phân phối của tập phát triển/kiểm tra.



Giả sử tập phát triển/kiểm tra ban đầu chủ yếu có ảnh mèo trưởng thành. Sau khi chạy ứng dụng, bạn nhận ra rằng thành viên thường tái lên nhiều ảnh mèo con hơn dự tính. Khi đó, phân phối của tập phát triển/kiểm tra không đại diện cho phân phối thực tế mà cần bạn hướng tới. Trong trường hợp này, bạn cần cập nhật tập phát triển/kiểm tra sao cho chúng có tính đại diện hơn.

2. Mô hình của bạn đã overfit tập phát triển.

Quá trình lặp đi lặp lại việc đánh giá những ý tưởng trên tập phát triển khiến thuật toán dần "overfit" tập dữ liệu này. Sau quá trình phát triển, bạn sẽ đánh giá mô hình trên tập kiểm tra. Nếu bạn thấy rằng chất lượng trên tập phát triển tốt hơn nhiều do với chất lượng trên tập kiểm tra, đây là dấu hiệu bạn đã overfit tập phát triển. Trong trường hợp này, bạn hãy tạo một tập phát triển hoàn toàn mới.

Nếu bạn cần theo dõi tiến trình của nhóm, bạn cũng có thể đánh giá hệ thống thường xuyên -- chẳng hạn mỗi tuần hoặc mỗi tháng một lần -- trên tập kiểm tra. Tuy nhiên, không được sử dụng tập kiểm tra để ra quyết định thay đổi thuật toán, bao gồm việc quay lui về hệ thống trước đó. Nếu bạn làm vậy, bạn sẽ bắt đầu overfit tập kiểm tra, và không thể tiếp tục dựa vào nó để tạo ra một đánh giá hoàn toàn không thiên lệch cho chất lượng của hệ thống (đánh giá này bạn sẽ cần nếu bạn xuất bản công trình nghiên cứu, hoặc có thể sử dụng phép đo này để ra những quyết định quan trọng trong kinh doanh).

3. Phép đo không phù hợp với mục tiêu tối ưu của dự án.

Giả sử trong ứng dụng mèo, phép đo của bạn là độ chính xác phân loại. Phép đo này hiện tại xếp hạng bộ phân loại A tốt hơn bộ phân loại B. Tuy nhiên, giả sử bạn thử cả hai thuật toán, và nhận ra rằng bộ phân loại A thi thoảng chấp nhận những bức ảnh khiêu dâm. Ngay cả khi bộ phân loại A chính xác hơn, ấn tượng xấu gây ra bởi một vài bức ảnh khiêu dâm đồng nghĩa với việc chất lượng của nó là không chấp nhận được. Bạn sẽ làm gì?

Ở đây, phép đo thất bại trong việc xác định được thực tế Thuật toán B tốt hơn Thuật toán A trong sản phẩm của bạn. Bởi vậy, bạn không thể đặt niềm tin vào phép đo này để chọn thuật toán tốt nhất. Đây là lúc phải thay đổi phép đo. Ví dụ, bạn có thể thay đổi phép đo sao cho nó "phạt" thật nặng nếu một thuật toán chấp nhận ảnh khiêu dâm. Tôi khuyên bạn chọn một phép đo mới và sử dụng phép đo này để định nghĩa lại mục tiêu rõ ràng cho nhóm, hơn là tiếp tục chọn ra một cách thủ công trong số các bộ phân loại khi không có một phép đo đáng tin.

Việc thay đổi tập phát triển/kiểm tra hoặc phép đo trong một dự án là khá phổ biến. Có một tập phát triển/kiểm tra và phép đo ban đầu giúp bạn hoàn thành chu kỳ phát triển một cách nhanh chóng. Nếu bạn nhận ra rằng tập phát triển/kiểm tra hoặc phép đo không còn giúp nhóm đi đúng hướng, không sao cả! Chỉ cần thay chúng và đảm bảo nhóm biết về hướng đi mới.

12. Điều cần nhớ: Thiết lập các tập phát triển và kiểm tra

- Chọn tập phát triển và tập kiểm tra từ một phân phối phản ánh dữ liệu bạn dự tính nhận được trong tương lai và muốn hoạt động tốt trên nó. Phân phối này có thể không giống phân phối dữ liệu huấn luyện của bạn.
- Chọn tập phát triển và kiểm tra từ cùng một phân phối xác suất nếu có thể.
- Chọn một phép đo đơn trị để tối ưu hóa. Nếu có nhiều thông số cần quan tâm, hãy kết hợp chúng thành một công thức duy nhất (chẳng hạn như lấy trung bình của các phép đo) hoặc định nghĩa phép đo thỏa mãn và phép đo để tối ưu.
- Học máy là một quá trình lặp đi lặp lại: Bạn có thể phải thử hàng tá ý tưởng trước khi tìm thấy một ý tưởng mà bạn hài lòng.
- Có tập phát triển/kiểm tra và một phép đo đơn trị giúp bạn nhanh chóng đánh giá các thuật toán và do đó lặp lại nhanh hơn.
- Khi bắt đầu trên một ứng dụng hoàn toàn mới, cố gắng thiết lập tập phát triển/kiểm tra và một phép đo một cách nhanh chóng, trong vòng chưa đầy một tuần. Với các ứng dụng đã được phát triển, quá trình này có thể kéo dài hơn.
- Việc chia dữ liệu huấn luyện/kiểm tra theo tỉ lệ 70%/30% không áp dụng cho các bài toán với nhiều dữ liệu; tập phát triển và kiểm tra có thể chiếm ít hơn con số 30% rất nhiều.
- Tập phát triển của bạn phải đủ lớn để phát hiện các thay đổi có ý nghĩa đối với độ chính xác của thuật toán, nhưng không nhất thiết phải lớn hơn nhiều. Tập kiểm tra phải đủ lớn để cung cấp cho bạn ước lượng đáng tin cậy về chất lượng cuối cùng của hệ thống.
- Nếu tập phát triển và phép đo không còn chỉ cho nhóm của bạn đi đúng hướng, hãy nhanh chóng thay đổi chúng: (i) Nếu thuật toán đã overfit tập phát triển, hãy thu thập thêm dữ liệu cho tập này. (ii) Nếu phân phối xác suất thực tế mà bạn quan tâm khác với phân phối xác suất của tập phát triển/kiểm tra, hãy tạo tập phát triển và kiểm tra mới. (iii) Nếu phép đo không còn đo lường được điều quan trọng nhất với bạn, hãy thay đổi phép đo.

Phần 2: Phân tích lỗi cơ bản

13. Bạn mong muốn xây dựng một hệ thống phòng chống email rác mới. Nhóm của bạn có rất nhiều ý tưởng:

Thu thập một tập huấn luyện lớn về email rác. Ví dụ như thiết lập một Honeypot (Mồi nhử): cố ý gửi các địa chỉ email giả đến những spammer đã biết, và bạn có thể thu thập các tin nhắn rác mà họ gửi đến địa chỉ đó một cách tự động.

Phát triển những tính năng để hiểu được nội dung văn bản trong email.

Phát triển những tính năng để hiểu được các đặc tính của phông thư/nhân thư từ email nhằm hiển thị tập hợp các máy chủ internet mà thư đã đi qua.

- và nhiều hơn thế.

Mặc dù tôi đã kinh qua rất nhiều trong việc phòng chống email rác, tôi vẫn sẽ gặp khó khăn khi phải chọn một trong các hướng đi trên. Điều này sẽ còn khó hơn nếu bạn không phải là một chuyên gia trong lĩnh vực này.

Vì vậy, bạn không nên bắt đầu bằng việc thiết kế và xây dựng một hệ thống hoàn hảo. Thay vào đó, hãy xây dựng và huấn luyện nhanh một hệ thống cơ bản -- có thể là trong vài ngày[5]. Ngay cả khi hệ thống cơ bản khác xa với hệ thống tốt nhất mà bạn có thể xây dựng, nó vẫn có giá trị để kiểm tra cách thức hoạt động của hệ thống cơ bản này: bạn sẽ nhanh chóng tìm ra được những dấu hiệu sẽ chỉ cho bạn những hướng đi hứa hẹn nhất để đầu tư thời gian của bạn. Trong những chương tiếp theo sẽ chỉ cho bạn cách tìm thấy những dấu hiệu này.



CHÚ THÍCH:

Lời khuyên này dành cho những độc giả có mong muốn xây dựng các ứng dụng AI, hơn là những người có mục tiêu là xuất bản những bài báo học thuật. Tôi sẽ quay trở lại với chủ đề nghiên cứu này sau.

14. Phân tích lỗi: đánh giá ý tưởng dựa trên tập phát triển



Khi kiểm thử ứng dụng nhận dạng mèo, bạn thấy rằng một số bức ảnh chó bị nhận nhầm. Nhìn chúng tương đối giống mèo!

Một thành viên trong nhóm đề xuất tích hợp vào hệ thống phần mềm của bên thứ ba. Việc kết hợp này có thể giúp hệ thống phân biệt tốt hơn các bức ảnh chó. Có thể mất một tháng để hoàn thành quá trình tích hợp và người đề xuất ý tưởng rất hào hứng. Liệu bạn có nên yêu cầu thành viên đó bắt đầu công việc?

Trước khi bỏ ra một tháng thực hiện, bạn nên ước lượng công việc này có thể cải thiện độ chính xác của hệ thống tới mức nào. Từ đó, bạn sẽ có thể quyết định xem có đáng bỏ ra chừng đó thời gian vào việc phát triển hay là dành nó cho những việc khác.

Cụ thể, bạn có thể làm theo các bước sau:

- Thu thập 100 mẫu trong tập phát triển mà ứng dụng của bạn phân loại nhầm — không phải mèo nhưng được phân loại là mèo và ngược lại.
- Nhìn vào những mẫu trên và đếm xem bao nhiêu trong số đó là ảnh chó.

Quá trình nhìn vào những mẫu bị phân loại nhầm được gọi là **phân tích lỗi**. Trong ví dụ này, nếu bạn nhận thấy rằng chỉ 5% lỗi là chó nhầm thành mèo thì cho dù cải thiện thuật toán theo hướng tích hợp phần mềm nhận dạng chó vào ứng dụng, bạn không thể loại bỏ quá 5% số ảnh bị nhận dạng sai. Nói cách khác, 5% là "cận trên" (số lượng tối đa có thể đạt được) cho mức độ cải thiện mà hướng đi trên có thể giúp cho hệ thống. Nếu như độ chính xác ban đầu của ứng dụng là 90% (10% lỗi), việc cải thiện chỉ làm cho hệ thống của bạn đạt được độ chính xác mới là 90.5% (9.5% lỗi, ít hơn 5% so với số lượng lỗi ban đầu).

Ngược lại, nếu bạn nhận thấy rằng 50% lỗi là do chó bị nhầm thành mèo thì bạn có thể tự tin rằng phương án được đề xuất sẽ có tác động lớn. Nó có thể cải thiện đáng kể độ chính xác của hệ thống từ 90% lên 95% (giảm 50% tổng số lỗi, từ 10% xuống 5%).

Cách phân tích lỗi đơn giản ở trên giúp bạn ước lượng nhanh hiệu quả của việc tích hợp phần mềm nhận dạng chó của bên thứ ba vào hệ thống nhận dạng mèo. Đây cũng là cơ sở định lượng để bạn lựa chọn xem có nên đi theo hướng này hay không.

Việc **phân tích lỗi** thường giúp bạn nhìn thấy được triển vọng của những hướng giải quyết khác nhau. Tôi thấy nhiều kỹ sư tiến hành phân tích lỗi một cách miễn cưỡng. Dường như đối với họ ngay lập tức thực hiện một số ý tưởng sẽ thú vị hơn là tự hỏi xem ý tưởng đó có thật sự đáng để bạn bỏ thời gian thực hiện. Đây là một lỗi phổ biến: nó có thể gây lãng phí hàng tháng chỉ để nhận ra rằng sự cải thiện là không đáng kể.

Quan sát 100 mẫu để phân tích lỗi không tốn nhiều thời gian. Kể cả khi bạn bỏ ra một phút để kiểm tra từng ảnh, thời gian tổng cộng vẫn nhỏ hơn hai giờ. Nếu như ý tưởng kia không tốt, bỏ ra hai giờ phân tích lỗi này có thể giúp bạn tiết kiệm được một tháng.

Việc phân tích lỗi là quá trình kiểm tra các mẫu trong tập phát triển bị phân loại nhầm, từ đó bạn có thể hiểu được nguyên nhân. Hiểu rõ nguyên nhân tạo ra lỗi sẽ giúp bạn nhìn ra những hướng giải quyết mới mà chúng ta sẽ thảo luận ở phần sau. Một số chương tiếp theo sẽ trình bày những "best practices" được dùng để phân tích lỗi.

15. Đánh giá song song các ý tưởng trong quá trình phân tích lỗi

Nhóm của bạn có một số ý tưởng cải thiện ứng dụng nhận dạng mèo:

- Sửa lỗi nhận dạng chó thành mèo trong thuật toán.
- Sửa lỗi nhận dạng thú họ mèo (sư tử, báo, v.v) thành mèo nhà (thú nuôi).
- Cải thiện chất lượng của hệ thống trên ảnh mờ.
- ...

Bạn có thể đánh giá song song tất cả các ý kiến trên một cách hiệu quả. Tôi thường tạo một bảng và điền vào đó khi phân tích ~100 ảnh phân loại nhầm trong tập phát triển. Tôi cũng ghi chú ngắn gọn để ghi nhớ những trường hợp đặc biệt. Để minh họa cho quá trình này, bạn có thể tham khảo bảng được tạo ra từ một tập phát triển nhỏ với bốn mẫu dưới đây:

Ảnh	Chó	Thú Họ Mèo	Ảnh Mờ	Ghi chú
1	✓			Chó pitbull có màu lạ
2			✓	
3		✓	✓	Ảnh sư tử chụp ở sở thú trong một ngày mưa
4		✓		Một con báo bị khuất sau cây
Tổng %	25%	50%	50%	

Bảng #3 ở trên có cả hai cột Thú Họ Mèo và Ảnh Mờ được đánh dấu.Thêm vào đó, bởi vì một mẫu có thể nằm ở nhiều hạng mục, tổng phần trăm của hàng cuối có thể không đạt 100%.

Mặc dù bạn có thể tạo các hạng mục (Chó, Thú Họ Mèo, Ảnh Mờ) từ trước và sau đó phân loại các mẫu thủ công, thực tế trong quá trình phân tích mẫu, bạn có thể có những ý tưởng mới để tạo thêm các hạng mục. Ví dụ: bạn phân loại hàng chục bức ảnh và nhận ra nhiều lỗi xảy ra ở những tấm ảnh chỉnh bởi bộ lọc Instagram. Bạn có thể quay lại và thêm cột "Instagram" vào bảng. Bằng cách nhìn vào từng mẫu mà thuật toán phân loại nhầm và đặt câu hỏi làm thế nào/liệu rằng con người có thể nhận dạng mẫu này một cách chính xác, nhiều khả năng là bạn sẽ tìm được các hạng mục lỗi và giải pháp mới.

Những hạng mục lỗi hữu ích nhất sẽ là những lỗi mà bạn có thể khắc phục. Ví dụ, hạng mục Instagram sẽ là hữu ích nhất để thêm vào nếu bạn biết cách "đảo ngược" bộ lọc Instagram và phục hồi ảnh gốc. Tuy nhiên bạn không nhất thiết phải giới hạn bản thân chỉ với những hạng mục mà bạn biết cách cải thiện; mục tiêu của quá trình này là xây dựng một góc nhìn rõ hơn về những đặc trưng tiềm năng mà bạn nên tập trung vào.

Phân tích lỗi là một quá trình lặp đi lặp lại. Đừng lo nếu bạn bắt đầu mà vẫn chưa nghĩ được hạng mục nào. Bạn sẽ có thêm ý tưởng về các hạng mục lỗi mới sau khi phân tích một vài tấm ảnh. Sau khi phân loại thủ công một số hình ảnh, bạn có thể nghĩ ra các hạng mục mới và đổi chiều lại các mẫu ảnh theo hạng mục mới đó.

Giả sử bạn hoàn thành việc phân tích lỗi 100 mẫu bị phân loại nhầm trên tập phát triển và có được kết quả như sau:

Ảnh	Chó	Thú Họ Mèo	Ảnh Mờ	Ghi chú
1	✓			Chó pitbull có màu lạ
2			✓	
3		✓	✓	Ảnh sư tử chụp ở sở thú trong một ngày mưa
4		✓		Một con báo bị khuất sau cây
...
Tổng %	8%	43%	61%	

Bạn thấy rằng việc khắc phục lỗi phân loại nhầm trên hạng mục Chó có thể loại bỏ tối đa 8% lỗi. Khắc phục các lỗi trên hạng mục Thú Họ Mèo và Ảnh Mờ có thể loại bỏ được nhiều lỗi hơn. Vì vậy bạn có thể chọn một trong hai hạng mục trên để tập trung vào. Nếu nhóm của bạn có đủ nhân lực để khắc phục nhiều hạng mục lỗi song song, bạn có thể phân công một số kỹ sư khắc phục lỗi trên hạng mục Thú Họ Mèo, những người còn lại khắc phục lỗi trên hạng mục Ảnh Mờ.

Phân tích lỗi không tạo ra một công thức toán học cứng nhắc cho bạn biết hạng mục nào có độ ưu tiên cao nhất. Bạn cũng cần đánh giá khả năng cải thiện có thể đạt được trên các hạng mục cũng như khối lượng công việc cần thiết để giải quyết từng hạng mục đó.

16. Đọn dẹp những mẫu bị gán nhãn nhầm trong tập phát triển và tập kiểm tra

Trong quá trình phân tích lỗi, bạn có thể nhận thấy rằng một vài mẫu trong trong tập phát triển đã bị gán nhãn nhầm. Khi tôi nói "bị gán nhãn nhầm" ở đây, ý tôi là những tấm ảnh đã bị gán nhãn nhầm bởi người dán nhãn trước cả khi được đưa vào thuật toán. Hay nói cách khác, nhãn lớp của một mẫu (x,y) có giá trị y sai. Ví dụ, có thể một số ảnh không chứa mèo bị gán nhãn nhầm thành có mèo và ngược lại. Nếu bạn nghi ngờ rằng tỷ lệ những ảnh bị gán nhãn nhầm là đáng kể, hãy thêm một hạng mục để theo dõi tỷ lệ các mẫu bị gán nhãn nhầm:

Ảnh	Chó	Thú họ mèo	Ảnh mờ	Dán nhãn sai	Ghi chú
...					
98				✓	Người dán nhãn đã bỏ qua con mèo ở phần nền
99		✓			
100				✓	Bức vẽ của con mèo, không phải con mèo thật
Tổng %	8%	43%	61%	6%	

Vậy bạn có nên sửa lại những nhãn sai trong tập phát triển không? Hãy nhớ rằng mục tiêu của tập phát triển là giúp bạn nhanh chóng đánh giá những thuật toán nhờ đó bạn có thể biết liệu Thuật toán A hay Thuật toán B là tốt hơn. Nếu tỷ lệ bị gán nhãn nhầm trong tập phát triển cản trở khả năng ra những quyết định này của bạn, thì sẽ là đáng để bỏ thời gian ra để sửa lại những nhãn bị gán nhầm của tập phát triển.

Để ví dụ, giả sử chất lượng bộ phân loại của bạn là:

- Độ chính xác tổng thể trên tập phát triển..... 90% (10% lỗi tổng thể.)
- Những lỗi gây ra bởi các mẫu bị gán nhãn nhầm..... 0.6% (6% các lỗi trong tập phát triển.)
- Những lỗi do các nguyên nhân khác..... 9.4% (94% các lỗi trong tập phát triển)

Ở đây, tỷ lệ 0.6% sai do gán nhầm nhãn có thể không quá đáng kể so với tỷ lệ 9.4% các lỗi mà bạn có thể cải thiện. Không có một tác hại nào trong việc sửa thủ công những ảnh bị gán nhãn nhầm trong tập phát triển cả, nhưng nó không quá quan trọng để làm vậy: Việc bạn không biết liệu hệ thống của mình có 10% hay 9.4% lỗi chung là có thể chấp nhận được.

Giả sử bạn tiếp tục cải thiện bộ nhận dạng mèo và đạt chất lượng:

- Độ chính xác tổng thể trên tập phát triển..... 98.0% (2.0% lỗi tổng thể.)
- Những lỗi gây ra do các mẫu bị gán nhãn nhầm..... 0.6%. (30% các lỗi trong tập phát triển.)
- Những lỗi do các nguyên nhân khác..... 1.4% (70% các lỗi trong tập phát triển)

30% lỗi của bạn đến từ những ảnh bị gán nhãn nhầm trong tập phát triển, việc này thêm một lượng đáng kể lỗi vào các đánh giá độ chính xác của bạn. Trong trường hợp này, cải thiện chất lượng của các nhãn trong tập phát triển là một việc đáng làm. Xử lý những mẫu bị gán nhầm sẽ giúp bạn biết được lỗi của bộ phân loại gần với 1.4% hay 2% -- một sự khác biệt đáng kể.

Việc bắt đầu xây dựng hệ thống với một vài nhãn sai trong tập phát triển/kiểm tra không phải là không phổ biến. Khi hệ thống được cải thiện, số mẫu bị gán nhãn sai dần chiếm tỷ lệ tương đối lớn trong tập lỗi, lúc này ta mới tiến hành sửa lại các nhãn đó.

Chương trước đã hướng dẫn cách bạn có thể cải thiện các hạng mục lỗi như Chó, Thú Họ Mèo và Ảnh Mờ qua những cải tiến về thuật toán. Bạn đã học trong chương này rằng bạn cũng có thể làm việc trên hạng mục Bị Gán Nhãn Sai nữa -- thông qua cải thiện các nhãn của dữ liệu.

Bất kể quy trình bạn áp dụng để sửa các nhãn trong tập huấn luyện là gì, hãy nhớ áp dụng cùng một quy trình cho các nhãn của tập kiểm tra, để đảm bảo tập phát triển và kiểm tra vẫn được lấy ra từ cùng một phân phối. Chỉnh sửa các tập phát triển và kiểm tra cùng nhau sẽ giúp tránh được những vấn đề chúng ta đã bàn trong Chương 6, khi nhóm của bạn tối ưu chất lượng cho tập phát triển để rồi phát hiện ra sau đó là chúng đang được đánh giá dựa trên một tiêu chuẩn khác dựa trên một tập kiểm tra khác.

Nếu bạn quyết định cải thiện chất lượng nhãn, hãy xem xét việc kiểm tra kỹ các nhãn của những mẫu mà hệ thống của bạn đã phân loại nhầm cũng như các nhãn của những mẫu mà nó đã phân loại chính xác. Rất có thể là cả nhãn gốc và thuật toán học

tập của bạn đều đã sai trên một mẫu. Nếu bạn chỉ sửa những nhãn của mẫu mà hệ thống đã phân loại nhầm, bạn có thể đã gây ra thiên lệch trong đánh giá. Nếu bạn có 1.000 mẫu trong tập phát triển, và nếu bộ phân loại của bạn có 98,0% độ chính xác, sẽ dễ hơn khi kiểm tra 20 mẫu đã bị phân loại nhầm hơn là cả 980 mẫu được phân loại chính xác. Bởi vì trên thực tế, sẽ dễ hơn khi chỉ kiểm tra những mẫu bị phân loại nhầm, do đó sự thiên lệch sẽ lén vào một vài tập phát triển. Sự thiên lệch này là chấp nhận được nếu bạn chỉ quan tâm vào việc phát triển một sản phẩm hay một ứng dụng, nhưng nó sẽ là một vấn đề nếu bạn định sử dụng kết quả trong một bài báo nghiên cứu khoa học hay cần một phép đo hoàn toàn không thiên lệch cho độ chính xác của tập kiểm tra.

17. Nếu bạn có một tập phát triển lớn, chia nó thành hai tập con và chỉ phân tích trên một tập

Giả sử bạn có một tập phát triển lớn gồm 5000 mẫu, thu về tỉ lệ lỗi là 20%. Ở đây, thuật toán của bạn đang phân loại nhầm khoảng 1000 mẫu ảnh của tập phát triển. Sẽ rất lâu để phân tích thủ công 1000 ảnh này, vì vậy, ta có thể không sử dụng tất cả 1000 ảnh đó trong phân tích lỗi.

Trong trường hợp này, tôi sẽ chia tập phát triển thành hai tập con không giao nhau: một tập sẽ được phân tích thủ công, tập còn lại thì không. Thuật toán sẽ overfit phần được phân tích thủ công nhanh hơn tập còn lại. Phần còn lại có thể được sử dụng để điều chỉnh tham số.



Let's continue our example above, in which the algorithm is misclassifying 1,000 out of 5,000 dev set examples. Suppose we want to manually examine about 100 errors for error analysis (10% of the errors). You should randomly select 10% of the dev set and place that into what we'll call an **Eyeball dev set** to remind ourselves that we are looking at it with our eyes. (For a project on speech recognition, in which you would be listening to audio clips, perhaps you would call this set an Ear dev set instead). The Eyeball dev set therefore has 500 examples, of which we would expect our algorithm to misclassify about 100.

Hãy cùng tiếp tục với ví dụ ở trên: ví dụ thuật toán đang phân loại nhầm 1000 mẫu trên tổng số 5000 mẫu trong tập phát triển. Giả sử chúng ta muốn sử dụng 100 mẫu bị phân loại nhầm để phân tích lỗi (10% tổng số lỗi). Bạn nên chọn ra 10% mẫu trong tập phát triển một cách ngẫu nhiên và đặt nó vào trong một tập mà chúng ta sẽ gọi là **tập phát triển Eyeball** để tự nhắc chúng ta rằng ta sẽ trực tiếp nhìn vào bằng mắt. (Đối với những dự án nhận diện giọng nói mà bạn phải nghe audio, có lẽ bạn sẽ gọi tập này là tập phát triển Ear). Tập phát triển Eyeball chứa 500 mẫu, trong đó chúng ta sẽ kỳ vọng thuật toán phân loại nhầm khoảng 100 mẫu.

Tập con thứ hai của tập phát triển, được gọi là **tập phát triển Blackbox**, sẽ chứa 4500 mẫu còn lại. Bạn có thể sử dụng tập phát triển Blackbox để đánh giá các bộ phân loại một cách tự động bằng cách đo tỉ lệ lỗi chúng. Bạn cũng có thể sử dụng tập này để lựa chọn thuật toán hoặc điều chỉnh tham số. Tuy nhiên, bạn nên tránh trực tiếp phân tích thủ công tập này. Chúng ta sử dụng thuật ngữ "Blackbox" vì chúng ta chỉ sử dụng tập con này để thu về những đánh giá "Blackbox" của bộ phân loại.



Tại sao chúng ta lại chia tập phát triển một cách riêng biệt thành tập phát triển Eyeball và tập phát triển Blackbox? Khi bạn hiểu rõ hơn về các mẫu trong tập phát triển Eyeball, bạn sẽ overfit tập phát triển đó nhanh hơn. Khi bạn thấy chất lượng của mô hình trên tập phát triển Eyeball đang tăng nhanh hơn nhiều so với tập phát triển Blackbox, bạn đã overfit tập phát triển Eyeball. Trong trường hợp này, bạn có thể phải loại bỏ tập Eyeball đi, tìm một tập khác thay thế bằng cách chuyển các mẫu từ tập phát triển Blackbox sang tập phát triển Eyeball, hoặc thu về những mẫu có nhãn mới.

Việc phân chia tập phát triển thành hai tập riêng biệt -- tập phát triển Eyeball và tập phát triển Blackbox -- cho biết khi nào việc phân tích lỗi thủ công đang khiến tập Eyeball bị overfit.

18. Tập phát triển Eyeball và Blackbox nên lớn như thế nào?



Tập phát triển Eyeball phải đủ lớn để giúp bạn có cái nhìn về các hạng mục lỗi chính của thuật toán. Nếu bạn đang làm một tác vụ mà con người làm tốt (chẳng hạn như nhận diện mèo trong các ảnh), dưới đây là một vài hướng dẫn sơ bộ.

- Một tập phát triển Eyeball ở đó các bộ phân loại tạo ra 10 lỗi có thể được coi là rất nhỏ. Với chỉ 10 lỗi, rất khó để ước lượng chuẩn xác ảnh hưởng của những hạng mục lỗi khác nhau. Nhưng nếu bạn có rất ít dữ liệu và không thể tăng tập phát triển, việc này vẫn tốt hơn so với không làm gì và sẽ giúp ích đối với việc sắp xếp ưu tiên của dự án.
- Nếu bộ phân loại tạo ra khoảng 20 lỗi trong tập phát triển Eyeball, bạn sẽ bắt đầu cảm nhận được sơ bộ về các nguồn lỗi chính.
- Với khoảng 50 lỗi, bạn sẽ có cảm nhận tốt về các nguồn lỗi chính.
- Với khoảng 100 lỗi, bạn sẽ cảm nhận được rất tốt các nguồn lỗi chính. Tôi đã chứng kiến nhiều người phân tích thủ công thậm chí tới 500 lỗi. Điều đó không gây hại miễn là bạn có đủ dữ liệu.

Giả sử bộ phân loại của bạn có tỷ lệ lỗi 5%. Để đảm bảo bạn có khoảng 100 mẫu bị phân loại sai trong tập phát triển Eyeball, tập phát triển Eyeball sẽ phải có khoảng 2.000 mẫu (bởi vì $0,05 * 2.000 = 100$). Tỷ lệ lỗi do bộ phân loại gây ra càng thấp, tập dữ liệu phát triển Eyeball càng phải lớn để có được một tập lỗi đủ lớn cho phân tích.

Nếu bạn đang làm việc trong một tác vụ mà ngay cả con người cũng không thể làm tốt, thì việc kiểm tra tập phát triển Eyeball sẽ không hữu ích vì khó hình dung tại sao thuật toán không phân loại mẫu một cách chính xác. Trong trường hợp này, bạn có thể bỏ qua việc thiết lập tập phát triển Eyeball. Chúng ta thảo luận hướng dẫn cho những vấn đề này trong một chương sau.



Tập phát triển Blackbox thì sao? Trước đây chúng ta đã khẳng định rằng các tập phát triển khoảng 1.000-10.000 mẫu là khá phổ biến. Để củng cố nhận định đó, một tập phát triển Blackbox gồm 1.000-10.000 mẫu thường cung cấp đủ dữ liệu để tinh chỉnh siêu tham số và lựa chọn mô hình, mặc dù có rất ít bất lợi khi có nhiều dữ liệu hơn. Một tập phát triển Blackbox kích thước 100 sẽ nhỏ nhưng vẫn hữu ích.

Nếu bạn có một tập phát triển nhỏ thì bạn có thể không đủ dữ liệu để phân chia thành tập phát triển Eyeball và tập phát triển Blackbox đủ lớn để đáp ứng mục đích sử dụng của chúng. Thay vào đó, toàn bộ tập phát triển của bạn có thể phải được sử dụng như là tập phát triển Eyeball, tức là bạn sẽ kiểm tra thủ công toàn bộ tập phát triển.

Giữa tập phát triển Eyeball và tập phát triển Blackbox, tôi cho rằng tập phát triển Eyeball quan trọng hơn (giả định bạn đang giải quyết một vấn đề mà con người có thể giải quyết tốt và việc kiểm tra mẫu giúp bạn hiểu rõ hơn). Nếu bạn chỉ có một tập phát triển Eyeball, bạn có thể thực hiện phân tích lỗi, lựa chọn mô hình và tinh chỉnh tất cả siêu tham số trên tập dữ liệu này. Nhược điểm của việc chỉ có một tập phát triển Eyeball là nguy cơ overfitting trên tập phát triển là lớn hơn.

Nếu bạn có quyền truy cập vào nhiều dữ liệu thì kích thước của tập phát triển Eyeball sẽ chủ yếu dựa trên bao nhiêu mẫu mà

bạn có thời gian để phân tích thủ công. Ví dụ, tôi hiếm khi thấy ai phân tích thủ công hơn 1.000 lỗi.

19. Điều cần nhớ: Phân tích lỗi cơ bản

- Khi bạn bắt đầu một dự án mới, đặc biệt nếu bạn không phải là chuyên gia trong lĩnh vực đó, sẽ rất khó để đoán những hướng giải quyết triển vọng nhất.
- Vì vậy đừng cố bắt đầu với việc thiết kế và xây dựng một hệ thống hoàn hảo. Thay vào đó hãy xây dựng và huấn luyện một hệ thống cơ bản một cách nhanh nhất có thể -- thậm chí chỉ trong một vài ngày. Sau đó, sử dụng phân tích lỗi để xác định những hướng đi triển vọng và từ đó lặp đi lặp lại việc cải thiện thuật toán của bạn.
- Thực hiện phân tích lỗi bằng cách kiểm tra thủ công khoảng 100 mẫu trong tập phát triển mà thuật toán phân loại sai và điểm qua những hạng mục lỗi chính. Sử dụng thông tin này để sắp xếp thứ tự ưu tiên các loại lỗi cần khắc phục.
- Xem xét việc tách tập phát triển thành một tập phát triển Eyeball cho việc kiểm tra thủ công, và một tập phát triển Blackblox mà bạn sẽ không kiểm tra thủ công. Nếu chất lượng trên tập phát triển Eyeball tốt hơn rất nhiều so với trên tập phát triển Blackbox, bạn đã overfit tập phát triển Eyeball và nên xem xét việc thu thập thêm dữ liệu cho tập này.
- Tập phát triển Eyeball nên đủ lớn để số lượng mẫu mà thuật toán của bạn phân loại sai đủ cho bạn phân tích. Một tập phát triển Blackbox khoảng 1.000-10.000 mẫu là đủ cho rất nhiều những ứng dụng.
- Nếu tập phát triển của bạn không đủ lớn để tách ra theo cách này, hãy lấy toàn bộ tập phát triển làm một tập phát triển Eyeball dành cho việc phân tích lỗi thủ công, chọn mô hình, và điều chỉnh siêu tham số.

Phần 3: Độ chêch và Phuơng sai

20. Độ chệch và Phương sai: Hai nguồn lớn của lỗi

Giả sử khi huấn luyện, tập phát triển và tập kiểm tra có cùng phân phối. Khi đó bạn cần luôn cố gắng thu thập thêm dữ liệu huấn luyện, vì dù sao điều đó cũng chỉ giúp cải thiện chất lượng, đúng không?

Mặc dù có thêm dữ liệu không thể giảm hiệu quả, thật không may điều này không luôn luôn mang lại hiệu quả nhiều như bạn mong đợi. Việc thu thập thêm dữ liệu có thể trở nên lãng phí thời gian. Vậy làm thế nào để quyết định khi nào nên thêm dữ liệu, khi nào không nên?

Có hai nguồn chính dẫn đến lỗi trong machine learning: độ chệch và phương sai. Hiểu được chúng sẽ giúp bạn quyết định liệu rằng có đáng bỏ ra thời gian để thêm dữ liệu, cũng như các kỹ thuật khác để cải thiện chất lượng của mô hình.

Giả sử bạn hy vọng xây dựng một bộ nhận dạng mèo với 5% lỗi. Hiện tại, tập huấn luyện của bạn có tỉ lệ lỗi là 15%, và tập phát triển có tỉ lệ lỗi là 16%. Trong trường hợp này, việc thêm dữ liệu có thể không giúp được gì nhiều. Bạn nên tập trung vào các thay đổi khác. Chắc chắn rằng việc tăng số mẫu cho tập huấn luyện chỉ khiến quá trình huấn luyện mô hình trên tập này trở nên khó khăn hơn. (Chúng tôi sẽ giải thích trong một chương sau.)

Nếu tỉ lệ lỗi trên tập huấn luyện là 15% (tức độ chính xác 85%), nhưng mục tiêu của bạn là 5% lỗi (độ chính xác 95%), thì vấn đề trước tiên cần giải quyết là cải thiện chất lượng thuật toán của bạn trên tập huấn luyện. Hiệu quả trên tập phát triển/kiểm tra thường thấp hơn chất lượng trên tập huấn luyện. Bởi vậy nếu bạn đang có độ chính xác 85% trên các mẫu mà thuật toán từng thấy, không có cách nào để đạt được độ chính xác 95% cho các mẫu mà thuật toán chưa thấy bao giờ.

Giả sử như trên rằng thuật toán của bạn có 16% lỗi (độ chính xác 84%) trên tập phát triển. Chúng ta tách 16% lỗi này ra hai thành phần:

- Thứ nhất, tỉ lệ lỗi của thuật toán trên tập huấn luyện, là 15% trong ví dụ này. Chúng ta tạm nghĩ giá trị này như **độ chệch** của thuật toán.

Thứ hai, chất lượng của thuật toán trên tập phát triển (hoặc kiểm tra) kém hơn bao nhiêu so với trên tập huấn luyện. Trong ví dụ này, thuật toán làm việc kém hơn 1% trên tập phát triển so với tập huấn luyện. Chúng ta tạm coi giá trị này như **phương sai** của thuật toán [6].

Một số thay đổi trong thuật toán học có thể giải quyết thành phần thứ nhất của lỗi -**độ chệch** -- và cải thiện chất lượng của nó trên tập huấn luyện. Một số thay đổi giải quyết thành phần thứ hai -- **phương sai** -- và giúp thuật toán tổng quát hóa tốt hơn từ tập huấn luyện tới tập phát triển/kiểm tra [7]. Để lựa chọn thay đổi tiềm năng nhất, sẽ rất hữu ích khi hiểu thành phần nào trong hai thành phần lỗi là đáng để giải quyết hơn.

Phát triển một trực giác tốt về Độ chệch và Phương sai sẽ giúp bạn chọn những thay đổi hữu hiệu cho thuật toán.

[6] Ngành thống kê có những định nghĩa chính thống hơn cho độ chệch và phương sai mà ở đây chúng ta không cần lưu tâm. Một cách xấp xỉ, độ chệch là tỉ lệ lỗi thuật toán của bạn trên tập huấn luyện khi tập này rất lớn. Phương sai là sự giảm chất lượng trên tập kiểm tra so với tập huấn luyện trong thiết lập này. Khi phép đo lỗi là trung bình bình phương lỗi, bạn có thể viết được công thức tính hai đại lượng này, và chứng minh được rằng Tổng Lỗi = Độ Chệch + Phương Sai. Nhưng với mục đích xác định làm thế nào để tạo sự tiến triển trong một bài toán học máy, định nghĩa ít chính thống hơn của độ chệch và phương sai như ở đây là đã đủ.

Cũng có một vài phương pháp có thể đồng thời giảm độ chệch và phương sai bằng cách tạo sự thay đổi lớn trong kiến trúc hệ thống. Tuy nhiên, những phương pháp này có xu hướng khó phát hiện và triển khai hơn.

21. Những ví dụ về Độ chêch và Phương sai

Hãy xem xét việc phân loại mèo của chúng ta. Một bộ phân loại "lý tưởng" (như con người) có thể đạt được hiệu suất gần như hoàn hảo cho việc này.

Giả sử thuật toán của bạn thực hiện như sau:

- Tỉ lệ lỗi huấn luyện = 1%
- Tỉ lệ lỗi phát triển = 11%

Nó gặp phải vấn đề gì? Áp dụng định nghĩa từ những chương trước, chúng ta ước tính độ chêch là 1% và phương sai là 10% (=11%-1%). Do đó, nó có **phương sai cao**. Bộ phân loại có lỗi huấn luyện rất thấp, nhưng nó lại không khái quát hoá được cho tập phát triển. Điều này cũng được gọi là **overfitting**.

Bây giờ hãy xem xét điều này:

- Tỉ lệ lỗi huấn luyện = 5%
- Tỉ lệ lỗi phát triển = 16%

Chúng ta ước tính độ chêch là 15% và phương sai là 1%. Bộ phân loại này khớp khá kém với tập huấn luyện với 15% tỉ lệ lỗi, nhưng tỉ lệ lỗi ở tập phát triển chỉ cao hơn một chút so với tập huấn luyện. Do đó, bộ phân loại này có **độ chêch cao**, nhưng phương sai thấp. Chúng ta nói rằng thuật toán này là **underfitting**.

Bây giờ hãy xem xét điều này:

- Tỉ lệ lỗi huấn luyện = 15%
- Tỉ lệ lỗi phát triển = 30%

Chúng ta ước tính độ chêch là 15% và phương sai là 15%. Bộ phân loại này có **độ chêch cao và phương sai cao**: Nó hoạt động kém ở tập huấn luyện, và do đó có độ chêch cao, và hiệu suất của nó trên tập phát triển còn tệ hơn, do đó nó cũng có phương sai cao. Thuật ngữ overfitting/underfitting rất khó áp dụng ở đây vì bộ phân loại đồng thời bị overfitting và underfitting.

Cuối cùng, hãy xem xét điều này:

- Tỉ lệ lỗi huấn luyện = 0,5%
- Tỉ lệ lỗi phát triển = 1%

Bộ phân loại này đang hoạt động tốt, vì nó có độ chêch thấp và phương sai thấp. Chúc mừng bạn đã đạt được hiệu suất tuyệt vời!

22. So sánh với tỉ lệ lỗi tối ưu

Trong ví dụ nhận dạng mèo của chúng ta, tỉ lệ lỗi "lý tưởng" -- tỉ lệ có thể đạt được bởi một bộ phân loại "tối ưu" -- là gần với 0%. Gần như mọi lúc, một người nhìn vào bức ảnh có thể nhận ra có mèo trong đó hay không; do đó chúng ta có thể hy vọng máy móc cũng làm được điều tương tự.

Một số bài toán khác thì khó hơn. Ví dụ như chúng ta xây dựng một hệ thống nhận dạng giọng nói và nhận ra rằng 14% các clip âm thanh có quá nhiều nhiễu nên hoặc khó hiểu tới mức ngay cả con người cũng không thể nhận ra những gì được nói tới. Trong trường hợp này, ngay cả hệ thống "tối ưu" nhất cũng có thể có lỗi khoảng 14%.

Giả sử rằng với bài toán nhận dạng giọng nói này, thuật toán của bạn đạt được:

- Lỗi huấn luyện = 15%
- Lỗi phát triển = 30%

Chất lượng trên tập huấn luyện đã gần với tỉ lệ lỗi tối ưu là 14%. Do đó, không có nhiều chỗ để cải thiện độ chênh hoặc chất lượng trên tập huấn luyện. Tuy nhiên, thuật toán này không tổng quát hoá tốt trên tập phát triển, do đó có rất nhiều chỗ để cải thiện lỗi do phương sai.

Ví dụ này tương tự như ví dụ thứ ba trong chương trước, ở đó lỗi huấn luyện là 15% và lỗi phát triển là 30%. Nếu tỉ lệ lỗi tối ưu là xấp xỉ 0%, 15% lỗi huấn luyện để lại nhiều chỗ để cải thiện. Điều này gợi ý cho chúng ra rằng những thay đổi làm giảm độ chênh có thể mang lại kết quả. Nhưng nếu tỉ lệ lỗi tối ưu là 14%, thì chất lượng tương tự trên tập huấn luyện sẽ cho chúng ta biết rằng có rất ít chỗ để cải thiện độ chênh của bộ phân loại.

Với các bài toán trong đó tỉ lệ lỗi tối ưu khác xa 0%, đây là phân tích chi tiết hơn về lỗi của một thuật toán. Tiếp tục với ví dụ nhận dạng giọng nói của chúng ta ở trên, lỗi tổng cộng trên tập phát triển là 30% và nó có thể được chia nhỏ như sau (một phân tích tương tự có thể được áp dụng cho lỗi trên tập kiểm tra):

- **Tỉ lệ lỗi tối ưu ("độ chênh không tránh được")** 14%. Giả sử chúng ra quyết định rằng, ngay cả khi dùng hệ thống nhận dạng giọng nói tốt nhất trên thế giới, chúng ta vẫn phải chịu 14% lỗi. Chúng ta có thể coi lỗi đó là phần không tránh được của độ chênh của một thuật toán học máy.
- **Độ chênh có thể tránh được:** 1%. Hiệu giữa tỉ lệ lỗi huấn luyện và tỉ lệ lỗi tối ưu. [8]
- **Phương sai:** 15%. Hiệu giữa tỉ lệ lỗi trên tập phát triển và tỉ lệ lỗi trên tập huấn luyện.

Để kết nối điều này với những định nghĩa trước kia, độ chênh và độ chênh có thể tránh được liên hệ với nhau như sau: [9]

Độ chênh = Tỉ lệ lỗi tối ưu ("độ chênh không thể tránh được") + độ chênh có thể tránh được

"Độ chênh có thể tránh được" phản ánh thuật toán của bạn hoạt động kém bao nhiêu so với "bộ phân loại tối ưu".

Khái niệm phương sai giữ nguyên như trước. Theo lý thuyết, chúng ta luôn có thể giảm phương sai về gần 0 bằng cách huấn luyện trên một tập huấn luyện cực lớn. Do đó, tất cả phương sai là "có thể tránh được" khi tập dữ liệu đủ lớn, và không có cái gọi là "phương sai không thể tránh được".

Xem xét thêm một ví dụ nữa, trong đó tỉ lệ lỗi tối ưu là 14%, chúng ta có:

- Lỗi huấn luyện = 15%
- Lỗi phát triển = 16%

Trong khi ở chương trước chúng ta gọi đây là một bộ phân loại có độ chênh cao, bây giờ chúng ta nói rằng lỗi từ độ chênh có thể tránh được là 1% và lỗi từ phương sai là khoảng 1%. Do đó, thuật toán của chúng ta đã là rất tốt và có rất ít cơ hội cải thiện. Nó chỉ kém đúng 2% so với tỉ lệ lỗi tối ưu.

Từ những ví dụ này chúng ta thấy rằng tỉ lệ lỗi tối ưu rất hữu ích cho việc định hướng các bước tiếp theo. Trong thống kê, tỉ lệ lỗi tối ưu được gọi là **tỉ lệ lỗi Bayes**, hay tỉ lệ Bayes.

Làm sao chúng ta biết được tỉ lệ lỗi tối ưu? Với những việc mà con người làm tốt, như nhận dạng ảnh hay phiên âm clip, bạn có thể nhờ ai đó gán nhãn sau đó đo độ chính xác của những nhãn này với tập huấn luyện của bạn. Điều này cung cấp con số ước tính của tỉ lệ lỗi tối ưu. Nếu bạn làm việc với một bài toán mà ngay cả con người cũng khó giải (ví dụ như dự đoán xem nên gợi ý bộ phim nào, hay hiện quảng cáo nào trước người dùng) thì khó để ước tính tỉ lệ lỗi tối ưu.

Trong phần "So sánh với chất lượng mức con người (chương 33 tới chương 35), tôi sẽ thảo luận chi tiết hơn quá trình so sánh

chất lượng một thuật toán học máy với chất lượng mức con người.

Trong một vài chương trước, bạn đã học cách ước tính phương sai và độ chêch có thể tránh được/không thể tránh được bằng cách xem xét tỉ lệ lỗi huấn luyện và phát triển. Chương tiếp theo sẽ thảo luận về cách bạn có thể sử dụng những hiểu biết sâu sắc từ phân tích như vậy để ưu tiên các kỹ thuật làm giảm độ chêch so với các kỹ thuật làm giảm phương sai. Có nhiều kỹ thuật khác nhau nên áp dụng, tùy thuộc vào vấn đề hiện tại trong dự án của bạn là độ chêch (có thể tránh được) cao hay phương sai cao. Đọc tiếp!

■

CHÚ THÍCH:

[8] Nếu con số này là âm, bạn đang làm tốt hơn ở trên tập huấn luyện so với tỉ lệ lỗi tối ưu. Điều này có nghĩa là bạn đang overfit tập huấn luyện và thuật toán đã ghi nhớ quá mức tập huấn luyện. Bạn nên tập trung vào các phương pháp giảm phương sai hơn là các phương pháp giảm sâu hơn độ chêch.

[9] Các định nghĩa này được chọn để truyền đạt cái nhìn sâu sắc về cách cải thiện thuật toán học máy của bạn. Các định nghĩa này khác với cách các nhà thống kê định nghĩa Độ chêch và Phương sai. Về mặt kỹ thuật, những gì tôi định nghĩa là "Độ chêch" nên được gọi là "Lỗi chúng ta quy cho độ chêch", và "Độ chêch có thể tránh được" nên là "Lỗi chúng ta quy cho độ chêch của thuật toán học mà lớn hơn tỉ lệ lỗi tối ưu".

23. Xử lý Độ chêch và Phương sai

Đây là công thức đơn giản nhất để giải quyết các vấn đề độ chêch và phương sai:

- Nếu bạn có độ chêch cao có thể tránh được, hãy tăng kích thước mô hình của bạn (ví dụ: tăng kích thước của mạng nơ-ron bằng cách thêm các tầng/neurons)
- Nếu bạn có phương sai cao, hãy thêm dữ liệu vào tập huấn luyện của bạn.

Nếu bạn có thể tăng kích thước của mạng nơ-ron và dữ liệu huấn luyện lên vô hạn thì bạn sẽ có khả năng xử lý rất tốt trên nhiều bài toán machine learning.

Trong thực tế, việc tăng kích thước của mô hình cuối cùng sẽ khiến bạn gặp phải các vấn đề về tính toán bởi vì việc huấn luyện các mô hình cực lớn là rất chậm. Bạn cũng có thể cạn kiệt khả năng có được nhiều dữ liệu huấn luyện hơn. (Ngay cả trên internet, chỉ có một số lượng hữu hạn những hình ảnh mèo!)

Những kiến trúc mô hình khác nhau, ví dụ các kiến trúc mạng nơ-ron khác nhau, sẽ có các mức độ chêch/phương sai khác nhau cho vấn đề của bạn. Những nghiên cứu gần đây về học sâu đã phát triển nhiều kiến trúc mô hình sáng tạo. Vì vậy, nếu bạn đang sử dụng các mạng nơ-ron, những tài liệu học thuật có thể là một nguồn cảm hứng tuyệt vời. Ngoài ra còn có rất nhiều ứng dụng triển khai mã nguồn mở tuyệt vời trên GitHub. Nhưng kết quả của việc thử nghiệm các kiến trúc mới khó dự đoán hơn so với công thức đơn giản của việc tăng kích thước mô hình và thêm dữ liệu.

Việc tăng kích thước mô hình nhìn chung làm giảm độ chêch, nhưng nó cũng có thể làm tăng phương sai và tăng nguy cơ overfitting. Tuy nhiên, vấn đề overfitting này thường chỉ phát sinh khi bạn không sử dụng regularization. Nếu bạn thêm vào một phương pháp regularization được thiết kế tốt, thì bạn thường có thể tăng kích thước mô hình một cách an toàn mà không tăng overfitting.

Giả sử bạn đang áp dụng học sâu, với L2 regularization hoặc dropout, với tham số regularization hoạt động tốt nhất phát triển. Nếu bạn tăng kích thước mô hình, thường thì chất lượng của mô hình sẽ giữ nguyên hoặc cải thiện; nó thường không có khả năng xấu đi đáng kể. Lý do duy nhất để tránh sử dụng một mô hình lớn hơn là chi phí tính toán tăng lên.

24. Sự đánh đổi giữa Độ chêch và Phuong sai

Bạn có thể đã nghe nói về "sự đánh đổi giữa Độ chêch và Phuong sai". Trong các thay đổi bạn có thể thực hiện đổi với hầu hết các thuật toán học, có một số cách giảm sai số độ chêch nhưng với chi phí phải trả là gia tăng phuong sai và ngược lại. Điều này tạo ra một sự "đánh đổi" giữa độ chêch và phuong sai.

Ví dụ việc tăng kích thước mô hình của bạn, thêm các neurons/tầng trong mạng nơ-ron hoặc thêm các đầu vào đặc trưng -- nhìn chung sẽ làm giảm độ chêch nhưng có thể làm tăng phuong sai. Một cách khác, việc thêm regularization thường làm tăng độ chêch nhưng giảm phuong sai.

Ngày nay, chúng ta thường có thể truy cập vào nguồn dữ liệu phong phú và có thể sử dụng các mạng nơ-ron rất lớn (học sâu). Do đó, có ít đánh đổi hơn, và hiện có nhiều lựa chọn hơn để giảm độ chêch mà không làm ảnh hưởng phuong sai, và ngược lại.

Ví dụ, bạn thường có thể tăng kích thước mạng nơ-ron và điều chỉnh phuong thức regularization để giảm độ chêch mà không gia tăng đáng kể phuong sai. Bằng cách thêm dữ liệu huấn luyện, bạn cũng thường có thể giảm phuong sai mà không ảnh hưởng đến độ chêch.

Nếu bạn chọn một kiến trúc mô hình phù hợp với tác vụ của mình, bạn cũng có thể giảm đồng thời độ chêch và phuong sai. Tuy nhiên sẽ khó khăn để chọn một kiến trúc như vậy.

Trong một vài chương tiếp theo, chúng ta sẽ thảo luận thêm các kỹ thuật cụ thể để giải quyết độ chêch và phuong sai.

25. Các kĩ thuật để giảm độ chêch có thể tránh được

Nếu thuật toán học của bạn gặp vấn đề với *độ chêch có thể tránh được* lớn, bạn có thể thử những kĩ thuật sau:

- **Tăng kích thước mô hình** (ví dụ như số lượng neuron/tầng): Kĩ thuật này làm giảm độ chêch, vì nó cho phép fit tập huấn luyện tốt hơn. Nếu bạn thấy việc này làm tăng phương sai, hãy sử dụng regularization, vốn thường loại bỏ việc tăng phương sai.
- **Thay đổi các đặc trưng đầu vào dựa trên những nhận định có được từ việc phân tích lỗi**: Giả sử việc phân tích lỗi gợi ý rằng hãy tạo thêm những đặc trưng bổ sung nhằm giúp thuật toán loại bỏ một nhóm các lỗi đặc thù. (Chúng ta sẽ bàn vấn đề này kĩ hơn ở chương sau.) Những đặc trưng mới này có thể hiệu quả với cả độ chêch và phương sai. Theo lý thuyết, thêm đặc trưng có thể làm tăng phương sai; nhưng nếu bạn thấy đúng là điều đó xảy ra, hãy sử dụng regularization, vốn thường loại bỏ việc tăng phương sai.
- **Giảm hoặc loại bỏ regularization** (L2 regularization, L1 regularization, dropout): Việc này sẽ làm giảm độ chêch có thể tránh được, nhưng sẽ làm tăng phương sai.
- **Thay đổi kiến trúc mô hình** (ví dụ như kiến trúc mạng nơ-ron) để nó trở nên phù hợp hơn với bài toán của bạn: Kĩ thuật này có thể tác động đến cả độ chêch và phương sai.

Một phương pháp không hữu ích:

- **Thêm dữ liệu huấn luyện**: Kĩ thuật này có ích với các vấn đề về phương sai, nhưng nó thường không có tác động đáng kể đến độ chêch.

26. Phân tích lỗi trên tập huấn luyện

Thuật toán của bạn phải hoạt động tốt trên tập huấn luyện trước khi bạn có thể mong đợi nó hoạt động tốt trên tập phát triển/kiểm tra.

Ngoài các kỹ thuật được mô tả trước đây để giải quyết độ chêch cao, đôi khi tôi cũng thực hiện phân tích lỗi trên *dữ liệu huấn luyện*, theo một giao thức tương tự như phân tích lỗi trên tập phát triển Eyeball. Điều này có thể hữu ích nếu thuật toán của bạn có độ chêch cao, ví dụ như nếu nó không fit tốt với tập huấn luyện.

Ví dụ: giả sử bạn đang xây dựng một hệ thống nhận dạng giọng nói cho một ứng dụng và đã thu thập một tập huấn luyện gồm các đoạn âm thanh từ các tình nguyện viên. Nếu hệ thống của bạn không hoạt động tốt trên tập huấn luyện, bạn có thể xem xét việc nghe một bộ ~100 mẫu mà thuật toán hoạt động kém để hiểu các hạng mục lỗi chính của tập huấn luyện. Tương tự như phân tích lỗi trên tập phát triển, bạn có thể đếm các lỗi trong các hạng mục khác nhau:

Clip âm thanh	Nhiều nền lớn	Người dùng nói nhanh	Xa micro	Bình luận
1	✓			Tiếng ồn xe hơi
2	✓		✓	Tiếng ồn nhà hàng
3		✓	✓	Người dùng la hét khắp phòng khách?
4	✓			Quán cà phê
% tổng thẻ	75%	25%	50%	

Trong ví dụ này, bạn có thể nhận ra rằng thuật toán của bạn đang gặp khó với các mẫu huấn luyện có nhiều nền. Do đó, bạn có thể tập trung vào các kỹ thuật cho phép nó fit hơn với các ví dụ đào tạo với nhiều nền.

Bạn cũng có thể kiểm tra kỹ xem liệu rằng một người có thể diễn dịch các đoạn âm thanh đầu vào cho thuật toán học của bạn. Nếu có quá nhiều nền đến nỗi đơn giản là không ai có thể phát hiện ra những gì được nói, thì có thể sẽ bất hợp lý khi mong đợi bất kỳ thuật toán nào nhận ra chính xác những phát ngôn đó. Chúng ta sẽ thảo luận về lợi ích của việc so sánh thuật toán của bạn với chất lượng mức con người trong một phần sau.

27. Các kỹ thuật làm giảm phương sai

Nếu như thuật toán của bạn có phương sai lớn, bạn có thể thử các kỹ thuật sau:

- **Thêm dữ liệu huấn luyện:** Đây là cách đơn giản và đáng tin cậy nhất để giảm phương sai, miễn là bạn có thể thu thập nhiều dữ liệu hơn một cách đáng kể và đủ sức mạnh tính toán để xử lý dữ liệu.
- **Thêm regularization (L2 regularization, L1 regularization, dropout):** Kỹ thuật này làm giảm phương sai nhưng tăng độ chêch. Kỹ thuật dừng sớm hoạt động rất giống như các phương pháp regularization, và một số tác giả gọi đó là một kỹ thuật regularization.
- **Lựa chọn đặc trưng để giảm số lượng/kiểu đặc trưng đầu vào:** Kỹ thuật này có thể giúp giải quyết các vấn đề về phương sai, nhưng nó cũng có thể làm tăng độ chêch. Việc giảm một chút số lượng các đặc trưng (giả sử giảm từ 1.000 đặc trưng xuống còn 900 đặc trưng) thường như không có ảnh hưởng lớn đến độ chêch. Việc giảm đáng kể số đặc trưng (giả sử giảm từ 1.000 đặc trưng xuống còn 100 đặc trưng, tức là giảm 10 lần) nhiều khả năng có tác dụng đáng kể, miễn là bạn không loại trừ quá nhiều các đặc trưng hữu ích. Trong học sâu hiện đại, khi dữ liệu dồi dào, đã có những thay đổi từ việc lựa chọn đặc trưng và giờ đây hầu như chúng ta cung cấp tất cả các đặc trưng chúng ta có cho thuật toán và để thuật toán chọn ra những đặc trưng nào sẽ sử dụng dựa trên dữ liệu. Nhưng khi tập huấn luyện của bạn nhỏ, kỹ thuật lựa chọn đặc trưng vẫn có thể rất hữu ích.
- **Giảm kích thước mô hình** (chẳng hạn như số lượng neurons/tầng): *Sử dụng một cách thận trọng.* Kỹ thuật này có thể làm giảm phương sai, trong khi có thể làm tăng độ chêch. Tuy nhiên, tôi không khuyến khích sử dụng kỹ thuật này để giảm phương sai. Thêm regularization thường cho chất lượng phân loại tốt hơn. Ưu điểm của việc giảm kích thước mô hình là giảm chi phí tính toán của bạn và do đó tăng tốc độ bạn huấn luyện mô hình. Nếu tăng tốc độ huấn luyện mô hình là hữu ích, thì bằng mọi cách hãy xem xét giảm kích thước mô hình. Nhưng nếu mục tiêu của bạn là giảm phương sai và bạn không quan tâm đến chi phí tính toán, thay vào đó, hãy xem xét việc thêm regularization.

Đây là hai chiến thuật bổ sung, được lặp lại từ chương trước về giải quyết độ chêch:

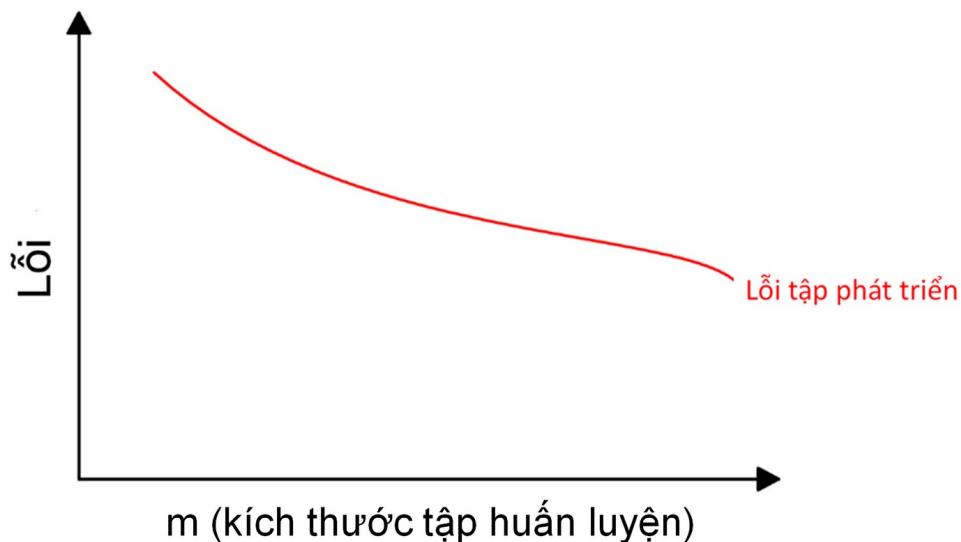
- **Thay đổi các đặc trưng đầu vào dựa trên hiểu biết sâu sắc từ phân tích lỗi** Giả sử rằng việc phân tích lỗi của bạn truyền cảm hứng cho bạn để tạo các đặc trưng bổ sung giúp thuật toán của bạn loại bỏ một hạng mục lỗi cụ thể. Những đặc trưng mới này có thể giảm cả độ chêch và phương sai. Về lý thuyết, việc thêm nhiều đặc trưng có thể làm tăng phương sai; nhưng nếu bạn gặp trường hợp này, hãy sử dụng regularization, việc này thường sẽ loại bỏ sự gia tăng phương sai.
- **Thay đổi kiến trúc mô hình** (chẳng hạn như kiến trúc mạng nơ-ron) để phù hợp hơn với vấn đề của bạn: Kỹ thuật này có thể ảnh hưởng đến cả độ lệch và phương sai.

Phần 4: Đồ thị quá trình học

28. Chẩn đoán độ chêch và phuong sai: Đồ thị quá trình học

Chúng ta đã thấy một số cách để ước tính có bao nhiêu lỗi tạo ra bởi độ chêch tránh được so với phuong sai. Một trong số đó là dự đoán tỉ lệ lỗi tối ưu và tính toán lỗi của thuật toán trên tập huấn luyện và tập phát triển. Hãy cũng thảo luận về một kỹ thuật khác thậm chí còn mang lại nhiều thông tin hơn: biểu diễn một đồ thị quá trình học.

Một đồ thị quá trình học cho thấy sự tương quan giữa lỗi của tập phát triển so với số lượng các mẫu huấn luyện. Để biểu diễn nó, bạn cần áp dụng thuật toán của bạn với các tập huấn luyện có độ lớn khác nhau. Ví dụ, nếu bạn có 1,000 mẫu, bạn có thể huấn luyện riêng biệt các bản sao của thuật toán trên các tập 100, 200, 300, ..., 1000 mẫu. Sau đó bạn có thể biểu diễn sự thay đổi giữa lỗi của tập phát triển so với độ lớn của tập mẫu. Dưới đây là một ví dụ:

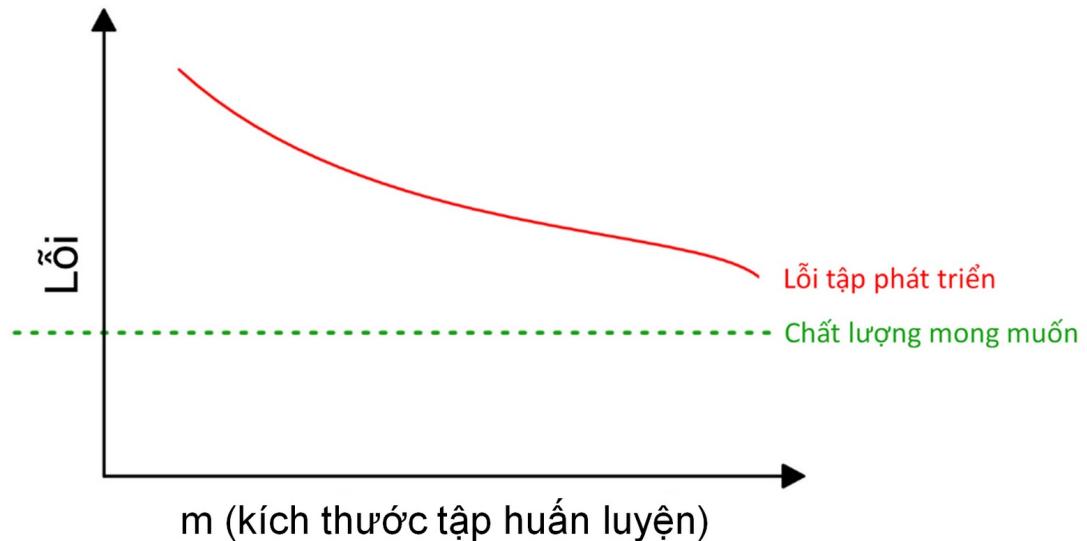


Khi số lượng mẫu tăng, lỗi của tập phát triển nên giảm.

Chúng ta thường sẽ có một số "tỉ lệ lỗi mong muốn" mà chúng ta hy vọng thuật toán của mình cuối cùng sẽ đạt được. Ví dụ:

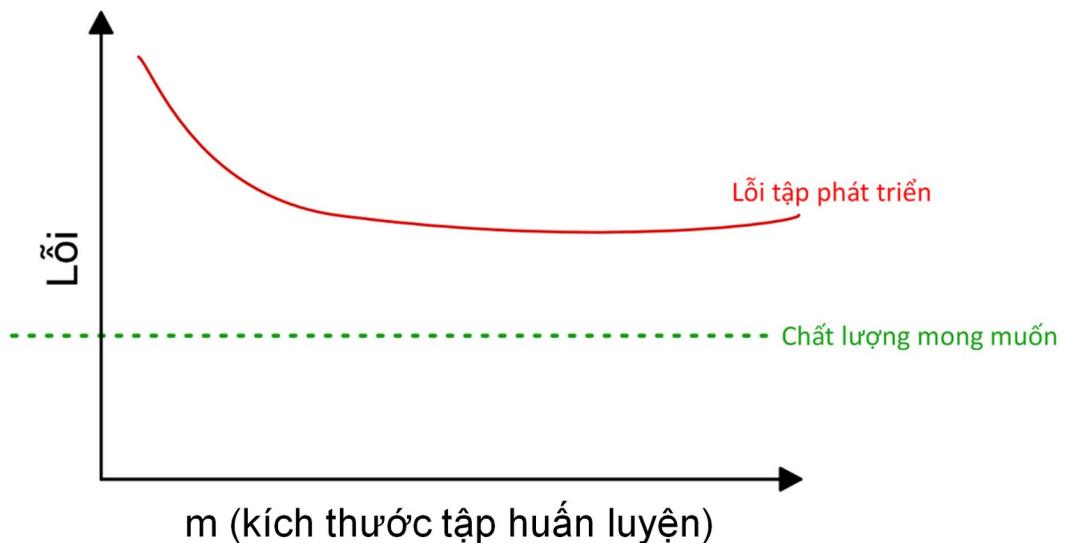
- Nếu chúng ta hy vọng đạt được chất lượng ở cấp độ con người, thì tỷ lệ lỗi của con người là "tỉ lệ lỗi mong muốn".
- Nếu thuật toán học của chúng ta phục vụ một số sản phẩm (ví dụ như cung cấp ảnh mèo), chúng ta có thể có một trực giác về mức chất lượng cần thiết để người dùng có được trải nghiệm tốt nhất.
- Nếu bạn đã làm việc trên một ứng dụng quan trọng trong thời gian dài, thì bạn sẽ có trực giác về mức cải thiện hợp lý có thể đạt được trong quý/năm tới.

Thêm mức chất lượng mong muốn vào đồ thị quá trình học của bạn:



Bạn có thể ngoại suy đường cong đỏ thể hiện "lỗi tập phát triển" để ước tính mức độ cải thiện có thể đạt được so với mức chất lượng mong muốn bằng cách thêm vào dữ liệu. Ví dụ trên cho thấy bạn có thể đạt được mức chất lượng mong muốn bằng cách tăng gấp đôi độ lớn tập huấn luyện.

Tuy nhiên nếu đường cong lỗi tập phát triển đã "nằm ngang" (phẳng), thì bạn có thể hiểu ngay lập tức rằng việc thêm vào dữ liệu cũng sẽ không giúp bạn đạt được mục tiêu:



Do đó nhìn vào đồ thị đường cong học tập có thể giúp bạn tránh khỏi việc dành hàng tháng trời thu thập một lượng dữ liệu lớn gấp đôi, chỉ để nhận ra rằng điều đó là vô ích.

Một nhược điểm của quá trình này là nếu bạn chỉ nhìn vào đường cong lỗi của tập phát triển, thì có thể bạn sẽ khó ngoại suy và dự đoán chính xác vị trí đường cong đỏ khi có thêm dữ liệu. Một đồ thị khác có thể giúp bạn dự đoán sự tác động của việc thêm dữ liệu đó là: đồ thị lỗi tập huấn luyện.

29. Vẽ đồ thị sai số huấn luyện

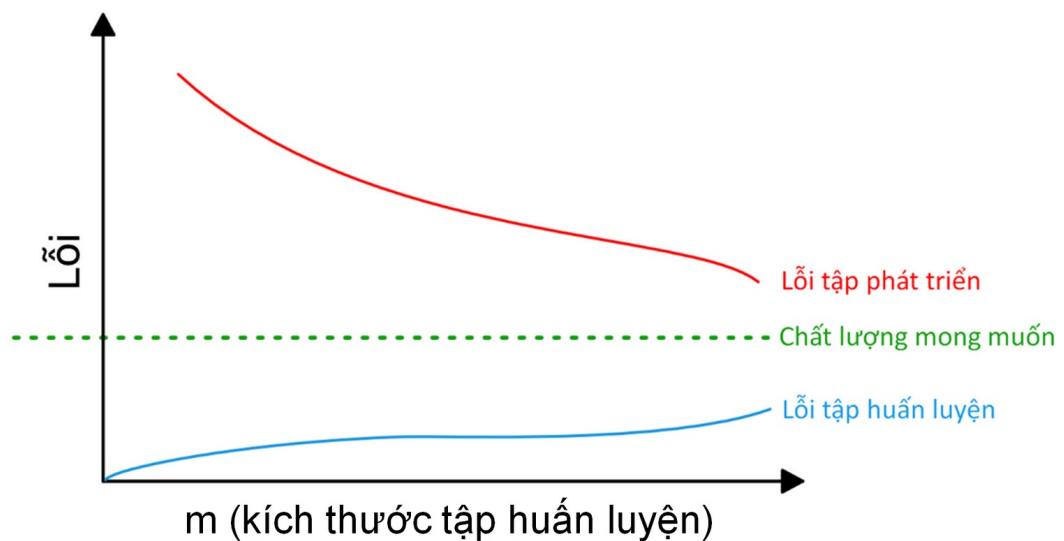
Sai số tập phát triển của bạn (và tập kiểm thử) thường giảm khi kích thước tập huấn luyện tăng lên. Nhưng sai số tập huấn luyện của bạn thường xuyên *tăng* khi kích thước tập huấn luyện tăng.

Chúng ta hãy mô tả ánh hưởng này bằng một ví dụ. Giả sử tập huấn luyện của bạn chỉ 2 gồm mẫu: Một ảnh mèo và một ảnh không phải mèo. Khi đó rất dễ dàng để thuật toán học ghi nhớ cả 2 mẫu trong tập huấn luyện, và nhận 0% sai số huấn luyện. Thậm chí nếu cả 2 mẫu huấn luyện bị gán nhãn sai, vẫn khá dễ dàng cho thuật toán ghi nhớ cả 2 nhãn.

Bây giờ giả định tập huấn luyện có 100 mẫu. Thậm chí một vài mẫu bị gán nhãn sai, hoặc một vài hình ảnh là rất mơ hồ do bị mờ, nên ngay cả con người cũng không thể khẳng định đó là một chú mèo. Có lẽ thuật toán học vẫn có thể "ghi nhớ" được hầu hết tập huấn luyện, nhưng nó khó mà đạt được 100% độ chính xác vào lúc này. Bằng cách gia tăng tập huấn luyện từ 2 lên 100 mẫu, bạn sẽ nhận ra rằng độ chính xác của tập huấn luyện sẽ giảm một ít.

Cuối cùng, giả sử tập huấn luyện có 10,000 mẫu. Trong trường hợp này, sẽ khó hơn cho thuật toán fit hoàn hảo 10,000 mẫu, đặc biệt là nếu một vài mẫu mơ hồ hoặc bị gán nhãn sai. Do đó, thuật toán học của bạn sẽ hoạt động thậm chí là kém hơn trên tập huấn luyện.

Chúng ta hãy thêm một đồ thị sai số huấn luyện vào các hình trước đó:

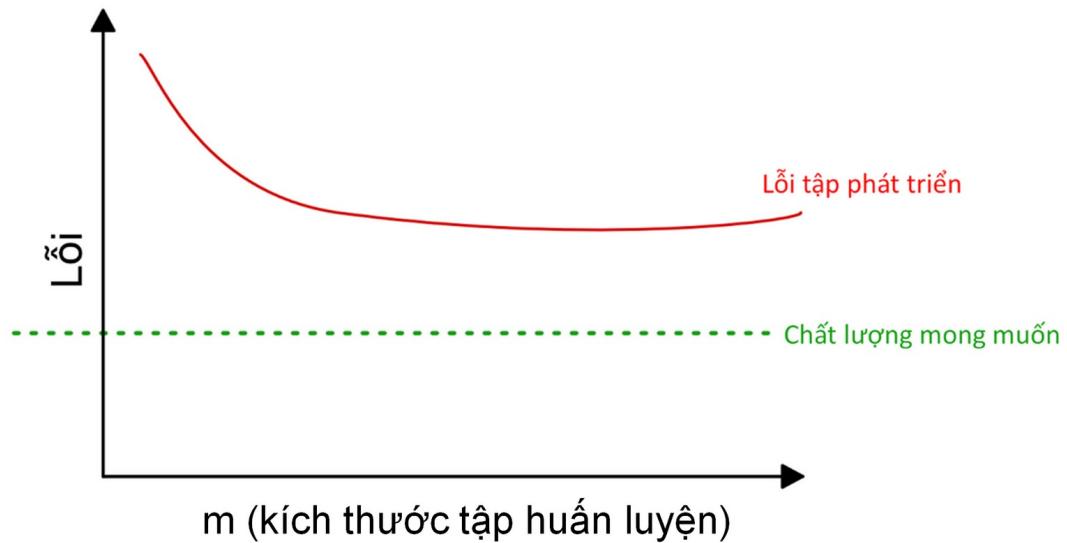


Bạn có thể thấy rằng đồ thị sai số huấn luyện (training error) màu xanh lam tăng theo kích thước của tập huấn luyện.Thêm nữa, thuật toán của bạn thường hoạt động tốt trên tập huấn luyện hơn là tập phát triển; do đó đồ thị sai số tập phát triển hoàn toàn nằm trên đồ thị sai số huấn luyện.

Tiếp theo chúng ta sẽ thảo luận làm thế nào để diễn giải những đồ thị này.

30. Diễn giải đồ thị quá trình học: Độ chêch cao

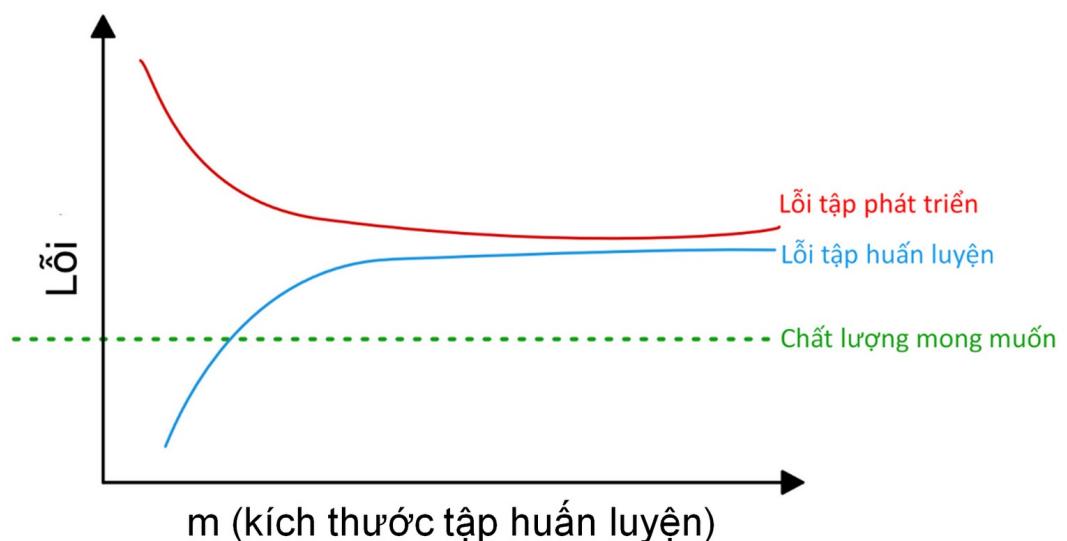
Giả sử đường cong sai số trên tập phát triển có dạng như sau:



Như chúng ta đã thảo luận, nếu đường cong sai số trên tập phát triển đã nằm ngang, việc chỉ thêm dữ liệu sẽ khó có thể đem về hiệu suất ta mong muốn.

Tuy nhiên, sẽ thật khó để biết chính xác ngoại suy đường cong thể hiện sai số trên tập phát triển (màu đỏ) sẽ trông như thế nào. Trong trường hợp tập phát triển nhỏ, việc dự đoán chính xác sẽ càng trở nên khó khăn bởi khi đó đường cong này sẽ có khả năng bị nhiễu.

Giả sử chúng ta thêm đường cong sai số tập huấn luyện vào biểu đồ như hình dưới:



Lúc này, bạn có thể hoàn toàn chắc chắn việc chỉ thêm dữ liệu là không đủ. Tại sao vậy? Hãy nhớ hai nhận định sau:

Khi chúng ta thêm dữ liệu huấn luyện, sai số huấn luyện chỉ có thể tăng lên. Vì vậy, đường cong chỉ sai số huấn luyện (màu xanh)

dương) chỉ có thể giữ nguyên hoặc hướng cao lên. Bởi vậy, đường cong đó chỉ có thể cách xa hơn hiệu suất mong đợi (được thể hiện bởi màu xanh lục).

Đường cong thể hiện sai số tập phát triển (màu đỏ) thường cao hơn so với đường cong thể hiện sai số tập huấn luyện (màu xanh). Vì vậy, việc lấy thêm dữ liệu không thể nào giảm đường cong sai số tập phát triển (màu đỏ) xuống mức hiệu xuất mong muốn khi ngay cả sai số trên tập huấn luyện vẫn còn lớn hơn mức đó.

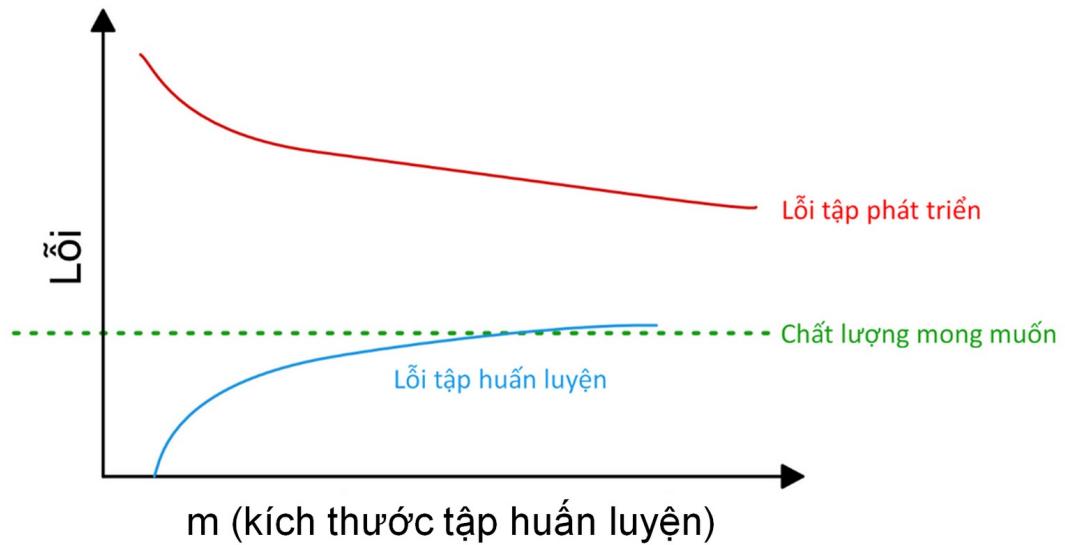
Đánh giá cả đường cong sai số trên tập phát triển lẫn đường cong sai số trên tập huấn luyện trên cùng một biểu đồ giúp những ngoại suy về đường cong sai số tập phát triển có độ tin cậy cao hơn.

Cho mục đích thảo luận, giả sử hiệu suất mong muốn chính là ước lượng của tỉ lệ lỗi tối ưu. Đồ thị trên trở thành ví dụ chuẩn "sách giáo khoa" về hình dáng của một đồ thị quá trình học với độ chệch có thể tránh cao: Khi tập huấn luyện có kích cỡ lớn nhất -- tương ứng với tất cả dữ liệu trong tập huấn luyện - có một khoảng cách lớn giữa sai số huấn luyện và hiệu suất mong muốn. Đây chính là dấu hiệu của độ chệch có thể tránh cao. Ngược lại, lúc này, khoảng cách nhỏ giữa đường cong của tập huấn luyện và đường cong của tập phát triển tương ứng với phuơng sai nhỏ.

Trước đó, chúng ta chỉ đo sai số tập huấn luyện và sai số tập phát triển tại điểm ngoài cùng bên phải của đồ thị, tương ứng với việc sử dụng tất cả dữ liệu trong tập huấn luyện. Biểu diễn đầy đủ đồ thị quá trình học sẽ cho chúng ta một bức tranh tổng thể hơn về chất lượng của những thuật toán trên các kích cỡ tập huấn luyện khác nhau.

31. Giải nghĩa các đồ thị quá trình học: Những trường hợp khác

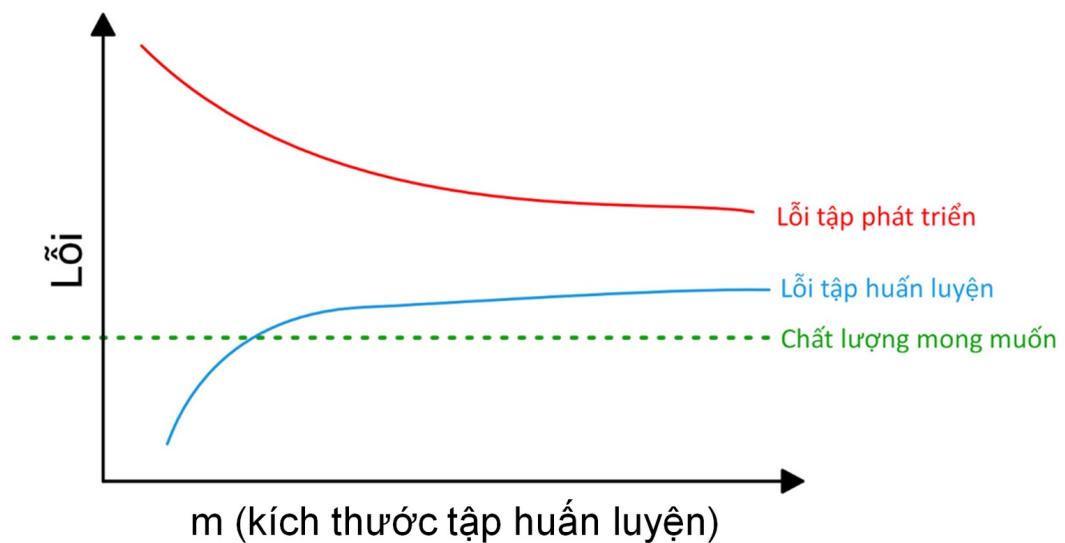
Hãy xem xét đồ thị quá trình học này:



Đồ thị này thể hiện độ chêch lớn, phương sai lớn hay cả hai?

Đường cong lỗi huấn luyện màu xanh lam tương đối thấp và đường cong lỗi phát triển màu đỏ cao hơn nhiều so với lỗi huấn luyện màu xanh lam. Do đó, độ chêch nhỏ, nhưng phương sai lớn. Thêm dữ liệu huấn luyện (nhiều khả năng) sẽ giúp thu hẹp khoảng cách giữa lỗi phát triển và lỗi huấn luyện.

Bây giờ, hãy xem xét đồ thị này:



Lần này, lỗi huấn luyện lớn, vì nó cao hơn nhiều so với mức chất lượng mong muốn. Lỗi phát triển cũng lớn hơn nhiều so với lỗi huấn luyện. Vì vậy, bạn có độ chêch đáng kể và phương sai cũng đáng kể. Bạn sẽ phải tìm cách giảm cả độ chêch và phương sai trong thuật toán của mình.

32. Vẽ đồ thị quá trình học

Giả sử bạn có một tập huấn luyện rất nhỏ gồm 100 mẫu. Bạn huấn luyện thuật toán của mình bằng cách sử dụng một tập hợp con được chọn ngẫu nhiên gồm 10 mẫu, sau đó 20 mẫu, sau đó 30, lên đến 100, tăng số lượng mẫu theo các khoảng mươi. Sau đó, bạn sử dụng 10 điểm dữ liệu này để vẽ đồ thị quá trình học. Bạn có thể thấy rằng đồ thị trông hơi nhiễu (có nghĩa là các giá trị cao hơn/thấp hơn dự kiến) ở kích thước tập huấn luyện nhỏ hơn.

Khi huấn luyện chỉ với 10 mẫu được chọn ngẫu nhiên, bạn có thể không may mắn và có một tập huấn luyện đặc biệt "xấu", chẳng hạn như một tập có nhiều mẫu không rõ ràng/bị gán nhãn sai. Hoặc, bạn có thể gặp may mắn và nhận được một tập huấn luyện đặc biệt "tốt". Có một tập huấn luyện nhỏ đồng nghĩa với lỗi trên tập phát triển và tập huấn luyện có thể dao động ngẫu nhiên.

Nếu ứng dụng học máy của bạn bị lệch nhiều về một lớp: chẳng hạn như nhiệm vụ phân loại mèo mà số mẫu âm (không phải mèo) lớn hơn nhiều số mẫu dương (là mèo). Hoặc, nếu nó có một số lượng lớn các lớp (chẳng hạn như nhận dạng 100 loài động vật khác nhau), khi đó xác suất chọn một tập huấn luyện rất không "mang tính đại diện" hoặc xấu cũng lớn hơn. Ví dụ: nếu 80% mẫu của bạn là mẫu âm ($y = 0$), và chỉ 20% là mẫu dương ($y = 1$), thì có khả năng một tập huấn luyện gồm 10 mẫu chỉ chứa các mẫu âm, thì rất khó để thuật toán học được điều gì đó có ý nghĩa.

Nếu nhiễu trong đồ thị quá trình học khiến bạn khó nhìn thấy xu hướng thực sự, thì đây là hai giải pháp:

- Thay vì chỉ huấn luyện một mô hình trên 10 mẫu, hãy chọn ra ngẫu nhiên một vài (ví dụ 3-10) tập huấn luyện khác nhau gồm 10 mẫu bằng cách lấy mẫu có hoàn lại [10] từ bộ dữ liệu 100 mẫu ban đầu. Huấn luyện mô hình khác nhau trên mỗi tập đó và tính toán lỗi huấn luyện, lỗi phát triển của từng mô hình sau khi huấn luyện xong. Tính toán và vẽ đồ thị lỗi trung bình trên tập huấn luyện và lỗi trung bình trên tập phát triển.
- Nếu tập huấn luyện của bạn bị lệch về một lớp, hoặc nếu nó có nhiều lớp, hãy chọn một tập hợp con "cân bằng" thay vì chọn ngẫu nhiên 10 trên 100 mẫu huấn luyện. Ví dụ, bạn có thể chắc chắn rằng 2/10 các mẫu là các mẫu dương và 8/10 là âm. Tổng quát hơn, bạn có thể đảm bảo tỷ lệ các mẫu trong mỗi lớp càng gần với tỉ lệ trong tập huấn luyện ban đầu.

Nếu bạn đã thử vẽ các đồ thị quá trình học và kết luận rằng các đường cong quá nhiễu để nhìn thấy các xu hướng cơ bản, thì hãy sử dụng một trong những kỹ thuật trên. Nếu tập huấn luyện của bạn có quy mô lớn, ví dụ hơn 10.000 mẫu, và phân phối lớp của bạn không bị lệch nhiều, có lẽ bạn không cần các kỹ thuật này.

Cuối cùng, vẽ đồ thị quá trình học có thể tốn kém về mặt tính toán: Ví dụ: bạn có thể phải huấn luyện mười mô hình với 1.000, rồi 2.000, cho đến 10.000 mẫu. Huấn luyện các mô hình với các bộ dữ liệu nhỏ nhanh hơn nhiều so với huấn luyện các mô hình với các bộ dữ liệu lớn. Do đó, thay vì cách đều các kích thước tập huấn luyện theo tỷ lệ tuyến tính như trên, bạn có thể huấn luyện các mô hình với 1.000, 2.000, 4.000, 6.000 và 10.000 mẫu. Điều này vẫn sẽ cung cấp cho bạn một cảm giác rõ ràng về các xu hướng trong các đồ thị quá trình học. Tất nhiên, kỹ thuật này chỉ thích hợp nếu chi phí tính toán để huấn luyện tất cả các mô hình bổ sung là đáng kể.

CHÚ THÍCH

[10] Ở đây, việc lấy mẫu có *hoàn lại* có nghĩa là: Bạn sẽ chọn ngẫu nhiên 10 mẫu khác nhau trong số 100 để tạo thành tập huấn luyện đầu tiên của mình. Sau đó để tạo tập huấn luyện thứ hai, bạn sẽ lại lấy 10 mẫu ngẫu nhiên trong 100 mẫu ban đầu. Vì vậy, có thể một mẫu cụ thể xuất hiện trong cả tập huấn luyện thứ nhất và thứ hai. Ngược lại, nếu bạn lấy mẫu *không hoàn lại*, tập huấn luyện thứ hai sẽ chỉ được chọn từ 90 mẫu không được chọn ở lần đầu tiên. Trong thực tế, sử dụng lấy mẫu có hoàn lại hoặc không hoàn lại không tạo ra sự khác biệt lớn, nhưng lấy mẫu có hoàn lại là cách làm phổ biến.

Phần 5: So sánh với chất lượng mức con người

33. Tại sao chúng ta so sánh với chất lượng mức con người?

Nhiều hệ thống học máy hướng tới tự động hóa những thứ con người làm tốt. Ví dụ như nhận dạng hình ảnh, nhận dạng giọng nói, và phân loại thư rác. Các thuật toán học cũng đã được cải thiện rất nhiều, đến mức vượt qua chất lượng mức con người trong ngày càng nhiều tác vụ.

Hơn nữa, có rất nhiều lý do khiến việc xây dựng một hệ thống ML dễ dàng hơn nếu bạn đang giải quyết một tác vụ con người có thể làm tốt:

- a. **Dễ dàng thu thập dữ liệu từ người gán nhãn** Ví dụ, con người nhận diện hình ảnh mèo tốt nên việc mọi người cung cấp nhãn có độ chính xác cao cho thuật toán học tập là điều đơn giản.
- b. **Phân tích lỗi có thể dựa vào trực giác của con người** Giả sử rằng một thuật toán nhận dạng giọng nói làm tệ hơn so với con người. Giả dụ nó ghi nhầm một đoạn âm thanh thành "This recipe calls for a pear of apples," (dịch là "công thức nấu ăn này cần một quả lê của táo") gây ra lỗi tại từ "pair" trở thành "pear". Bạn có thể dựa vào trực giác và cố gắng hiểu thông tin nào một người sử dụng để thu được bản ghi thoại chuẩn, và dùng thông tin này để điều chỉnh thuật toán.
- c. **Sử dụng chất lượng mức con người để ước tính tỷ lệ lỗi tối ưu cũng như đặt ra một "tỷ lệ lỗi mong muốn."** Giả sử thuật toán của bạn trả về 10% lỗi trong một tác vụ, nhưng một người chỉ lỗi 2%. Dựa vào đó, chúng ta biết rằng tỷ lệ lỗi tối ưu là 2% hoặc thấp hơn và độ chênh có thể tránh ít nhất là 8%. Vì vậy, bạn nên thử các kỹ thuật giảm độ chênh.

Mặc dù mục số 3 nghe có vẻ không quan trọng, tôi thấy rằng việc xác định mục tiêu tỷ lệ lỗi hợp lý sẽ giúp đẩy nhanh tiến độ của nhóm. Việc biết thuật toán của bạn có độ chênh cao nhưng có thể tránh được là vô cùng có giá trị và mở ra nhiều tùy chọn để thử.

Có những tác vụ mà ngay cả con người cũng không giải. Ví dụ, chọn một cuốn sách để giới thiệu cho bạn; hoặc chọn một quảng cáo để hiển thị cho người dùng trên một trang web; hoặc dự đoán thị trường chứng khoán. Máy tính đã trở nên hiệu quả hơn hầu hết mọi người trong những tác vụ này. Với các ứng dụng này, chúng ta gặp phải các vấn đề sau:

- Việc lấy nhãn khó hơn. Ví dụ, người ghi nhãn khó có thể dán nhãn một cơ sở dữ liệu người dùng với danh sách gợi ý sách tối ưu. Nếu bạn vận hành một trang web hoặc ứng dụng bán sách, bạn có thể lấy dữ liệu bằng cách hiển thị sách cho người dùng và xem những gì họ mua. Nếu bạn không vận hành một trang web như vậy, bạn cần tìm những cách sáng tạo hơn để lấy dữ liệu.
- Trực giác của con người khó dựa vào hơn. Ví dụ, khá nhiều người không thể dự đoán được thị trường chứng khoán. Vì vậy, nếu thuật toán dự đoán cổ phiếu của chúng ta không tốt hơn đoán ngẫu nhiên, thật khó để tìm ra cách cải thiện nó.
- Thật khó để biết tỷ lệ lỗi tối ưu và tỷ lệ lỗi mong muốn hợp lý là gì. Giả sử bạn đã có một hệ thống giới thiệu sách đang hoạt động khá tốt. Làm thế nào để bạn biết nó có thể cải thiện bao nhiêu nếu không có giải pháp cấp con người?

34. Cách xác định chất lượng mức con người

Giả sử bạn đang làm việc trên một ứng dụng hình ảnh y tế tự động đưa ra chẩn đoán từ hình ảnh X quang. Một người bình thường không có nền tảng y khoa nào ngoài một số đào tạo cơ bản có thể đạt được mức 15% lỗi trong tác vụ này. Một bác sĩ trẻ mới ra trường đạt được mức 10% lỗi. Một bác sĩ dày dặn kinh nghiệm đạt được mức 5% lỗi. Và một nhóm nhỏ các bác sĩ trao đổi và tranh luận mỗi hình ảnh đạt được mức 2% lỗi. Vậy cái nào trong những tỉ lệ lỗi này xác định "chất lượng mức con người"?

Trong trường hợp này, tôi sẽ sử dụng 2% làm "chất lượng mức con người" cho tỉ lệ lỗi tối ưu của chúng ta. Bạn cũng có thể đặt 2% làm mức chất lượng mong muốn vì nó thỏa mãn cả ba lý do để so sánh với chất lượng mức con người mà chúng ta đã nói ở chương trước:

- **Dữ liệu được lấy dễ dàng từ người gán nhãn.** Bạn có thể nhờ một nhóm bác sĩ cung cấp nhãn cho bạn với tỉ lệ lỗi 2%.
- **Phân tích lỗi có thể dựa vào trực giác.** Bằng cách thảo luận với một nhóm các bác sĩ, bạn có thể dựa trên được trực giác của họ để đưa ra kết luận về các bức ảnh.
- **Sử dụng chất lượng mức con người để ước tính tỉ lệ lỗi tối ưu cũng như đặt ra "tỉ lệ lỗi mong muốn" khả thi** Đó là điều khá hợp lý khi sử dụng mức 2% lỗi làm ước lượng của chúng ta về tỉ lệ lỗi tối ưu. Tỉ lệ lỗi tối ưu có thể thậm chí thấp hơn 2%, nhưng không thể cao hơn, vì một nhóm bác sĩ có thể đạt được mức 2% lỗi. Ngược lại, sẽ không hợp lý khi sử dụng 5% hoặc 10% làm ước tính tỉ lệ lỗi tối ưu, vì chúng ta biết các ước tính này đang quá cao.

Khi nói đến việc lấy dữ liệu được gán nhãn, bạn có thể không muốn thảo luận về mọi bức ảnh với toàn bộ đội ngũ bác sĩ vì thời gian của họ rất đáng giá. Có lẽ bạn có thể có một bác sĩ trẻ mới ra trường gán nhãn cho phần lớn các trường hợp và chỉ dành những trường hợp khó hơn cho các bác sĩ có kinh nghiệm hơn hoặc cho đội ngũ bác sĩ.

Nếu hệ thống hiện tại của bạn có mức 40% lỗi, vậy thì nó không có ý nghĩa nhiều giữa việc bạn sử dụng một bác sĩ mới ra trường (10% lỗi) hoặc một bác sĩ có kinh nghiệm (5% lỗi) để gán nhãn và đưa ra những phán đoán trực giác cho dữ liệu của bạn. Nhưng nếu hệ thống của bạn đang có 10% lỗi thì việc xác định chất lượng mức con người ở mức 2% sẽ cho bạn các công cụ tốt hơn để tiếp tục cải thiện hệ thống của mình.

35. Vượt qua chất lượng mức con người

Bạn đang làm về nhận dạng giọng nói và bạn có một tập dữ liệu là các đoạn âm thanh. Giả sử, tập dữ liệu của bạn có nhiều âm thanh nhiễu mà thậm chí con người còn mắc phải 10% lỗi. Giả sử, hệ thống của bạn đã đạt được 8% lỗi. Liệu bạn có thể sử dụng bất kỳ kỹ thuật nào trong ba kỹ thuật được mô tả trong Chương 33 để tiếp tục tiến bộ nhanh chóng?

Nếu bạn có thể xác định một tập dữ liệu con mà con người vượt qua được hệ thống của bạn một cách đáng kể, thì bạn vẫn có thể sử dụng các kỹ thuật đó để thúc đẩy tiến trình nhanh chóng. Ví dụ, giả sử hệ thống của bạn tốt hơn nhiều so với con người trong việc nhận dạng giọng nói trong âm thanh nhiễu, nhưng con người vẫn làm tốt hơn trong việc ghi lại lời nói rất nhanh.

Đối với tập dữ liệu con với lời nói nhanh:

- a. Bạn vẫn có thể lấy được bản ghi thoại từ con người với chất lượng cao hơn so với đầu ra thuật toán của bạn.
- b. Bạn có thể dựa vào trực giác để hiểu lý do tại sao họ nghe chính xác một phát ngôn nhanh khi hệ thống của bạn chưa thể.
- c. Bạn có thể dùng chất lượng mức con người trên lời nói nhanh như một mục tiêu chất lượng mong muốn.

Tổng quát hơn, miễn là có các mẫu trong tập phát triển mà con người đúng và thuật toán của bạn sai, thì rất nhiều kỹ thuật được mô tả trước đây sẽ áp dụng được. Điều này đúng ngay cả khi chất lượng của bạn (tính trung bình trên toàn bộ tập phát triển/kiem tra) đã vượt qua chất lượng mức con người.

Có nhiều ứng dụng học máy quan trọng trong đó máy đã vượt qua chất lượng mức con người. Ví dụ, máy làm tốt hơn trong việc dự đoán xếp hạng phim, sẽ mất bao lâu để một chiếc xe giao hàng lái xe đi đâu đó, hoặc có chấp nhận hồ sơ vay vốn hay không. Chỉ một tập con những kỹ thuật là áp dụng được khi mà con người còn gặp khó khăn trong việc xác định các mẫu nào mà thuật toán còn rõ ràng đang làm sai. Do đó, tiến độ thường chậm hơn đối với các vấn đề trong đó máy móc đã vượt qua chất lượng mức con người, và ngược lại, nhanh hơn khi máy móc vẫn đang cố gắng bắt kịp con người.

Phần 6: Huấn luyện và kiểm tra trên các phân phối khác nhau

36. Khi nào bạn nên huấn luyện và kiểm tra trên những phân phối khác nhau

Người dùng của ứng dụng ảnh mèo của bạn đã đăng tải 10.000 tấm ảnh mà sau đó bạn đã gán nhãn chúng có mèo hay không một cách thủ công. Bạn cũng có một tập ảnh lớn hơn gồm 200.000 tấm bạn đã tải về trên mạng. Bạn nên tạo tập huấn luyện/phát triển/kiểm tra như thế nào?

Vì 10.000 tấm ảnh của người dùng phản ánh mật thiết phân bố xác suất thật của dữ liệu mà bạn muốn làm tốt, bạn có thể sử dụng chúng cho tập phát triển và kiểm tra. Nếu bạn đang huấn luyện một thuật toán deep learning "đổi" dữ liệu, bạn có thể đưa thêm 200.000 tấm ảnh trên mạng cho việc huấn luyện. Do vậy, tập huấn luyện và tập phát triển/kiểm tra sẽ đến từ những phân phối khác nhau. Điều này ảnh hưởng thế nào tới công việc của bạn?

Thay vì phân chia dữ liệu của chúng ta ra thành tập huấn luyện/phát triển/kiểm tra, chúng ta có thể lấy hết 210.000 tấm ảnh mà ta có, và trộn một cách ngẫu nhiên vào các tập huấn luyện/phát triển/kiểm tra. Trong trường hợp này, tất cả dữ liệu đều đến từ cùng một phân phối. Nhưng tôi không ủng hộ phương pháp này, bởi vì khoảng $205.000/210.000 \approx 97,6\%$ dữ liệu phát triển/kiểm tra đến từ những ảnh trên mạng nên nó không phản ánh được phân phối thật mà bạn muốn làm tốt trên nó. Hãy nhớ lời khuyên khi chọn tập phát triển/kiểm tra:

Chọn tập phát triển và kiểm tra phản ánh dữ liệu bạn kỳ vọng sẽ có trong tương lai và muốn làm tốt trên nó.

Đa số các tài liệu học thuật về machine learning đều giả định tập huấn luyện, tập phát triển và tập kiểm tra đến từ cùng một phân phối [11]. Trong những ngày đầu của học máy, dữ liệu rất khan hiếm. Ta thường chỉ có một bộ dữ liệu được lấy ra từ một phân bố xác suất nào đó. Bởi vậy, ta thường phân tách một cách ngẫu nhiên dữ liệu đó thành tập huấn luyện/phát triển/kiểm tra, và việc mặc định là tất cả các dữ liệu đến từ cùng một nguồn thường được thỏa mãn.

Nhưng trong thời đại của dữ liệu lớn, ta nay đã có thể tiếp cận với những tập huấn luyện khổng lồ, như là những tấm ảnh mèo trên mạng. Kể cả khi tập huấn luyện đến từ một phân phối khác với tập phát triển/kiểm tra, ta vẫn muốn sử dụng chúng cho quá trình học bởi vì chúng có thể cung cấp rất nhiều thông tin.

Với ví dụ về bộ nhận diện mèo, thay vì bỏ toàn bộ 10.000 tấm ảnh do người dùng đăng tải vào tập phát triển/kiểm tra, thay vào đó ta chỉ bỏ 5.000 tấm vào tập phát triển/kiểm tra. Còn lại 5.000 tấm do người dùng đăng tải, ta có thể bỏ vào tập huấn luyện. Bằng cách này, tập huấn luyện gồm 205.000 mẫu sẽ chứa một vài dữ liệu đến từ phân phối của tập phát triển/kiểm tra cùng với 200.000 tấm từ internet. Chúng ta sẽ bàn thêm trong chương sau vì sao phương pháp này lại có ích.

Hãy xem xét một ví dụ thứ hai. Giả sử bạn đang xây dựng một hệ thống nhận diện giọng nói để phiên âm địa chỉ đường cho một ứng dụng bản đồ/định vị trên di động điều khiển bằng giọng nói. Bạn có 20.000 mẫu của người dùng đang nói địa chỉ đường. Nhưng bạn cũng có 500.000 mẫu là những bản ghi âm khác của người dùng đang nói về những chủ đề khác. Bạn có thể lấy 10.000 mẫu nói về địa chỉ đường cho tập phát triển/kiểm tra, và sử dụng 10.000 mẫu còn lại, cộng thêm 500.000 mẫu, cho việc huấn luyện.

Chúng ta sẽ tiếp tục giả định rằng dữ liệu phát triển và dữ liệu kiểm tra của bạn đến từ cùng một phân phối. Nhưng cũng quan trọng khi hiểu rằng phân bố tập huấn luyện và phát triển/kiểm tra khác nhau sẽ dẫn tới một vài thách thức đặc biệt.

CHÚ THÍCH:

[11] Có một vài nghiên cứu khoa học về việc huấn luyện và kiểm tra trên các phân phối khác nhau. Những ví dụ bao gồm "thích ứng miền", "học chuyển tiếp" và "học đa nhiệm". Tuy nhiên vẫn còn một khoảng cách lớn giữa lý thuyết và thực hành. Nếu bạn huấn luyện trên bộ dữ liệu A và kiểm tra trên một vài kiểu dữ liệu rất khác B, may mắn sẽ có ảnh hưởng rất lớn tới việc thuật toán của bạn hoạt động tốt thế nào. (Ở đây, "may mắn" bao gồm những đặc trưng được tạo thủ công cho một bài toán nhất định của người làm nghiên cứu, cũng như một vài nhân tố khác mà chúng ta vẫn chưa hiểu rõ.) Điều này làm cho nghiên cứu khoa học của việc huấn luyện và kiểm tra trên những phân phối khác nhau khó có thể hoàn thành một cách có hệ thống.

37. Làm sao để quyết định có nên sử dụng toàn bộ dữ liệu?

Giả sử tập huấn luyện của bộ nhận diện mèo của bạn bao gồm 10.000 hình ảnh do người dùng tải lên. Dữ liệu này đến từ cùng một phân phối chia ra một tập phát triển/kiểm tra riêng biệt, và đại diện cho phân phối mà bạn muốn làm tốt trên đó. Bạn cũng có thêm 20.000 hình ảnh được tải xuống từ internet. Bạn có nên cung cấp tất cả $20.000 + 10.000 = 30.000$ ảnh cho thuật toán học dưới dạng tập huấn luyện, hay nên loại bỏ 20.000 ảnh từ internet để tránh làm chệch đi thuật toán học của bạn?

Khi sử dụng các thê hệ thuật toán học trước đó (như các tính năng nhận diện hình ảnh được thiết kế thủ công, theo sau là một phân loại tuyến tính đơn giản), có một rủi ro thực sự là việc hợp nhất cả hai loại dữ liệu sẽ khiến bạn đạt chất lượng tệ hơn. Do đó, một số kỹ sư sẽ cảnh báo bạn không thêm vào 20.000 ảnh internet.

Nhưng ngày nay, với các thuật toán học linh hoạt, mạnh mẽ -- chẳng hạn như các mạng nơ-ron lớn -- rủi ro này đã giảm đi rất nhiều. Nếu bạn có thể đủ khả năng để xây dựng một mạng nơ-ron với số lượng đơn vị/tầng ẩn đủ lớn, bạn có thể thêm 20.000 hình ảnh vào tập huấn luyện của mình một cách an toàn. Việc thêm hình ảnh có nhiều khả năng để tăng chất lượng của bạn.

Nhận định này dựa trên thực tế là có một số ảnh xạ $x \rightarrow y$ hoạt động tốt cho cả hai loại dữ liệu. Nói cách khác, vẫn tồn tại một số hệ thống nhận đầu vào là ảnh internet hoặc ảnh ứng dụng di động và dự đoán một cách đáng tin cậy nhãn ảnh, ngay cả khi không biết nguồn gốc của nó.

Thêm vào 20.000 hình ảnh bổ sung có những ảnh hưởng sau:

- a. Nó cung cấp cho mạng nơ-ron của bạn nhiều mẫu hơn về những gì giống/không giống mèo. Điều này rất hữu ích, vì hình ảnh internet và hình ảnh ứng dụng di động do người dùng tải lên có chung một số điểm tương đồng. Mạng neural của bạn có thể áp dụng một số kiến thức thu được từ hình ảnh internet vào hình ảnh ứng dụng di động.
- b. Nó buộc mạng nơ-ron phải sử dụng một số nguồn lực của nó để tìm hiểu về các thuộc tính dành riêng cho hình ảnh trên internet (chẳng hạn như độ phân giải cao hơn, các phân phối khác nhau về cách các hình ảnh được đóng khung, v.v.) Nếu các thuộc tính này khác nhiều so với hình ảnh ứng dụng di động, nó sẽ "sử dụng hết" một số khả năng đại diện của mạng nơ-ron. Vì vậy khả năng nhận biết dữ liệu trích xuất từ phân phối ảnh ứng dụng di động sẽ ít hơn, trong khi đó mới là điều bạn thực sự quan tâm. Trên lý thuyết, điều này có thể gây tổn thương đến chất lượng thuật toán của bạn.

Để mô tả ảnh hưởng thứ hai theo các thuật ngữ khác nhau, chúng ta có thể chuyển sang nhân vật hư cấu Sherlock Holmes, người nói rằng bộ não của bạn giống như một căn gác; nó chỉ có một không gian hữu hạn. Anh ta nói rằng "mỗi lần bổ sung kiến thức, bạn sẽ quên đi những điều mà bạn biết trước đây. Do đó, điều quan trọng bậc nhất là không để những thông tin vô dụng lấn át những thông tin hữu ích." [12]

May mắn thay, nếu bạn có khả năng tính toán cần thiết để xây dựng một mạng nơ-ron đủ lớn -- ví dụ, một căn gác đủ lớn -- thì đây không phải là một vấn đề nghiêm trọng. Bạn có đủ năng lực để học từ cả hình ảnh internet và từ hình ảnh ứng dụng di động mà không có sự cạnh tranh về dung lượng giữa hai loại dữ liệu. "Bộ não" của thuật toán đủ lớn để bạn không phải lo lắng về việc hết không gian căn gác.

Nhưng nếu bạn không có một mạng nơ-ron đủ lớn (hoặc một thuật toán học rất linh hoạt khác), thì bạn nên chú ý hơn đến dữ liệu huấn luyện phù hợp với việc phân phối tập hợp phát triển/kiểm tra của bạn.

Nếu bạn nghĩ rằng bạn có dữ liệu không giá trị, bạn nên loại bỏ dữ liệu đó vì lý do khả năng tính toán. Ví dụ: giả sử tập huấn luyện/kiểm tra của bạn chứa chủ yếu là hình ảnh thông thường về con người, địa điểm, địa danh, động vật. Giả sử bạn cũng có một bộ sưu tập lớn bản scan các tài liệu lịch sử:



Convention relative à la Tour Eiffel.

Entre Monsieur Édouard Lockroy,
Ministre du Commerce et de l'Industrie, Commissaire Général de l'Exposition Universelle de
1889, agissant au nom de l'Etat;

Monsieur Eugène Toubelle,
Président de la Seine, agissant au nom de la Ville
de Paris, ainsi qu'il y est autorisé par les
délibérations du Conseil Municipal des 22
Octobre et 28 Décembre 1886 et celles dans les
limites fixées par ces délibérations, qui
restent annexées aux présentes;

Et Monsieur Eiffel, Ingénieur-
Constructeur, demeurant à Levallois-Perret, rue
Touquet, N° 42; agissant en son nom personnel;

d'autre part.

Ont été faites les conventions suivantes:

Article premier.

M. Eiffel s'engage ouvert le Ministre
du Commerce & de l'Industrie, Commissaire Général

Những tài liệu này không có gì giống như một con mèo. Chúng cũng trông hoàn toàn không giống như tập phân phối phát triển/kiểm tra của bạn. Hoàn toàn không có ý nghĩa gì khi giữ lại dữ liệu này để làm mẫu thử ám tính, vì lợi ích từ ảnh hưởng đầu tiên ở trên là không đáng kể -- hầu như mạng nơ-ron của bạn không thể học được từ dữ liệu này để có thể áp dụng cho phân phối tập phát triển/kiểm tra của bạn. Giữ lại chúng sẽ lãng phí tài nguyên tính toán và khả năng đại diện của mạng nơ-ron.

CHÚ THÍCH:

[12] Một nghiên cứu về Scarlet của Arthur Conan Doyle

38. Làm thế nào để quyết định có nên bao gồm dữ liệu không nhất quán

Giả sử bạn muốn tìm hiểu để dự đoán giá nhà ở Thành phố New York. Với kích thước của một ngôi nhà (đặc trưng đầu vào x), bạn muốn dự đoán mức giá (nhãn mục tiêu y).

Giá nhà ở Thành phố New York rất cao. Giả sử bạn có bộ dữ liệu thứ hai về giá nhà đất ở Detroit, Michigan, nơi giá nhà đất thấp hơn nhiều. Bạn có nên bao gồm dữ liệu này trong tập huấn luyện của bạn không?

Với cùng kích thước x, giá của ngôi nhà y rất khác nhau tùy thuộc vào việc nó ở Thành phố New York hay ở Detroit. Nếu bạn chỉ quan tâm đến việc dự đoán giá nhà ở Thành phố New York, việc đặt hai bộ dữ liệu lại với nhau sẽ làm tổn hại đến hiệu suất của bạn. Trong trường hợp này, tốt hơn hết là bỏ qua dữ liệu không nhất quán Detroit. [13]

Ví dụ về Thành phố New York và Detroit này khác gì so với ví dụ về ảnh mèo từ ứng dụng di động và ảnh trên internet?

Ví dụ về ảnh mèo khác thường hợp trên bởi vì, với một ảnh đầu vào x, ta có thể dự đoán một cách đáng tin cậy nhãn y liệu có mèo trong ảnh hay không mà không cần biết hình ảnh đó là từ internet hay từ ứng dụng di động. Nghĩa là, có một hàm $f(x)$ ảnh xà đáng tin cậy từ đầu vào x đến đầu ra mục tiêu y, ngay cả khi không biết nguồn gốc của x. Do đó, nhiệm vụ nhận dạng hình ảnh từ internet là nhiệm vụ nhất quán với nhiệm vụ nhận dạng hình ảnh từ ứng dụng di động. Điều này có nghĩa là có rất ít nhược điểm (ngoài chi phí tính toán) khi bao gồm tất cả các dữ liệu và một số nhược điểm đáng kể có thể có. Ngược lại, dữ liệu của Thành phố New York và Detroit, Michigan không nhất quán. Cho cùng một x (kích thước của ngôi nhà), giá nhà rất khác nhau tùy thuộc vào vị trí ngôi nhà.

CHÚ THÍCH:

[13] Có một cách để giải quyết vấn đề dữ liệu Detroit không nhất quán với dữ liệu của Thành phố New York, đó là thêm một đặc trưng biểu diễn thành phố. Cho một đầu vào x—với đặc trưng biểu diễn thành phố—giá trị mục tiêu của y bây giờ không mập mờ nữa. Tuy nhiên, trong thực tế tôi không thấy điều này được thực hiện thường xuyên.

39. Đánh trọng số dữ liệu

Giả sử bạn có 200.000 hình ảnh từ internet và 5.000 hình ảnh từ người dùng ứng dụng di động của bạn. Tỷ lệ kích thước giữa các bộ dữ liệu này là 40:1. Về lý thuyết, miễn là bạn xây dựng một mạng nơ-ron khổng lồ và huấn luyện nó đủ lâu trên tất cả 205.000 hình ảnh thì sẽ không có vấn đề gì khi cố gắng làm cho thuật toán hoạt động tốt trên cả hình ảnh từ internet và hình ảnh từ di động.

Nhưng trên thực tế, việc có hình ảnh từ internet gấp 40 lần so với hình ảnh từ ứng dụng di động có thể nghĩa là bạn cần phải sử dụng 40 lần (hoặc nhiều hơn) tài nguyên tính toán để mô hình hóa cả hai, so với nếu bạn chỉ đào tạo trên 5.000 hình ảnh.

Nếu bạn không có tài nguyên tính toán khổng lồ, bạn có thể gán trọng số thấp hơn nhiều cho các hình ảnh từ internet như một cách thỏa hiệp.

Ví dụ, giả sử mục tiêu tối ưu của bạn là sai số bình phương (Đây không phải là một lựa chọn tốt cho một tác vụ phân loại, nhưng nó sẽ đơn giản hóa lời giải thích của chúng ta.) Vì vậy, thuật toán học tập của chúng ta cố gắng tối ưu hóa:

$$\min_{\theta} \sum_{(x,y) \in \text{MobileImg}} (h_{\theta}(x) - y)^2 + \sum_{(x,y) \in \text{InternetImg}} (h_{\theta}(x) - y)^2$$

Tổng đầu tiên phía trên là trên 5.000 hình ảnh từ di động và tổng thứ hai là trên 200.000 hình ảnh từ internet. Bạn cũng có thể tối ưu với một tham số bổ sung β :

$$\min_{\theta} \sum_{(x,y) \in \text{MobileImg}} (h_{\theta}(x) - y)^2 + \beta \sum_{(x,y) \in \text{InternetImg}} (h_{\theta}(x) - y)^2$$

Nếu bạn chọn $\beta = 1/40$, thuật toán sẽ gán trọng số tương đương cho 5.000 hình ảnh từ di động và 200.000 hình ảnh từ internet. Bạn cũng có thể chọn các giá trị khác cho β , có thể bằng cách điều chỉnh theo tập phát triển.

Khi giảm trọng số các hình ảnh bổ sung từ Internet, bạn không cần phải xây dựng một mạng nơ-ron khổng lồ để đảm bảo thuật toán thực hiện tốt cả hai loại tác vụ. Việc đánh lại trọng số này chỉ cần thiết khi bạn nghĩ ngờ dữ liệu bổ sung (Hình ảnh từ Internet) có phân phối rất khác so với tập phát triển/ tập kiểm tra, hoặc nếu dữ liệu bổ sung lớn hơn nhiều so với dữ liệu mà có cùng phân phối với tập phát triển/ tập kiểm tra (hình ảnh từ di động).

40. Tổng quát hóa từ tập huấn luyện đến tập phát triển

Giả sử bạn đang áp dụng học máy cho một ứng dụng mà phân phối của tập huấn luyện và tập phát triển khác nhau. Tập huấn luyện chứa ảnh Internet + ảnh Điện Thoại trong khi tập phát triển/kiểm tra chỉ chứa ảnh Điện Thoại. Tuy nhiên, thuật toán của bạn không hoạt động tốt: Lượng lỗi trên tập phát triển/kiểm tra cao hơn nhiều con số mà bạn muốn. Dưới đây là một số khả năng có thể gây ra vấn đề trên:

- a. Thuật toán không hoạt động tốt trên tập huấn luyện. Đây là vấn đề của độ chênh cao (tránh được) trên phân phối của tập huấn luyện.
- b. Thuật toán thể hiện tốt trên tập huấn luyện, nhưng không tổng quát hóa tốt trên dữ liệu chưa thấy trích xuất *từ cùng phân phối với tập huấn luyện*. Trường hợp này là phương sai cao.
- c. Thuật toán tổng quát hóa tốt với dữ liệu mới trích xuất từ cùng phân phối với tập huấn luyện, nhưng không tốt với dữ liệu trích xuất từ phân phối của tập phát triển/kiểm tra. Chúng ta gọi vấn đề này là **dữ liệu không tương đồng** bởi dữ liệu của tập huấn luyện khớp kém so với dữ liệu của tập phát triển/kiểm tra.

Ví dụ, giả sử con người đạt chất lượng hoàn hảo trong việc nhận dạng mèo. Thuật toán của bạn đạt được các kết quả như sau:

- 1% lỗi trên tập huấn luyện
- 1.5% lỗi trên dữ liệu trích xuất từ cùng phân phối với tập huấn luyện mà thuật toán chưa thấy trước đó
- 10% lỗi trên tập phát triển

Trường hợp này bạn rõ ràng có vấn đề về dữ liệu không tương đồng. Để khắc phục, bạn có thể cố gắng xử lý dữ liệu huấn luyện sao cho giống hơn với dữ liệu trên tập phát triển/kiểm tra. Chúng ta sẽ bàn luận các kỹ thuật xử lý vấn đề này về sau.

Để chẩn đoán mức độ tác động tới thuật toán từ mỗi vấn đề 1-3 ở trên, sẽ rất hữu ích khi bạn có một bộ dữ liệu khác. Cụ thể, thay vì áp dụng thuật toán với toàn bộ dữ liệu huấn luyện, bạn có thể chia nó thành hai tập con: Tập huấn luyện thực tế mà thuật toán sẽ huấn luyện và một tập riêng, ở đây chúng tôi gọi là tập "phát triển huấn luyện" và nó sẽ không được dùng cho việc huấn luyện.

Bạn giờ đây có bốn tập con dữ liệu:

Tập huấn luyện: Đây là dữ liệu mà thuật toán sẽ học từ nó (ví dụ: ảnh Internet + ảnh Điện Thoại). Tập dữ liệu này không nhất thiết phải được trích xuất từ cùng phân phối như là đối với tập phát triển/kiểm tra.

Tập phát triển huấn luyện: Đây là dữ liệu trích xuất từ cùng phân phối với tập huấn luyện (ví dụ: ảnh Internet + ảnh Điện Thoại). Nó thông thường nhỏ hơn tập huấn luyện và chỉ cần đủ lớn để có thể đánh giá và theo dõi quá trình học của thuật toán.

- Tập phát triển: Đây là dữ liệu trích xuất từ cùng phân phối với tập kiểm tra, nó phản ánh phân phối dữ liệu mà chúng ta mong muốn thuật toán thực hiện tốt nhất. (Ví dụ: ảnh điện thoại)
- Tập kiểm tra: Đây là dữ liệu trích xuất từ cùng phân phối với tập phát triển. (Ví dụ: ảnh điện thoại)

Được trang bị với bốn tập dữ liệu riêng biệt, bạn giờ đây có thể đánh giá:

- Lỗi huấn luyện, bằng cách đánh giá tập huấn luyện.
- Khả năng tổng quát hóa của thuật toán đối với dữ liệu mới trích xuất từ cùng phân phối với tập huấn luyện, bằng cách đánh giá tập phát triển huấn luyện.
- Chất lượng của thuật toán trên tác vụ mà bạn quan tâm, bằng cách đánh giá tập phát triển và/hoặc tập kiểm tra.

Phần lớn những hướng dẫn ở Chương 5-7 về lựa chọn kích cỡ của tập phát triển có thể áp dụng được với tập phát triển huấn luyện.

41. Xác định những lỗi về độ chêch, phuơng sai, và dữ liệu không tương đồng

Giả sử con người đạt được chất lượng gần như hoàn hảo (lỗi ≈0%) trong việc phát hiện mèo, và do vậy tỷ lệ lỗi tối ưu là khoảng 0%. Giả sử bạn có:

- 1% lỗi trên tập huấn luyện.
- 5% lỗi trên tập phát triển huấn luyện.
- 5% lỗi trên tập phát triển.

Như vậy nói lên được điều gì? Ở đây, bạn biết rằng bạn đang có phuơng sai cao. Bạn có thể cải tiến thêm với các kỹ thuật giảm phuơng sai được mô tả trước đây.

Bây giờ, giả sử thuật toán của bạn đạt được:

- 10% lỗi trên tập huấn luyện.
- 11% lỗi trên tập phát triển huấn luyện.
- 12% lỗi trên tập phát triển.

Điều này cho bạn biết rằng bạn có độ chêch có thể tránh được cao trên tập huấn luyện. Tức là, thuật toán đang hoạt động kém trên tập huấn luyện. Kỹ thuật giảm độ chêch sẽ có ích trong trường hợp này.

Trong hai ví dụ trên, thuật toán chỉ có vấn đề về độ chêch có thể tránh được cao, hoặc về phuơng sai cao. Một thuật toán có thể mắc phải một hoặc nhiều vấn đề về độ chêch tránh được cao, phuơng sai cao hoặc dữ liệu không tương đồng. Ví dụ như:

- 10% lỗi trên tập huấn luyện.
- 11% lỗi trên tập phát triển huấn luyện.
- 20% lỗi trên tập phát triển.

Thuật toán này có độ chêch có thể tránh được cao và lỗi dữ liệu không tương đồng. Tuy nhiên, nó không có vấn đề phuơng sai cao trong phân phối tập huấn luyện.

Mối quan hệ giữa các loại lỗi có thể sẽ dễ hiểu hơn bằng cách vẽ chúng trong bảng:

		Phân phối A: Ảnh trên Internet + điện thoại	Phân phối B: Ảnh trên điện thoại
Mức con người	"Lỗi mức con người" (≈0%)		
Lỗi trên các mẫu mà thuật toán đã được học	"Lỗi huấn luyện" (10%)		
Lỗi trên các mẫu mà thuật toán chưa được học	"Lỗi huấn luyện - phát triển" (11%)	"Lỗi phát triển - kiểm tra" (20%)	
Dữ liệu không tương đồng		<p>Độ chêch tránh được</p> <p>Phuơng sai</p>	

Tiếp tục với ví dụ về bộ phát hiện hình ảnh mèo, bạn có thể thấy rằng có hai phân phối dữ liệu khác nhau trên trục x. Trên trục y, chúng ta có ba loại lỗi: lỗi ở mức con người, lỗi trên các mẫu mà thuật toán đã được học và lỗi trên các mẫu mà thuật toán chưa được học. Chúng ta có thể điền vào các ô với các loại lỗi khác nhau mà chúng ta đã xác định được trong chương trước.

Nếu muốn, bạn cũng có thể điền vào hai ô còn lại: Bạn có thể điền vào ô phía trên bên phải (Chất lượng mức con người trên Hình ảnh từ điện thoại) bằng cách yêu cầu một vài người dán nhãn dữ liệu ảnh mèo được tải lên từ điện thoại và đo lỗi của họ. Bạn có thể điền vào ô tiếp theo bằng cách lấy một phần nhỏ những tấm ảnh mèo chụp từ điện thoại (Phân phối B) vào tập huấn luyện để mạng nơ-ron cũng có thể học theo. Sau đó, bạn đo lỗi mô hình đã học trên tập dữ liệu con đó. Việc điền thêm vào hai mục này đôi khi có thể cung cấp thêm cái nhìn sâu sắc về những gì thuật toán đang thực hiện trên những phân phối khác nhau (Phân phối A và B) của dữ liệu.

Bằng cách hiểu được loại lỗi mà thuật toán đang gặp nhiều vấn đề nhất, bạn sẽ ở trong vị thế tốt hơn để quyết định xem nên tập

trung vào việc giảm độ chêch, giảm phuơng sai hay giảm độ không tương đồng của dữ liệu.

42. Xử lý dữ liệu không tương đồng

Giả sử bạn đã phát triển một hệ thống nhận dạng giọng nói hoạt động rất tốt trên tập huấn luyện và trên tập phát triển huấn luyện. Tuy nhiên, hệ thống đó lại hoạt động kém trên tập phát triển: Bạn có vấn đề về dữ liệu không tương đồng. Bạn có thể làm gì?

Tôi đề xuất bạn nên: (i) Cố gắng hiểu những thuộc tính nào của dữ liệu là khác nhau giữa phân phối của tập huấn luyện và tập phát triển. (ii) Cố gắng tìm thêm dữ liệu huấn luyện tương đồng hơn với những mẫu trong tập phát triển mà thuật toán của bạn đang gặp vấn đề.

Ví dụ, giả sử bạn thực hiện phân tích lỗi nhận dạng giọng nói trên tập phát triển: Bạn duyệt qua 100 mẫu một cách thủ công và cố gắng hiểu xem thuật toán đang mắc lỗi ở những đâu. Bạn phát hiện rằng hệ thống của bạn hoạt động kém vì hầu hết những đoạn âm thanh trong tập phát triển được thu trong xe hơi, trong khi hầu hết các ví dụ để huấn luyện được thu trong môi trường yên tĩnh. Tiếng ồn từ động cơ và đường phố làm ảnh hưởng nghiêm trọng đến chất lượng của hệ thống của bạn. Trong trường hợp này, bạn có thể cố gắng thu thập thêm dữ liệu huấn luyện bao gồm những đoạn âm thanh được thu trong xe hơi. Mục đích của việc phân tích lỗi là để hiểu những khác biệt đáng kể giữa tập huấn luyện và tập phát triển, vốn là nguyên nhân dẫn đến dữ liệu không tương đồng.

Nếu tập huấn luyện và tập phát triển huấn luyện của bạn chứa những đoạn âm thanh thu trong xe hơi, bạn nên kiểm tra kĩ lưỡng chất lượng của hệ thống của bạn trên tập con dữ liệu này. Nếu hệ thống hoạt động tốt với dữ liệu xe hơi trong tập huấn luyện nhưng không tốt với dữ liệu xe hơi trong tập phát triển huấn luyện, điều đó càng khẳng định giả thuyết rằng thu thập thêm dữ liệu xe hơi sẽ có ích. Đó là lí do tại sao chúng ta thảo luận trong chương trước về khả năng thêm một số dữ liệu lấy từ tập phát triển/tập kiểm tra với cùng phân phối vào trong dữ liệu huấn luyện của bạn. Làm như vậy cho phép bạn so sánh chất lượng [hệ thống] trên tập huấn luyện so với trên tập phát triển/kiểm tra.

Thật không may, không có một sự bảo đảm nào trong quá trình này. Ví dụ, nếu bạn không có cách nào để có thể có thêm dữ liệu huấn luyện tương đồng với dữ liệu trong tập phát triển, bạn có thể không có được một lộ trình rõ ràng nào hướng đến việc cải thiện chất lượng hệ thống.

GHI CHÚ:

[14] Cũng có một số nghiên cứu về "thích ứng miền" -- làm sao để huấn luyện một thuật toán trên một phân phối và để nó tổng quát hóa trên một phân phối khác. Những phương pháp này thường chỉ ứng dụng được với một số loại bài toán đặc biệt, và ít được sử dụng hơn nhiều so với những ý tưởng đã được trình bày trong chương này.

43. Tổng hợp dữ liệu nhân tạo

Hệ thống giọng nói của bạn cần thêm dữ liệu nghe giống như được lấy từ trong xe hơi. Thay vì thu thập nhiều dữ liệu trong khi lái xe, bạn có thể lấy chúng dễ dàng hơn bằng: tổng hợp nhân tạo.

Giả sử bạn có một số lượng lớn đoạn âm thanh tiếng ồn xe hơi/đường phố. Bạn có thể tải dữ liệu này từ một số trang web. Giả sử bạn cũng có một tập huấn luyện lớn của tiếng người đang nói trong một căn phòng yên tĩnh. Nếu bạn lấy đoạn âm thanh của một người đang nói và "thêm" vào một đoạn âm thanh tiếng ồn xe hơi/đường phố, bạn sẽ có được một đoạn âm thanh như thể người đó đang nói trong một chiếc xe ồn ào. Sử dụng quy trình này, bạn có thể "tổng hợp" lượng dữ liệu khổng lồ nghe như thể nó được thu thập bên trong một chiếc xe hơi.

Tổng quát hơn, có một số trường hợp mà tổng hợp dữ liệu nhân tạo cho phép bạn tạo một tập dữ liệu khổng lồ phù hợp với tập phát triển. Hãy sử dụng bộ nhận dạng ảnh mèo làm ví dụ thứ hai. Bạn nhận thấy rằng những ảnh của tập phát triển hay bị mờ chuyển động nhiều hơn bởi vì chúng có xu hướng đến từ người dùng điện thoại di động -- thường hơi di chuyển điện thoại khi chụp ảnh. Bạn có thể lấy những ảnh không bị mờ từ tập huấn luyện của ảnh Internet, và thêm vào mô phỏng của hiệu ứng mờ chuyển động cho giống với tập phát triển hơn.

Hãy nhớ rằng tổng hợp dữ liệu nhân tạo có những thách thức của nó: đôi khi dễ dàng tạo ra dữ liệu tổng hợp có vẻ giống thật với người hơn là tạo dữ liệu có vẻ giống thật với máy tính. Giả sử bạn có 1.000 giờ dữ liệu huấn luyện giọng nói, nhưng chỉ có một giờ tiếng ồn xe hơi. Nếu bạn liên tục sử dụng cùng một giờ tiếng ồn xe hơi với các phần khác nhau từ 1.000 giờ dữ liệu huấn luyện ban đầu, bạn sẽ nhận được với một tập dữ liệu tổng hợp trong đó tiếng ồn xe hơi giống nhau lặp đi lặp lại. Mặc dù một người nghe âm thanh này có thể sẽ không phân biệt được -- tất cả tiếng ồn của xe hơi đều giống nhau với hầu hết chúng ta -- nhưng có thể thuật toán học sẽ "overfit" một giờ tiếng ồn của xe hơi đó. Do đó, nó có thể khai quật kém khi gặp một đoạn âm thanh mới với tiếng ồn xe hơi khác.

Ngoài ra, giả sử bạn có 1.000 giờ tiếng ồn xe hơi khác biệt, nhưng tất cả đều được lấy từ 10 chiếc xe khác nhau. Trong trường hợp này, thuật toán có thể "overfit" 10 chiếc xe này và đạt chất lượng kém nếu được thử nghiệm trên âm thanh từ một chiếc xe khác. Thật không may, những vấn đề này có thể khó phát hiện.



Lấy một ví dụ nữa, giả sử bạn đang xây dựng một hệ thống thị giác máy để nhận diện xe hơi. Giả sử bạn hợp tác với một công ty

sản xuất game có mô hình đồ họa máy tính của một số xe hơi. Để huấn luyện thuật toán của bạn, bạn sử dụng các mô hình để tạo ra hình ảnh xe hơi. Ngay cả khi các hình ảnh tổng hợp trông rất giống thật, phương pháp này (đã được nhiều người độc lập đề xuất) có thể sẽ không hoạt động tốt. Trong toàn bộ game có thể có ~20 thiết kế xe hơi. Nó rất tốn kém để xây dựng một mô hình 3D của một chiếc xe hơi; Nếu bạn đang chơi game, có lẽ bạn sẽ nhận thấy rằng bạn đang nhìn thấy những chiếc xe giống nhau lặp đi lặp lại, có lẽ chỉ được sơn khác nhau. Tức là, dữ liệu này trông rất giống thật với bạn. Nhưng so với tập hợp tất cả các xe hơi trên đường phố (hay những gì bạn có thể thấy trong tập phát triển/kiểm tra), bộ 20 chiếc xe được tổng hợp này chỉ chiếm một phần rất nhỏ trong phân phối xe hơi thế giới. Do đó, nếu 100.000 mẫu huấn luyện của bạn đều đến từ 20 chiếc xe này, hệ thống của bạn sẽ "overfit" với 20 thiết kế xe cụ thể này và sẽ không thể khái quát tốt cho các tập phát triển/kiểm tra bao gồm các thiết kế xe khác.

Khi tổng hợp dữ liệu, hãy suy nghĩ xem bạn có thực sự tổng hợp một tập hợp các mẫu đại diện hay không. Cố gắng tránh đưa ra các thuộc tính dữ liệu tổng hợp mà thuật toán học có thể phân biệt được các mẫu được tổng hợp với các mẫu không được tổng hợp, ví dụ như nếu tất cả dữ liệu được tổng hợp đến từ một trong 20 thiết kế xe hơi hoặc tất cả âm thanh được tổng hợp chỉ từ một giờ tiếng ôn xe hơi. Có thể rất khó để thực hiện lời khuyên này.

Khi làm việc về tổng hợp dữ liệu, các nhóm của tôi đôi khi phải mất vài tuần để tạo ra dữ liệu với các chi tiết đủ gần với phân phối thực tế để dữ liệu được tổng hợp có hiệu quả rõ rệt. Nhưng nếu bạn có thể có được các chi tiết đúng, bạn có thể đột nhiên có được một tập huấn luyện lớn hơn nhiều so với trước đây.

Phần 7: Gõ lỗi các Thuật toán suy luận

44. Bài kiểm tra xác minh tối ưu

Giả sử bạn đang xây dựng một hệ thống nhận dạng giọng nói. Hệ thống của bạn hoạt động bằng cách nhập một đoạn âm thanh A , và tính toán một giá trị $\text{Điểm}_A(S)$ cho mỗi câu đầu ra khả dĩ S . Ví dụ: bạn có thể thử ước tính $\text{Điểm}_A(S) = P(S|A)$, tức xác suất bản ghi thoại đầu ra chính xác là câu S với điều kiện âm thanh đầu vào là A .

Khi có cách tính $\text{Điểm}_A(S)$, bạn vẫn phải tìm câu tiếng Anh S để tối đa hóa nó:

$$\text{Output} = \arg \max_S \text{Score}_A(S)$$

Làm thế nào để bạn tính toán được "arg max" ở trên? Nếu tiếng Anh có 50.000 từ thì sẽ có (50.000)⁵⁰ câu khả dĩ có độ dài N -- quá nhiều để liệt kê ra một cách triệt để.

Vì vậy, bạn cần áp dụng thuật toán tìm kiếm gần đúng, để cố gắng tìm giá trị của S để tối ưu hóa (tối đa hóa) $\text{Điểm}_A(S)$. Ví dụ với thuật toán tìm kiếm chùm tia (beam search), thuật toán này chỉ giữ K ứng viên hàng đầu trong quá trình tìm kiếm. (Đối với mục đích của chương này, bạn không cần phải hiểu chi tiết về tìm kiếm chùm tia.) Các thuật toán như thế này không đảm bảo được việc tìm giá trị của S mà tối đa hóa $\text{Điểm}_A(S)$.

Giả sử rằng một đoạn âm thanh A ghi lại một người nào đó nói rằng "Tôi yêu thích học máy". Tuy nhiên, thay vì xuất ra bản ghi thoại chính xác, hệ thống của bạn lại đưa ra một phiên bản không chính xác "Tôi yêu thích người máy". Có hai khả năng giải thích cho việc thiếu chính xác này:

- Vấn đề về thuật toán tìm kiếm** Thuật toán tìm kiếm gần đúng (tim kiem chum tia) không thể tìm thấy giá trị của S thỏa mãn việc tối đa hóa $\text{Điểm}_A(S)$.
- Vấn đề về hàm mục tiêu (hàm tính điểm)** Ước lượng của chúng ta về $\text{Điểm}_A(S) = P(S|A)$ không chính xác. Cụ thể, việc chọn cách tính $\text{Điểm}_A(S)$ thất bại trong việc xác định "Tôi yêu thích học máy" là bản ghi thoại chính xác.

Tùy thuộc vào nguyên nhân nào dẫn đến thất bại mà bạn ưu tiên tập trung vào các hướng giải quyết khác nhau. Nếu #1 là vấn đề, bạn nên cải thiện thuật toán tìm kiếm. Nếu #2 là vấn đề, bạn nên làm việc với thuật toán học ước lượng $\text{Điểm}_A(S)$.

Đối mặt với tình huống này, một số nhà nghiên cứu sẽ ngẫu nhiên quyết định làm việc trên thuật toán tìm kiếm; những người khác sẽ ngẫu nhiên làm việc theo cách tốt hơn để thuật toán học các giá trị cho $\text{Điểm}_A(S)$. Nhưng nếu bạn không biết nguyên nhân nào trong số này là nguyên nhân cơ bản tạo nên lỗi, nỗ lực của bạn có thể trở nên lãng phí. Làm thế nào bạn có thể quyết định nên làm gì một cách có hệ thống hơn?

Đặt S_{out} là bản ghi thoại đầu ra ("Tôi yêu thích người máy"). Đặt S^* là bản ghi thoại chính xác ("Tôi yêu thích học máy"). Để hiểu vấn đề #1 hay #2 ở trên là nguyên nhân, bạn có thể thực hiện **Bài kiểm tra xác minh tối ưu** Đầu tiên, tính $\text{Điểm}_A(S^*)$ và $\text{Điểm}_A(S_{\text{out}})$. Sau đó kiểm tra xem $\text{Điểm}_A(S^*) > \text{Điểm}_A(S_{\text{out}})$ có đúng hay không. Có hai khả năng:

Trường hợp 1: $\text{Điểm}_A(S^*) > \text{Điểm}_A(S_{\text{out}})$

Trong trường hợp này, thuật toán học của bạn đã tính S^* cao hơn S_{out} . Tuy nhiên, thuật toán tìm kiếm gần đúng đã chọn S_{out} thay vì S^* . Điều này cho bạn biết rằng thuật toán tìm kiếm gần đúng của bạn có lỗi trong việc chọn giá trị S tối đa hóa $\text{Điểm}_A(S)$. Trong trường hợp này, Bài kiểm tra xác minh tối ưu cho bạn biết rằng bạn có vấn đề về thuật toán tìm kiếm và nên tập trung vào đó. Ví dụ, bạn có thể thử tăng độ rộng chùm tia của tìm kiếm chùm tia.

Trường hợp 2: $\text{Điểm}_A(S^*) \leq \text{Điểm}_A(S_{\text{out}})$

Trong trường hợp này, bạn biết việc bạn tính toán $\text{Điểm}_A(.)$ có lỗi: Không thể cho điểm cao hơn hẳn cho đầu ra chính xác S^* so với S_{out} không chính xác. Bài kiểm tra xác minh tối ưu cho bạn biết rằng bạn có vấn đề về hàm mục tiêu (tính điểm). Vì vậy, bạn nên tập trung vào việc cải thiện cách bạn học hoặc tính gần đúng $\text{Điểm}_A(S)$ cho các câu khác nhau S . Vì vậy, bạn nên tập trung vào việc cải thiện thuật toán học hoặc cách xấp xỉ $\text{Điểm}_A(S)$ cho các câu khác nhau S .

Thảo luận của chúng ta đã tập trung vào một ví dụ duy nhất. Để áp dụng Bài kiểm tra xác minh tối ưu trong thực tế, bạn nên kiểm tra các lỗi trong tập phát triển của mình. Đối với mỗi lỗi, bạn sẽ kiểm tra xem $\text{Điểm}_A(S^*) > \text{Điểm}_A(S_{\text{out}})$ không. Mỗi ví dụ trong tập phát triển mà bất đồng thức này thoả mãn được đánh dấu là lỗi gây ra bởi thuật toán tối ưu. Mỗi ví dụ không thoả mãn ($\text{Điểm}_A(S^*) \leq \text{Điểm}_A(S_{\text{out}})$) được tính là một lỗi do cách bạn tính toán $\text{Điểm}_A(.)$.

Ví dụ, giả sử bạn tìm ra rằng 95% các lỗi là do hàm tính điểm Điểm(.) , và chỉ 5% có nguyên do từ phía thuật toán tối ưu. Giờ bạn biết rằng dù có cải thiện quá trình tối ưu thế nào thì bạn cũng chỉ có thể loại bỏ được khoảng 5% lỗi. Thay vào đó, bạn nên tập trung cải thiện cách ước lượng Điểm_A(.).

45. Dạng tổng quát của bài kiểm tra xác minh tối ưu

Bạn có thể áp dụng bài kiểm tra xác minh tối ưu khi, với một số đầu vào x , bạn biết cách tính $\text{Điểm}_*(y)$ dùng để thể hiện mức độ phản hồi y tốt như thế nào với x . Hơn nữa, bạn đang sử dụng thuật toán gần đúng để cố gắng tìm $\arg \max_y \text{Điểm}_*(y)$, nhưng nghĩ ngợi rằng thuật toán tìm kiếm đôi khi không tìm thấy giá trị lớn nhất. Trong ví dụ nhận dạng giọng nói trước đây của chúng tôi, $x=A$ là một đoạn âm thanh và $y=S$ là bản ghi thoại đầu ra.

Giả sử y^* là đầu ra "chính xác" nhưng thuật toán thay vào đó tìm ra y_{out} . Thì bài kiểm tra chính là do xem liệu $\text{Điểm}(y^*) > \text{Điểm}_*(y_{\text{out}})$ có đúng hay không. Nếu bất đẳng thức này đúng, thì chúng ta coi lỗi là do thuật toán tối ưu. Tham khảo chương trước để đảm bảo bạn hiểu logic dồn sau điều này. Ngược lại, chúng ta coi lỗi thuộc về cách tính $\text{Điểm}_*(y)$.

Xem xét một ví dụ nữa. Giả sử bạn đang xây dựng một hệ thống dịch máy từ tiếng Trung sang tiếng Anh. Hệ thống của bạn nhận một câu tiếng Trung C và tính giá trị $\text{Điểm}_C(E)$ cho mỗi bản dịch khả dĩ E . Ví dụ, bạn có thể sử dụng $\text{Điểm}_C(E) = P(E|C)$, xác suất dịch ra E với câu đầu vào C .

Thuật toán của bạn dịch các câu bằng cách cố gắng tính:

$$\text{Output} = \arg \max_E \text{Score}_C(E)$$

Tuy nhiên, tập hợp các câu tiếng Anh có thể E quá lớn nên bạn dựa vào thuật toán tìm kiếm thực nghiệm.

Giả sử thuật toán của bạn dịch ra một bản không chính xác E_{out} thay vì một bản dịch chính xác E^* nào đó. Thì bài kiểm tra xác minh tối ưu sẽ yêu cầu bạn tính toán xem liệu $\text{Điểm}_C(E^*) > \text{Điểm}_C(E_{\text{out}})$ có đúng hay không. Nếu bất đẳng thức này đúng thì cách tính $\text{Điểm}_C(\cdot)$ đã nhận dạng chính xác E^* tốt hơn so với E_{out} ; do đó, bạn sẽ coi lỗi này là do thuật toán tìm kiếm gần đúng. Ngược lại, bạn coi lỗi này thuộc về cách tính $\text{Điểm}_C(\cdot)$.

Đây là một "mẫu thiết kế" rất phổ biến trong AI khi lần đầu học một hàm tính điểm gần đúng $\text{Điểm}(\cdot)$, sau đó sử dụng một thuật toán tối ưu đa xấp xỉ. Nếu bạn có thể phát hiện ra kiểu mẫu này, bạn sẽ có thể sử dụng bài kiểm tra xác minh tối ưu để hiểu nguồn gốc lỗi của mình.

46. Ví dụ về Học tăng cường



Giả sử như bạn đang sử dụng học máy để dạy trực thăng bay theo những chuyển động phức tạp. Đây là một tấm ảnh time-lapse của một chiếc trực thăng được máy tính điều khiển thực hiện việc hạ cánh khi động cơ đã tắt.

Đây được gọi là kĩ thuật "quay tự động". Nó cho phép trực thăng hạ cánh ngay cả khi động cơ bị hỏng ngoài dự kiến. Phi công thực hành kĩ thuật bay này như một phần trong công tác huấn luyện bay. Nhiệm vụ của bạn là sử dụng một thuật toán học tập để lái chiếc trực thăng qua một quỹ đạo T và kết thúc với một pha hạ cánh an toàn.

Để áp dụng học tăng cường, bạn phải phát triển một "hàm điểm thưởng" $R(\cdot)$ trả về một chỉ số đánh giá mức độ tốt của mỗi quỹ đạo T . Lấy ví dụ, nếu T kết thúc bằng việc trực thăng bị rơi, thì có thể nhận điểm thưởng $R(T) = -1.000$ -- một điểm thưởng âm rất lớn. Một quỹ đạo T kết thúc bằng việc trực thăng hạ cánh an toàn có thể sẽ cho $R(T)$ dương với giá trị chính xác phụ thuộc vào việc hạ cánh êm ái như thế nào. Hàm điểm thưởng $R(\cdot)$ thường được chọn thủ công để định lượng mức độ mong muốn của những quỹ đạo T khác nhau. Nó phải đánh đổi giữa những đặc tính như mức độ xóc khi hạ cánh, trực thăng có hạ cánh đúng vị trí mong muốn không, quá trình hạ độ cao có nhiều biến động đối với hành khách không, và vân vân. Thiết kế để có được những hàm điểm thưởng tốt không hề dễ dàng.

Với một hàm điểm thưởng $R(T)$ cho trước, công việc của thuật toán học tăng cường là điều khiển trực thăng sao cho nó đạt được điểm thưởng cao nhất $\max_T R(T)$. Tuy nhiên, thuật toán học tăng cường có nhiều phép xấp xỉ và có thể sẽ không thành công trong việc tối ưu này.

Giả sử bạn đã có một hàm điểm thưởng $R(\cdot)$ nào đó và đã chạy thuật toán học của bạn. Tuy nhiên chất lượng của nó còn thua xa chất lượng của người lái (ví dụ: hạ cánh xóc hơn và có vẻ kém an toàn hơn so với chất lượng của phi công). Làm sao để bạn biết được liệu đó có phải là lỗi của thuật toán học tăng cường -- được dùng để tính toán một quỹ đạo để tối đa $\max_T R(T)$, hay là lỗi của hàm điểm thưởng -- được dùng để đo cũng như xác định mức đánh đổi lý tưởng giữa độ xóc của chuyến bay và độ chính xác của vị trí hạ cánh?

Để áp dụng bài kiểm tra xác minh tố ưu, cho $T_{người}$ là quỹ đạo bay của phi công, và cho T_{ra} là quỹ đạo đạt được của thuật toán. Dựa theo mô tả ở phía trên của chúng ta, $T_{người}$ là quỹ đạo tốt hơn so với T_{ra} . Do vậy, bài kiểm tra chính là: Liệu có đúng không khi $R(T_{người}) > R(T_{ra})$?

Trường hợp 1: Nếu bất đẳng thức này đúng, thì hàm điểm thưởng $R(\cdot)$ đang đánh giá đúng rằng $T_{người}$ vượt trội hơn so với T_{ra} . Nhưng vậy thì thuật toán học tăng cường của chúng ta đang tìm T_{ra} kém hơn. Điều này gợi ý rằng bỗng nhiên cải thiện thuật toán học tăng cường của chúng ta là xứng đáng.

Trường hợp 2: Bất đẳng thức trên không đúng: $R(T_{người}) \leq R(T_{ra})$. Tức là $R(\cdot)$ đang gán cho $T_{người}$ một điểm số tệ hơn dù cho nó

là quỹ đạo tốt hơn. Bạn nên cải thiện $R(\cdot)$ để có thể nắm bắt việc đánh đổi giữa các tiêu chí tương đương với một cú hạ cánh tốt.

Nhiều ứng dụng machine learning có chung "khuôn mẫu" là tối ưu xấp xỉ một hàm tính điểm $\text{Điểm}(\cdot)$ sử dụng một thuật toán tìm kiếm xấp xỉ. Đôi khi cũng không tồn tại một đầu vào x được chỉ định trước, vậy nên nó suy giảm thành $\text{Điểm}(\cdot)$. Trong ví dụ trên của chúng ta, hàm tính điểm chính là hàm điểm thường $\text{Điểm}(T) = R(T)$, và thuật toán tối ưu là thuật toán học tăng cường đang cố thực thi một quỹ đạo bay T tốt.

Một điểm khác biệt so với những ví dụ trước là, thay vì so sánh với một kết quả "tối ưu", bạn so sánh với chất lượng mức con người $T_{người}$. Chúng ta giả sử $T_{người}$ khá là tốt, hoặc thậm chí là tối ưu. Nhìn chung, miễn là bạn có kết quả y^* (trong ví dụ này, $T_{người}$) tốt hơn so với thuật toán học của bạn hiện thời -- mặc dù có thể nó không phải là kết quả "tối ưu" -- thì Bài kiểm tra xác minh tối ưu có thể chỉ ra xem liệu cải thiện thuật toán tối ưu hay cải thiện hàm tính điểm sẽ hứa hẹn hơn.

Phần 8: Học sâu đầu-cuối

47. Sự trỗi dậy của học đầu-cuối

Giả sử bạn muốn xây dựng một hệ thống kiểm tra đánh giá các phản hồi sản phẩm trực tuyến và tự động cho biết liệu người viết có thích sản phẩm đó hay không. Ví dụ, bạn hi vọng có thể nhận ra phản hồi dưới đây là tích cực:

Cây lau nhà này thật tuyệt!

và đoạn dưới đây với kết quả là tiêu cực:

Cây lau nhà này thật kém chất lượng -- Tôi hối hận vì đã mua nó.

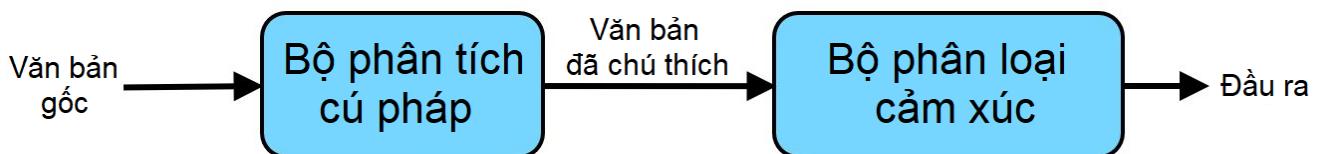
Bài toán về nhận dạng các quan điểm tích cực và tiêu cực được gọi là "phân loại cảm xúc". Để xây dựng hệ thống này, bạn có thể tạo một "pipeline" bao gồm hai thành phần:

- a. Bộ phân tích cú pháp: Một hệ thống tạo chú thích văn bản trích xuất thông tin từ những từ quan trọng nhất. [15] Ví dụ, bạn có thể sử dụng bộ phân tích cú pháp để tạo nhãn tất cả tính từ và danh từ. Từ đó có được đoạn chú thích như sau:

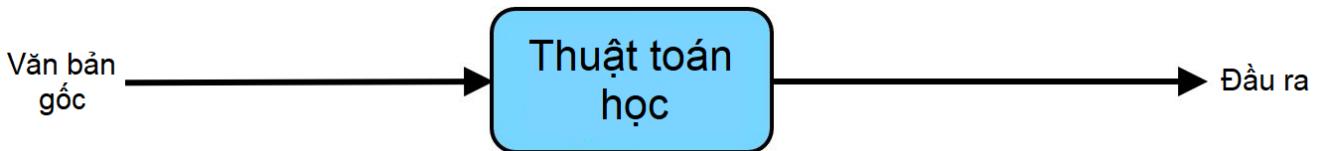
Cây lau nhà_{Danh Từ} này thật tuyệt_{Tính từ}!

- b. Bộ phân loại cảm xúc: Một thuật toán học sử dụng đầu vào là văn bản đã chú thích để dự đoán cảm xúc tổng thể. Khả năng chú thích của bộ phân tích cú pháp có thể giúp ích rất nhiều thuật toán học: Bằng việc tập trung hơn vào các tính từ, thuật toán của bạn có thể nhanh chóng xác định các từ quan trọng như "tuyệt", và lờ đi những từ ít quan trọng hơn như "này".

Chúng ta có thể hình dung "pipeline" của hai thành phần này như sau:



Xu hướng gần đây là thay đổi hệ thống pipeline với một thuật toán duy nhất. Một **thuật toán đầu-cuối** cho tác vụ này sẽ đơn giản là thu thập văn bản gốc "Cây lau nhà này thật tuyệt!", và cố gắng trực tiếp nhận ra cảm xúc từ nó:



Mạng neural được sử dụng phổ biến trong các hệ thống đầu-cuối. Thuật ngữ "đầu-cuối" phản ánh thực tế là chúng ta yêu cầu thuật toán chạy trực tiếp từ đầu vào cho đến đầu ra mong muốn. Tức là, thuật toán học kết nối trực tiếp "đầu vào" cho đến "đầu ra" của hệ thống.

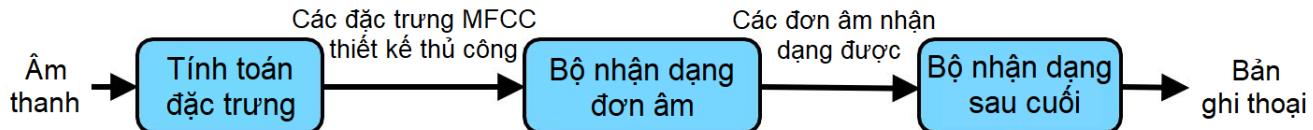
Đối với các vấn đề mà dữ liệu rất phong phú, hệ thống đầu-cuối hoạt động khá hiệu quả. Tuy nhiên không phải lúc nào nó cũng là một lựa chọn tốt. Các chương tiếp theo sẽ cung cấp thêm một số ví dụ về hệ thống đầu-cuối cũng như lời khuyên để bạn biết thời điểm nào nên hoặc không nên sử dụng chúng.

GHI CHÚ

[15] Bộ phân tích cú pháp có thể cung cấp nhiều hơn các chú thích từ văn bản, tuy nhiên định nghĩa tối giản này là đủ để giải thích cho hệ thống học sâu đầu-cuối.

48. Thêm những ví dụ về học đầu-cuối.

Giả sử bạn muốn xây dựng một hệ thống nhận dạng giọng nói. Bạn có thể xây dựng hệ thống với ba thành phần sau:



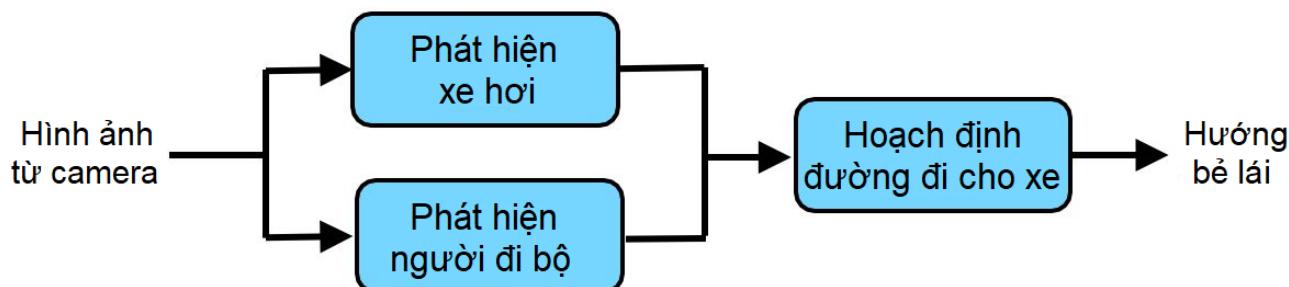
Các thành phần sẽ hoạt động như sau:

- Tính toán các đặc trưng: Trích xuất các đặc trưng được thiết kế thủ công, ví dụ như đặc trưng MFCC (Hệ số Mel-frequency cepstrum), được sử dụng để nắm bắt nội dung của đoạn phát biểu trong khi bỏ qua những thuộc tính ít liên quan hơn như âm sắc của người nói.
- Nhận diện các âm vị: Các nhà ngôn ngữ học tin rằng trong ngôn ngữ có các đơn vị cơ bản gọi là "âm vị." Ví dụ, âm bắt đầu "k" trong từ "keep" thì phát âm giống âm "c" trong từ "cake". Hệ thống này sẽ cố gắng để nhận diện các âm vị trong các đoạn âm thanh.
- Bộ nhận dạng cuối cùng: Dùng các chuỗi âm vị đã được nhận dạng, và cố gắng xâu chuỗi chúng lại với nhau thành một bản ghi thoại ở đầu ra.

Mặt khác, một hệ thống đầu-cuối có thể nhận đầu vào là một đoạn âm thanh, và sẽ cố gắng cho ra trực tiếp một bản ghi thoại:



Từ trước tới giờ, chúng ta chỉ mới mô tả các "pipeline" tuyến tính của học máy: đầu ra được truyền tuần tự từ giai đoạn này sang giai đoạn khác. Những pipeline có thể phức tạp hơn như trong ví dụ về một hệ thống xe tự lái đơn giản sau:



Hệ thống bao gồm ba thành phần: Một thành phần giúp phát hiện những xe khác bằng việc sử dụng hình ảnh từ máy quay; Một thành phần khác phát hiện người đi bộ; thành phần cuối cùng hoạch định một lộ trình giúp chiếc xe tránh những chiếc xe khác và người đi bộ.

Không phải mọi thành phần trong pipeline phải được huấn luyện. Ví dụ, các nghiên cứu về "hoạch định chuyển động của robot" đã có rất nhiều thuật toán cho bước hoạch định lộ trình của chiếc xe. Trong đó có nhiều thuật toán không yêu cầu việc huấn luyện.

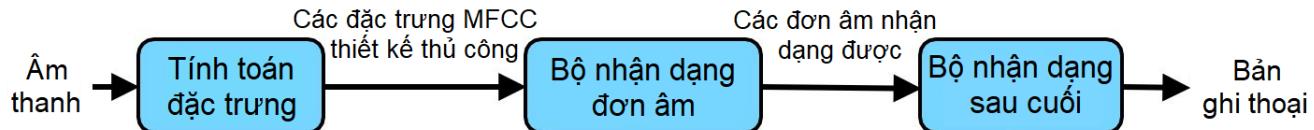
Ngược lại, hướng tiếp cận đầu-cuối có thể cố gắng lấy đầu vào là những tín hiệu cảm biến và cho ra trực tiếp kết quả hướng lái:



Mặc dù phương pháp học đầu-cuối đã đạt được nhiều kết quả tốt, nó không phải luôn luôn là hướng đi tốt nhất. Ví dụ, phương pháp nhận dạng giọng nói đầu-cuối đạt kết quả tốt. Nhưng tôi cảm thấy hoài nghi về việc sử dụng học đầu-cuối cho xe tự lái. Những chương kế tiếp sẽ giải thích về vấn đề này.

49. Ưu nhược điểm của học đầu-cuối

Xét một ví dụ về pipeline nhận dạng tiếng nói như trong các chương trước:



Rất nhiều thành phần của pipeline này được "thiết kế thủ công".

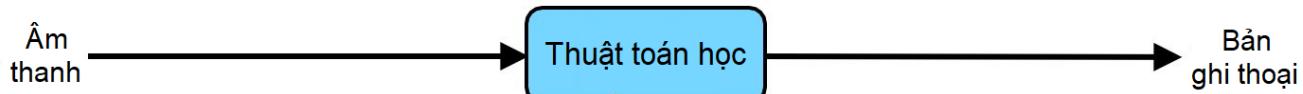
- MFCCs là một tập hợp của các đặc trưng âm thanh được thiết kế thủ công. Mặc dù chúng cung cấp một tóm tắt khá hợp lý cho dữ liệu âm thanh đầu vào, chúng cũng đã giản lược tín hiệu đầu vào bằng cách bỏ đi một vài thông tin.
- Hệ âm vị là một phát kiến của ngành ngôn ngữ học. Chúng là một biểu diễn không hoàn hảo của âm thanh thoại. Theo hướng hệ âm vị là một xấp xỉ khá tệ của thực tế, áp đặt một thuật toán sử dụng một biểu diễn âm vị sẽ giới hạn chất lượng của hệ thống tiếng nói.

Các thành phần được thiết kế thủ công này giới hạn chất lượng tiềm năng của hệ thống tiếng nói. Tuy nhiên, sử dụng các thành phần được thiết kế thủ công cũng có một vài ưu điểm:

- Đặc trưng MFCC có tính kháng đối với một vài tính chất không ảnh hưởng tới nội dung của tiếng nói, chẳng hạn như cao độ của giọng nói. Bởi vậy, chúng giúp giảm lược vấn đề của thuật toán học.
- Theo hướng hệ âm vị là một biểu diễn khá hợp lý của tiếng nói, chúng cũng có thể giúp thuật toán học hiểu được các thành phần cơ bản của âm thanh và bởi vậy cải thiện chất lượng của hệ thống.

Có nhiều thành phần được thiết kế thủ công hơn nhìn chung cho phép một hệ thống tiếng nói có thể học với ít dữ liệu hơn. Đặc trưng được thiết kế thủ công bởi MFCC và âm vị "bù đắp" đặc trưng thuật toán lấy được từ dữ liệu. Khi chúng ta không có nhiều dữ liệu, các đặc trưng này là hữu ích.

Bây giờ, xét hệ thống đầu-cuối:



Hệ thống này thiếu đặc trưng được thiết kế thủ công. Bởi vậy, khi tập huấn luyện nhỏ, nó có thể có chất lượng tệ hơn pipeline với đặc trưng được thiết kế thủ công.

Tuy nhiên, khi tập huấn luyện lớn, nó không bị cản trở bởi giới hạn của một biểu diễn MFCC hay hệ dựa trên hệ âm vị. Nếu thuật toán học là một mạng nơ-ron đủ lớn và được huấn luyện trên dữ liệu huấn luyện đủ lớn, nó có tiềm năng hoạt động tốt, và có thể đạt được tỉ lệ lỗi tối ưu.

Hệ thống học đầu-cuối có xu hướng làm việc tốt khi có rất nhiều dữ liệu được gán nhãn cho "cả hai đầu" -- đầu vào và đầu ra. Trong ví dụ này, chúng ta yêu cầu một tập dữ liệu lớn các cặp (âm thanh, bản ghi). Khi dữ liệu kiểu nay không tồn tại, hãy đặc biệt lưu ý khi sử dụng học đầu-cuối.

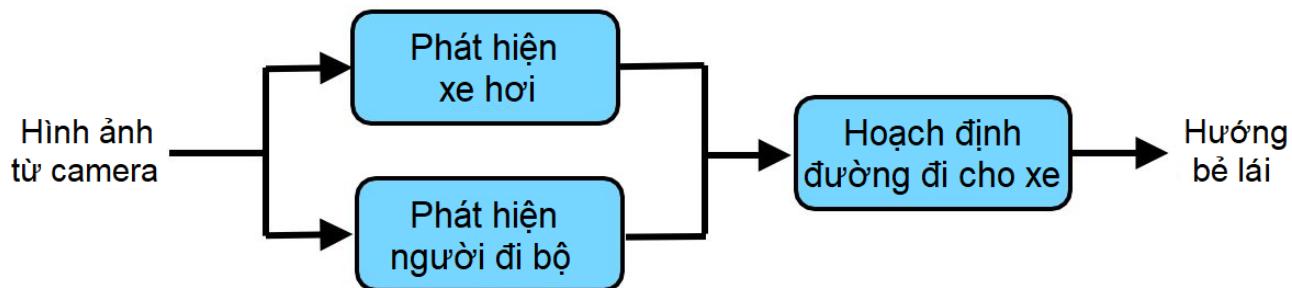
Nếu bạn đang làm việc với một bài toán học máy mà tập huấn luyện rất nhỏ, hầu hết các đặc trưng cho thuật toán phải đến từ hiểu biết của con người. Chẳng hạn, từ các thành phần được "thiết kế thủ công".

Nếu bạn không chọn sử dụng một hệ thống đầu-cuối, bạn sẽ phải lựa chọn từng bước trong pipeline của bạn và cách chúng được kết nối với nhau. Trong một vài chương tiếp theo, chúng tôi sẽ cung cấp một vài gợi ý khi thiết kế các pipeline dạng này.

50. Lựa chọn các thành phần cho pipeline: Tính sẵn có của dữ liệu

Khi xây dựng một hệ thống pipeline không phải đầu-cuối, các thành phần nào là những ứng viên tốt cho pipeline? Cách bạn thiết kế pipeline sẽ có tác động lớn tới toàn bộ chất lượng của hệ thống. Một nhân tố quan trọng là liệu rằng bạn có thể dễ dàng thu thập dữ liệu để huấn luyện mỗi thành phần.

Ví dụ, xét kiến trúc lái tự động dưới đây:



Bạn có thể sử dụng học máy để phát hiện xe hơi và người đi bộ. Hơn nữa, không khó để thu thập những dữ liệu này: Có vô vàn tập dữ liệu thị giác máy tính với lượng lớn xe hơi và người đi bộ được gán nhãn. Bạn cũng có thể dùng các dịch vụ cộng đồng (Amazon Mechanical Turk chẳng hạn) để có được những tập dữ liệu thậm chí lớn hơn. Bởi vậy khá dễ để thu thập dữ liệu huấn luyện để xây dựng một bộ phát hiện xe hơi và phát hiện người đi bộ.

Ngược lại, xét một hướng tiếp cận thuận đầu-cuối:



Để huấn luyện hệ thống này, chúng ta sẽ cần một tập dữ liệu lớn các cặp (Ảnh, Hướng Bẻ Lái). Sẽ rất mất thời gian và tiền của để có người lái xe loanh quanh và ghi lại hướng bẻ lái để thu thập dữ liệu này. Bạn cần những chiếc xe được gắn những thiết bị đặc biệt và một lượng lớn dữ liệu thu thập được khi lái xe để đảm bảo bao quát đầy đủ các tình huống khả dĩ. Việc này khiến hệ thống đầu-cuối rất khó để huấn luyện. Sẽ dễ hơn nhiều để đạt được một tập dữ liệu lớn với ảnh xe hơi và người đi bộ.

Tổng quát hơn, nếu có rất nhiều dữ liệu sẵn có để huấn luyện "các mô-đun trung gian" của một pipeline (chẳng hạn như một bộ phát hiện xe hơi và một bộ phát hiện người đi bộ), thì bạn có thể xem xét sử dụng một pipeline với nhiều bước. Kiến trúc này có thể ưu việt bởi vì bạn có thể sử dụng tất cả dữ liệu sẵn có để huấn luyện các mô-đun trung gian.

Cho tới khi nhiều dữ liệu đầu-cuối trở nên sẵn có, tôi tin rằng hướng tiếp cận phi đầu-cuối tiềm năng hơn một cách đáng kể cho xe tự lái: Kiến trúc của nó phù hợp hơn với tính sẵn có của dữ liệu.

51. Lựa chọn các thành phần cho pipeline: tính đơn giản của tác vụ

Ngoài sự sẵn có của dữ liệu, bạn cũng nên xem xét một nhân tố thứ hai khi lựa chọn các thành phần của một pipeline: Việc giải quyết từng tác vụ bằng các thành phần riêng lẻ đơn giản đến mức nào? Bạn nên cố gắng chọn những thành phần pipeline có thể dễ dàng xây dựng hay học riêng lẻ. Nhưng các thành phần "dễ" học nghĩa là gì?



Xét những tác vụ học máy này, được liệt kê theo thứ tự độ khó tăng dần:

- Phân loại bức ảnh có bị phơi sáng quá mức hay không (như trong ví dụ trên)
- Phân loại bức ảnh được chụp trong nhà hay ngoài trời
- Phân loại bức ảnh có chứa một con mèo hay không
- Phân loại bức ảnh có chứa một con mèo khoang đen trắng hay không
- Phân loại bức ảnh có chứa một con mèo Xiêm (tên một loại mèo) hay không

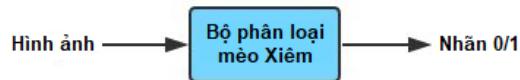
Mỗi tác vụ trên là một bài toán phân loại ảnh nhị phân: từ một bức ảnh đầu vào, mô hình phải cho ra giá trị 0 hoặc 1. Nhưng những tác vụ đầu tiên trong danh sách này có vẻ quá "dễ" đối với một mạng nơ-ron. Bạn sẽ có thể huấn luyện những tác vụ dễ hơn với ít mẫu huấn luyện hơn.

Học máy chưa có một định nghĩa chính thức nào về một tác vụ là dễ hay khó[16]. Với sự phát triển của học sâu và mạng nơ-ron đa tầng, chúng ta nói một tác vụ là "dễ" nếu có có thể được thực hiện với ít bước tính toán hơn (ứng với mạng nơ-ron nông), và "khó" nếu nó đòi hỏi nhiều bước tính toán hơn (đòi hỏi một mạng nơ-ron sâu hơn). Nhưng đây đều là các định nghĩa không chính thức.

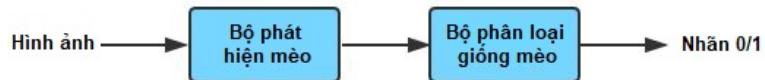
Nếu bạn có thể lấy một tác vụ phức tạp, và chia nhỏ nó thành những tác vụ con đơn giản hơn. Sau đó bằng cách viết mã nguồn cụ thể cho từng tác vụ con đó, bạn đang cung cấp cho thuật toán một tri thức tiền đề giúp nó học một tác vụ hiệu quả hơn.



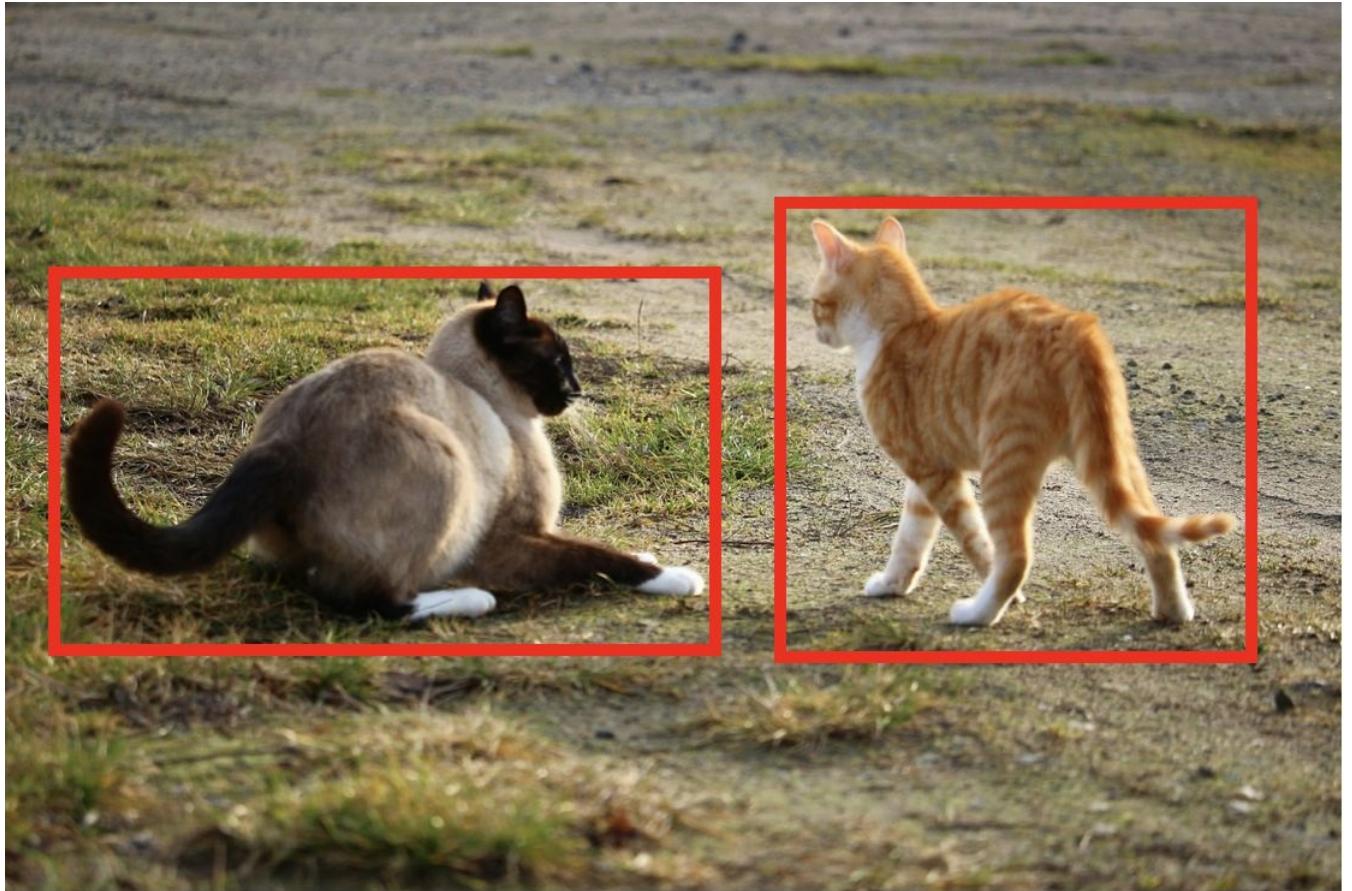
Giả sử bạn đang xây dựng bộ phát hiện một con mèo Xiêm. Dưới đây là một kiến trúc thuận đầu-cuối:



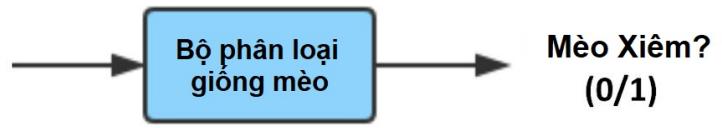
Ngược lại, bạn cũng có thể sử dụng một pipeline với hai bước:



Bước đầu tiên (bộ phát hiện mèo) tìm tất cả con mèo trong bức ảnh.

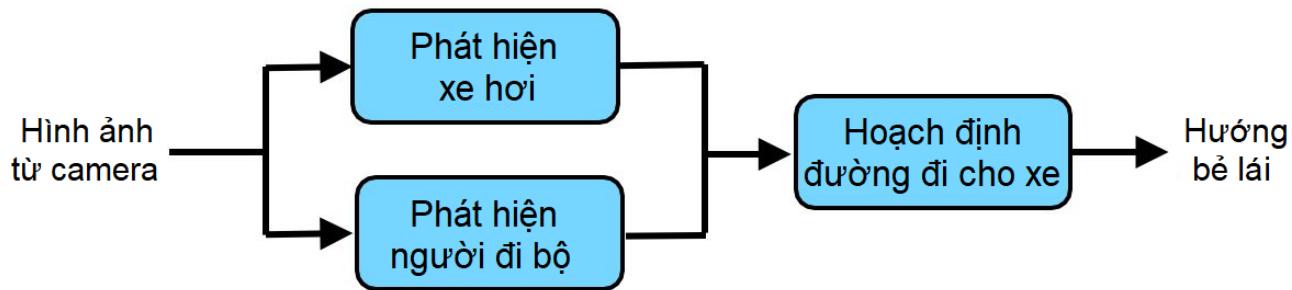


Bước thứ hai đưa những phần ảnh được cắt ra từ bộ phát hiện mèo (từng phần một) vào bộ phân loại mèo, và cuối cùng đưa ra 1 nếu có một phần bất kỳ được xác định là một con mèo Xiêm.



So với việc huấn luyện bộ phân loại thuần đầu-cuối chỉ sử dụng nhãn 0/1, mỗi trong hai thành phần trong pipeline -- bộ phát hiện mèo và bộ phân loại mèo -- có vẻ dễ hơn nhiều để học và đòi hỏi lượng dữ liệu ít hơn[17].

Ví dụ cuối cùng, cùng nhìn lại pipeline xe tự lái:



Bằng cách sử dụng pipeline này, bạn đang nói với thuật toán rằng có 3 bước chính để lái xe: (1) Phát hiện những chiếc xe hơi khác, (2) Phát hiện người đi bộ, và (3) Hoạch định đường đi cho xe của bạn. Ngoài ra, mỗi bước này là một hàm số tương đối đơn giản hơn -- và có thể được học với ít dữ liệu hơn -- so với hướng tiếp cận thuần đầu-cuối.

Tóm lại, khi lựa chọn các thành phần cho một pipeline, hãy cố gắng xây dựng một pipeline mà mỗi thành phần là một hàm số tương đối "đơn giản" sao cho nó có thể học được từ chỉ một lượng dữ liệu vừa phải.

CHÚ THÍCH:

[16] Lý thuyết thông tin có khái niệm về "Độ phức tạp Kolmogorov", lý thuyết này nói rằng độ phức tạp của hàm số học được là độ dài của chương trình máy tính ngắn nhất để có thể xây dựng thuật toán đó. Tuy nhiên, khái niệm lý thuyết này ít có ứng dụng thực tế trong AI. Xem thêm https://en.wikipedia.org/wiki/Kolmogorov_complexity

[17] Nếu bạn quen với các thuật toán thực tế về phát hiện vật thể, bạn sẽ nhận ra rằng chúng không chỉ học với ảnh có nhãn 0/1, và thay vào đó được huấn luyện với các khung chứa từ dữ liệu huấn luyện. Thảo luận về vấn đề này nằm ngoài phạm vi của chương này. Tham khảo khóa "Deep Learning specialization" trên Coursera (<http://deeplearning.ai>) nếu bạn muốn học thêm về thuật toán này.

52. Trực tiếp học những đầu ra phức tạp

Một thuật toán phân loại sẽ nhận đầu vào là một ảnh x rồi trả về một số nguyên thể hiện nhãn phân loại của đồ vật trong ảnh đó. Thay vào đó, liệu một thuật toán có thể đưa ra một câu mô tả hoàn chỉnh cho bức ảnh đó?

Ví dụ:



A yellow bus driving down a road with green trees and green grass in the background.

x =

y = "Một chiếc xe buýt màu vàng đang đi xuống một con đường với nền xanh của cây cỏ."

Những ứng dụng truyền thống của các thuật toán học có giám sát học một hàm: $X \rightarrow Y$, trong đó đầu ra y thường là một số nguyên hoặc một số thực. Ví dụ:

Bài toán	X	Y
Phân loại email rác	Email	email rác/ không rác(0/1)
Nhận dạng ảnh	Ảnh	Nhân số nguyên
Dự đoán giá nhà đất	Đặc trưng của căn nhà	Giá theo Đô-la
Gợi ý sản phẩm	Đặc trưng của sản phẩm và người dùng	Xác suất mua sản phẩm

Một trong những hướng phát triển thú vị nhất của học sâu đầu-cuối là nó cho phép chúng ta trực tiếp học những kết quả phức tạp hơn rất nhiều so với đầu ra của việc học truyền thống. Trong ví dụ chú thích hình ảnh ở trên, bạn có thể cho hình ảnh (x) vào một mạng nơ-ron và trực tiếp thu về một câu chú thích miêu tả hình ảnh đó (y)

Dưới đây là một số ví dụ khác:

Bài toán	X	Y	Trích dẫn ví dụ
Chú thích hình ảnh	Ảnh	Văn bản	Mao et al., 2014
Dịch máy	Văn bản tiếng Anh	Văn bản tiếng Pháp	Suskever et al., 2014
Hỏi đáp	Cặp (Văn bản, Câu hỏi)	Văn bản trả lời	Bordes et al., 2015
Nhận dạng giọng nói	Âm thanh	Bản ghi thoại	Hannun et al., 2015
Văn bản sang giọng nói	Đặc trưng văn bản	Âm thanh	van der Oord et al., 2016

Đây là một xu hướng đang ngày càng phát triển trong học sâu. Với các cặp có nhãn (đầu vào, đầu ra) phù hợp, đôi khi bạn có thể học đầu cuối ngay cả khi đầu ra là một câu, hình ảnh, âm thanh hoặc các đầu ra khác phức tạp hơn nhiều thay vì chỉ một số.

Phần 9: Phân tích lỗi từng phần

53. Phân tích lỗi từng phần

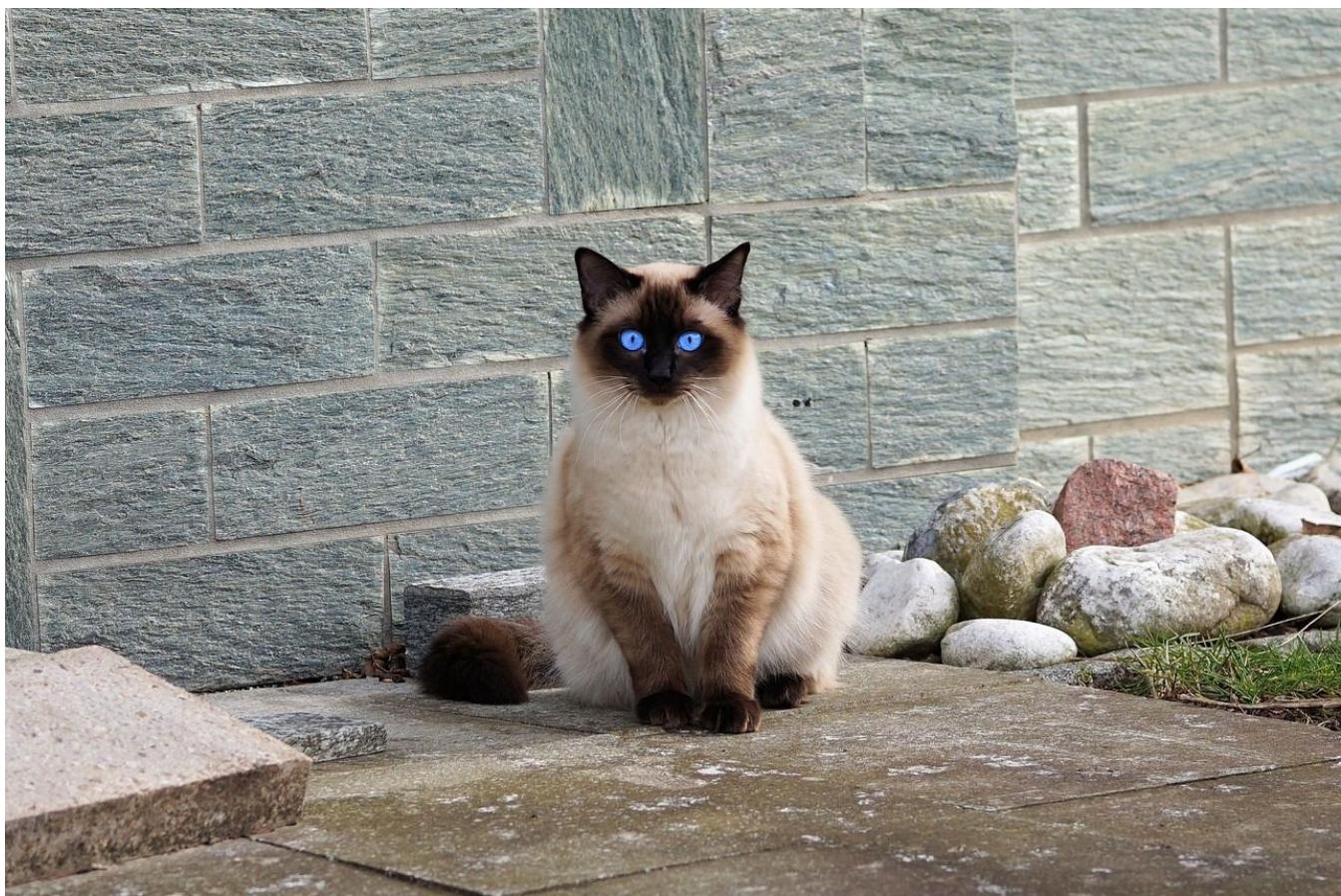
Giả sử hệ thống của bạn được xây dựng dựa trên một pipeline học máy phức tạp và bạn muốn cải thiện chất lượng của nó. Bạn nên cải thiện phần nào trong pipeline này? Bạn có thể sắp xếp thứ tự ưu tiên công việc bằng cách quy lỗi cụ thể cho từng phần trong pipeline.

Hãy sử dụng ví dụ bộ phân loại mèo Xiêm của chúng ta:

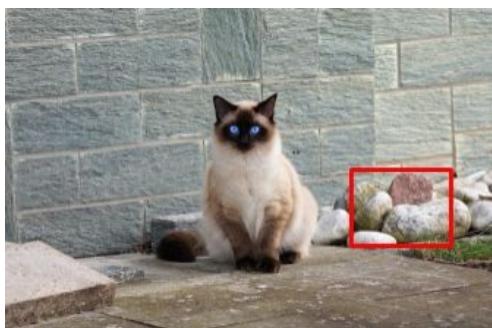


Phần đầu tiên, bộ phát hiện mèo, xác định vị trí của mèo và cắt chúng ra khỏi tấm ảnh. Phần thứ hai, bộ phân loại giống mèo, xác định xem đó có phải là một con mèo Xiêm hay không. Việc cải thiện bất kì bộ phận nào trong pipeline này cũng có thể tốn tới hàng năm trời. Làm sao để bạn quyết định được (những) bộ phận nào cần tập trung cải thiện?

Bằng việc thực hiện **phân tích lỗi từng phần**, bạn có thể cố quy trách nhiệm cho một (hoặc đôi khi là cả hai) phần trong pipeline trên từng dự đoán sai của thuật toán. Ví dụ, thuật toán phân loại sai tấm ảnh này không có một con mèo Xiêm ở trong đó ($y=0$) mặc dù nhãn chính xác là $y=1$.



Hãy kiểm chứng một cách thủ công xem hai bước của thuật toán đã làm gì. Giả sử bộ phát hiện mèo Xiêm đã phát hiện ra một chú mèo như dưới đây:

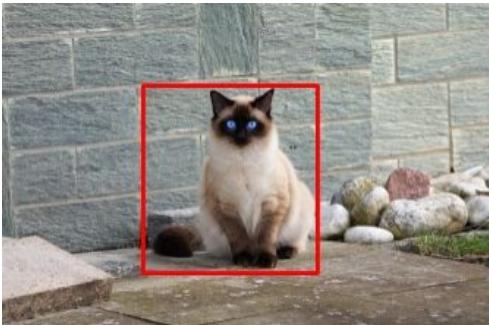


Tức là bộ phận loại giống mèo được đưa cho tấm hình sau:



Bộ phân loại giống mèo sau đó xác định chính xác rằng tấm hình này không có mèo Xiêm. Vậy nên, bộ phân loại giống mèo không có lỗi: Nó được đưa cho cho xem một đống đá và trả ra nhãn $y=0$ rất hợp lý. Thực tế, nếu một người mà phải phân loại tấm ảnh được cắt ra toàn đá ở trên thì cũng sẽ dự đoán $y=0$ mà thôi. Do vậy, bạn rõ ràng có thể quy lỗi này cho bộ phát hiện mèo.

Mặt khác, nếu giả sử bộ phát hiện mèo có cho ra kết quả khung chứa như dưới đây:



thì bạn sẽ kết luận rằng bộ phát hiện mèo đã hoàn thành công việc của nó, và lỗi là do bộ phân loại giống mèo mà ra.

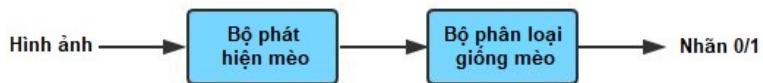
Giả sử bạn kiểm chứng 100 tấm ảnh bị phân loại nhầm trong tập phát triển và nhận ra rằng 90 trong số đó là do lỗi của bộ phát hiện mèo, chỉ có 10 tấm là do lỗi của bộ phân loại giống mèo. Bạn có thể an toàn kết luận rằng bạn nên tập trung nhiều hơn vào việc cải thiện bộ phát hiện mèo.

Ngoài ra, tiện đây bạn cũng đã tìm ra 90 mẫu mà bộ phát hiện mèo trả về khung chứa chưa chính xác. Bạn có thể sử dụng 90 mẫu này để thực hiện việc phân tích lỗi kĩ hơn trên bộ phát hiện mèo và tìm cách cải thiện nó.

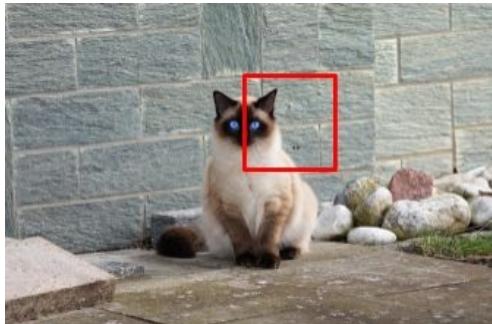
Việc làm thế nào để quy lỗi cho một phần của pipeline vẫn đang được mô tả một cách phi chính thống: bạn nhìn vào đầu ra của mỗi phần để xem liệu có thể quyết định phần nào gây ra lỗi. Phương pháp phi chính thống này có thể là đủ với bạn. Tuy nhiên trong chương sau, bạn sẽ thấy một cách chính thống hơn trong việc quy lỗi.

54. Quy lỗi cho một thành phần

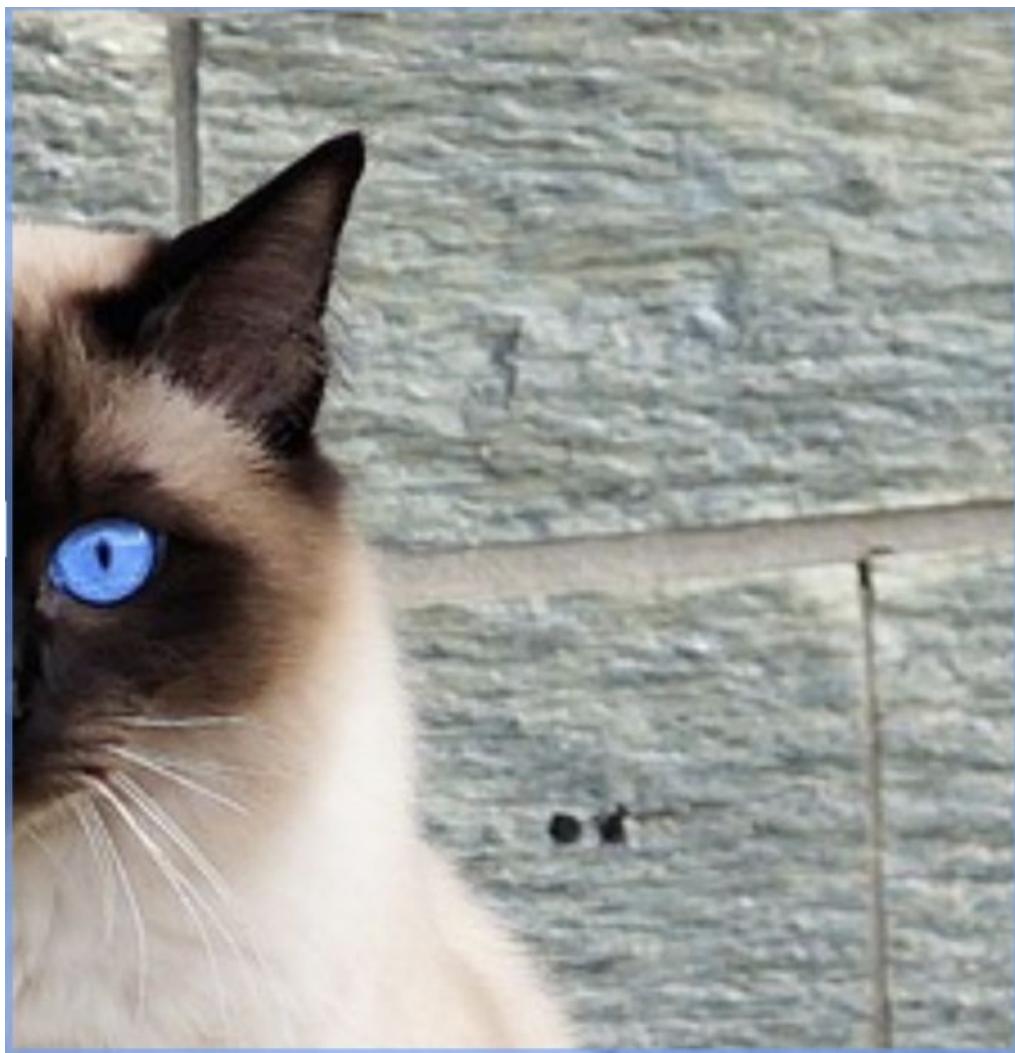
Cùng tiếp tục với ví dụ này:



Giả sử bộ phát hiện mèo cho kết quả khung chứa như sau:



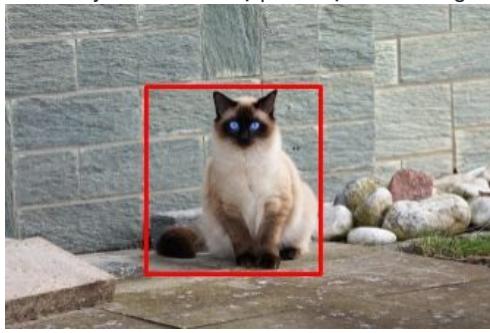
Khi đó bộ phân loại giống mèo nhận một ảnh bị cắt, và cho kết quả không chính xác là $y=0$, tức là không có con mèo nào trong hình.



Bộ phát hiện mèo đã hoạt động không tốt. Tuy nhiên, một người có kỹ năng vẫn có thể nhận dạng mèo Siamese từ bức ảnh bị cắt lệch. Trường hợp này chúng ta nên quy lỗi cho bộ phát hiện mèo, bộ phân loại giống mèo, hay là cả hai? Có sự không rõ ràng ở đây.

Nếu số lượng các trường hợp không rõ ràng là nhỏ, thì bất kỳ quyết định nào mà bạn lựa chọn đều sẽ đạt kết quả tương đương. Tuy nhiên một bài kiểm tra chính thức hơn sẽ giúp bạn quy lỗi chính xác cho một thành phần:

a. Thay đầu ra của bộ phát hiện mèo bằng một khung chứa thủ công:



b. Nạp ảnh bị cắt tương ứng vào bộ phân loại giống mèo. Nếu bộ phân loại giống mèo vẫn phân loại sai thì quy lỗi cho bộ phân loại giống mèo. Ngược lại thì quy lỗi cho bộ phát hiện mèo.

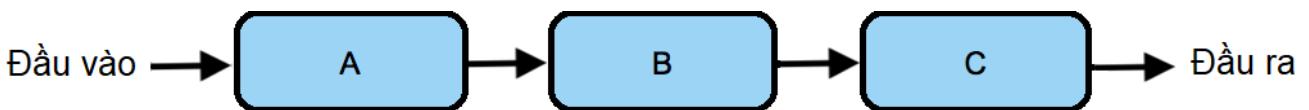
Nói cách khác, thực hiện thử nghiệm mà ở đó bạn cung cấp cho bộ phân loại giống mèo một đầu vào "hoàn hảo". Hai trường hợp có thể xảy ra:

- Trường hợp 1: Kể cả với một khung chứa "hoàn hảo", bộ phân loại giống mèo vẫn đưa ra kết quả không chính xác $y=0$. Trong trường hợp này rõ ràng là bộ phân loại giống mèo có lỗi.
- Trường hợp 2: Với một khung chứa "hoàn hảo", bộ phân loại giống mèo đưa ra kết quả chính xác $y=1$. Điều này cho thấy nếu bộ phát hiện mèo có thể đưa ra khung chứa chính xác hơn, thì kết quả tổng thể của toàn hệ thống sẽ được cải thiện. Trong trường hợp này bộ phát hiện mèo có lỗi.

Bằng cách phân tích các ảnh bị phân loại sai trên tập phát triển, bạn có thể quy lỗi chính xác cho một thành phần. Điều này cho phép bạn ước tính tỉ lệ lỗi cho từng thành phần của pipeline, từ đó quyết định thành phần cần tập trung khắc phục.

55. Trường hợp tổng quát của việc quy lỗi

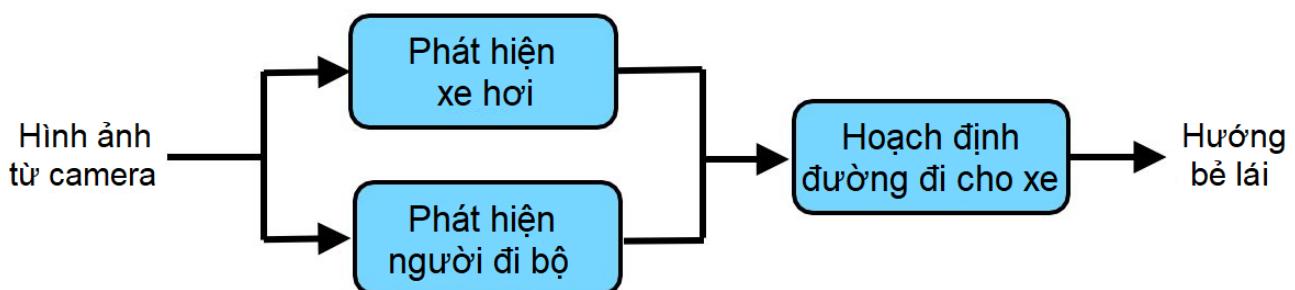
Đây là những bước tổng quát cho việc quy lỗi. Giả sử một thiết kế pipeline có ba thành phần A, B, C trong đó A cung cấp thông tin trực tiếp cho B, và B cung cấp thông tin trực tiếp cho C.



Với từng lỗi của hệ thống trên tập phát triển:

- Thử điều chỉnh thủ công kết quả đầu ra ở A cho "hoàn hảo" (ví dụ, một khung chứa hình mèo "hoàn hảo"), và sau đó tiến hành chạy thuật toán cho pipeline gồm có B và C với đầu ra này. Nếu thuật toán trả về kết quả cuối cùng chính xác, điều đó chỉ ra rằng, thuật toán sẽ cho ra kết quả chính xác nếu A trả về kết quả tốt hơn. Vậy ta có thể quy lỗi cho A. Nếu không, ta sẽ kiểm chứng thêm ở bước 2.
- Thử điều chỉnh thủ công kết quả đầu ra ở công đoạn B cho "hoàn hảo". Nếu thuật toán cho ra kết quả đầu ra cuối cùng chính xác, ta có thể quy lỗi cho B. Ngược lại, ta tiến hành bước 3.
- Quy lỗi cho thành phần C.

Chúng ta hãy cùng tìm hiểu một ví dụ phức tạp hơn sau đây:



Xe tự lái của bạn sử dụng pipeline như trên. Bạn sẽ sử dụng kỹ thuật phân tích lỗi từng phần như thế nào để quyết định (những) thành phần nào cần tập trung cải thiện?

Bạn có thể gọi tên ba thành phần trong hệ thống là A, B, C tương ứng với các chức năng như sau:

A: phát hiện xe hơi

B: phát hiện người đi bộ

C: hoạch định đường đi cho xe

Với hệ thống xe tự lái mô tả như trên, giả sử bạn kiểm tra xe của bạn trên một cung đường kín và xác định trường hợp nào xe chọn hướng bẻ lái giật nhiều hơn so với một người lái xe kinh nghiệm điều khiển. Trong lĩnh vực lái xe tự động, một trường hợp như thế thường được gọi là "tình huống". Bạn cần thực hiện:

- Thử điều chỉnh thủ công kết quả đầu ra của thành phần A (phát hiện xe hơi) sao cho "hoàn hảo" (ví dụ, cho xe biết vị trí của những chiếc xe khác). Sau đó tiếp tục chạy phần còn lại của pipeline gồm có B, C, nhưng cho phép C (hoạch định đường đi) sử dụng đầu ra đã hoàn hảo của A. Nếu thuật toán hoạch định đường đi cho xe tốt hơn, điều đó cho thấy rằng, kết quả cuối cùng của thuật toán tự lái sẽ được cải thiện nếu mà A trả về kết quả tốt hơn. Như vậy, bạn có thể quy lỗi cho A. Nếu không, ta tiếp tục bước 2.
- Thử điều chỉnh thủ công kết quả đầu ra ở công đoạn B (phát hiện người đi bộ) cho "hoàn hảo". Nếu thuật toán cho ra kết quả đầu ra cuối cùng chính xác, ta có thể quy lỗi cho B. Ngược lại, ta tiến hành bước 3.
- Quy lỗi cho thành phần C.

Các thành phần của một mô hình học máy dạng pipeline nên được sắp xếp theo đồ thị có hướng không chu trình (DAG), nghĩa là bạn có thể tính toán chúng theo thứ tự cố định từ trái sang phải nào đó, và các thành phần sau chỉ nên phụ thuộc vào đầu ra của các thành phần trước đó. Miễn là việc xâu chuỗi các thành phần theo thứ tự A->B->C tuân thủ theo quy tắc DAG, việc phân tích lỗi sẽ tốt. Bạn có thể nhận được các kết quả hơi khác nhau nếu hoán chuyển vị trí của A và B cho nhau như sau:

A: Nhận dạng người đi bộ (trước đây là "Nhận dạng xe")

B: Nhận dạng xe (trước đây là "Nhận dạng người đi bộ")

C: Hoạch định đường đi cho xe

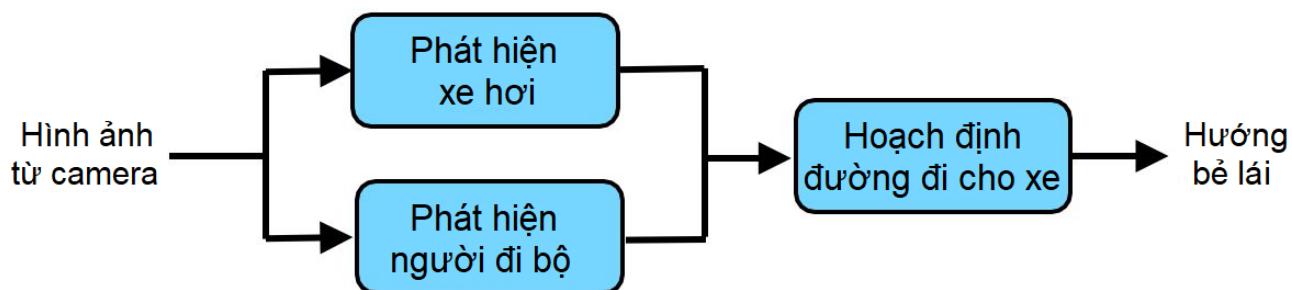
Nhưng các kết quả của việc phân tích lỗi thì sẽ vẫn hợp lệ và cho ta định hướng tốt thành phần nào cần cải thiện.

56. Phân tích lỗi từng phần và so sánh với chất lượng mức con người

Thực hiện phân tích lỗi của thuật toán học giống với việc sử dụng khoa học dữ liệu phân tích lỗi của hệ thống học máy để biết chính xác những việc cần làm kế tiếp. Cơ bản nhất, phân tích lỗi từng phần sẽ cho ta biết được chất lượng của (những) phần nào cần được cải thiện.

Giả sử bạn có bộ dữ liệu về khách hàng mua đồ trên một trang mạng. Một nhà khoa học dữ liệu có thể có rất nhiều cách khác nhau để phân tích dữ liệu đó. Người đó có thể đưa ra nhiều kết luận khác nhau như có nên tăng giá, giá trị vòng đời khách hàng đạt được thông qua các chiến dịch tiếp thị khác nhau, v.v. Không có một việc phân tích dữ liệu "chuẩn mực" nào, và có thể có rất nhiều kết luận hữu ích có thể rút ra. Tương tự, không chỉ có một cách "chuẩn mực" cho việc thực hiện phân tích lỗi. Thông qua các chương này bạn đã học được những cách phổ biến nhất để rút ra những nhận định chính xác về hệ thống học máy của bạn, nhưng bạn cũng nên thử nghiệm những phương pháp phân tích lỗi khác.

Chúng ta hãy quay trở lại ứng dụng xe tự lái, trong đó thuật toán phát hiện xe đưa ra vị trí (có thể có thêm vận tốc) của những chiếc xe gần đó, thuật toán phát hiện người đi bộ đưa ra vị trí của người đi bộ gần đó, và hai điều này cuối cùng được sử dụng để hoạch định đường đi cho xe.



Để kiểm tra lỗi pipeline này, thay vì tuân thủ nghiêm ngặt quy trình đã thấy trong chương trước, bạn nên đặt những câu hỏi như:

- Cách biệt về khả năng xác định xe giữa thuật toán và con người là bao xa?
- Cách biệt về khả năng phát hiện người đi bộ giữa thuật toán và con người là bao xa?
- Cách biệt giữa khả năng của toàn hệ thống và con người tới cỡ nào? Ở đây, chất lượng của con người được giả sử là cách con người tính đường đi cho xe chỉ dựa vào kết quả đầu ra từ hai thành phần trước đó trong pipeline (thay vì dựa vào hình ảnh camera). Nói cách khác, với cùng thông tin đầu vào, khả năng ước lượng đường đi của thuật toán so với con người sẽ như thế nào?

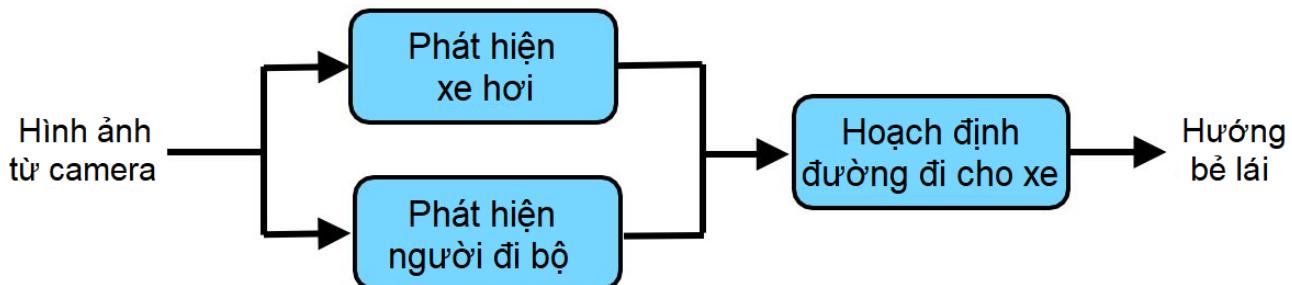
Nếu bạn thấy rằng một trong những thành phần này thua xa chất lượng mức con người, thì bây giờ bạn biết phần nào cần được cải thiện. Hãy tập trung vào việc cải thiện chất lượng của phần đó.

Nhiều quy trình phân tích lỗi hoạt động tốt nhất khi chúng ta cố gắng tự động hóa một thứ gì đó mà con người có thể làm, do đó có thể so sánh với con người. Hầu hết các ví dụ trước của chúng ta ngầm giả định điều này. Nếu bạn đang xây dựng một hệ thống học máy trong đó đầu ra hoặc một số thành phần trung gian đang làm những việc mà thậm chí con người không thể làm tốt, thì một trong số những quy trình này sẽ không được áp dụng.

Đây là một thuận lợi của việc giải quyết các vấn đề mà con người có thể giải quyết--bạn có các công cụ mạnh mẽ để phân tích lỗi, do đó bạn có thể ưu tiên các công việc trong nhóm một cách hiệu quả hơn.

57. Phát hiện một pipeline học máy bị lỗi

Nếu mỗi thành phần đơn lẻ trong pipeline học máy của bạn đều hoạt động ở chất lượng mức con người hoặc gần mức con người, nhưng pipeline tổng thể lại kém xa mức con người thì sao? Điều này thường có nghĩa là pipeline có lỗi và cần được thiết kế lại. Việc phân tích lỗi có thể giúp bạn nhận định liệu bạn có cần thiết kế lại pipeline của mình.



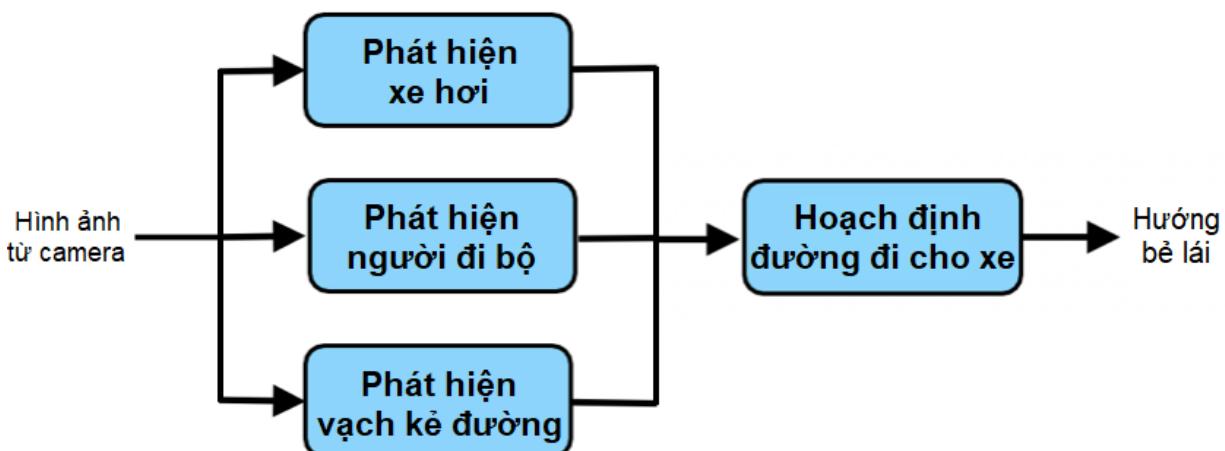
Trong chương trước, chúng ta đã đặt câu hỏi liệu mỗi trong số ba thành phần có ở chất lượng mức con người. Giả sử câu trả lời cho cả ba câu hỏi là có. Điều đó có nghĩa là:

- a. Thành phần Phát hiện xe hơi đạt chất lượng (xấp xỉ) mức con người trong việc phát hiện xe hơi từ ảnh camera.
- b. Thành phần Phát hiện người đi bộ đạt chất lượng (xấp xỉ) mức con người trong việc phát hiện xe hơi từ ảnh camera.
- c. *So sánh với một người phải lập kế hoạch đường đi cho xe khi chỉ dựa trên đầu ra của hai thành phần pipeline trước đó (thay vì được tiếp cận với hình ảnh từ camera), thành phần Lập kế hoạch có chất lượng ở mức tương đương.*

Tuy nhiên, chiếc xe tự lái tổng thể của bạn lại hoạt động kém hơn chất lượng mức con người một cách rõ rệt. Có nghĩa là, con người được tiếp cận hình ảnh từ camera có thể dự tính những đường đi tốt hơn nhiều cho xe. Bạn có thể rút ra kết luận gì?

Kết luận khả dĩ duy nhất là pipeline học máy đã bị lỗi. Trong trường hợp này, thành phần Lên kế hoạch đã hoạt động ở mức tốt nhất có thể với *những đầu vào của nó*, nhưng đầu vào không chứa đủ thông tin. Bạn nên tự hỏi liệu những thông tin nào khác, ngoài đầu ra của hai thành phần pipeline trước, là cần thiết cho việc lên kế hoạch đường đi thật tốt cho xe tự lái. Nói cách khác, những thông tin nào mà một người lái xe có kinh nghiệm cần đến?

Ví dụ, giả sử bạn nhận ra rằng người lái xe cũng cần biết vị trí của chỉ dấu làn đường. Điều này gợi ý rằng bạn nên thiết kế lại pipeline như sau:



Cuối cùng, nếu bạn không nghĩ rằng pipeline như một chỉnh thể sẽ đạt chất lượng mức con người, ngay cả khi mỗi thành phần đơn lẻ đạt chất lượng mức con người (nhớ rằng bạn đang so sánh với một người được cung cấp cùng một đầu vào như các thành phần), có nghĩa là pipeline có lỗi và cần được thiết kế lại.

GHI CHÚ:

[18] Trong ví dụ về xe tự lái ở trên, theo lý thuyết ta có thể giải quyết vấn đề bằng cách cũng cho hình ảnh thô từ camera vào thành phần lên kế hoạch. Tuy nhiên, điều đó sẽ vi phạm nguyên tắc thiết kế "Tính đơn giản của tác vụ" đã được trình bày ở Chương 51, vì thành phần lên kế hoạch đường đi giờ đây cần có đầu vào là ảnh thô và có một tác vụ rất phức tạp để giải quyết.

Thế nên thêm một thành phần Phát hiện chỉ dấu làn đường là một lựa chọn tốt hơn -- nó giúp lấy thêm những thông tin quan trọng vốn thiếu về làn đường cho khối lên kế hoạch đường đi, đồng thời bạn cũng tránh được việc làm bất cứ module nào trở nên quá phức tạp để xây dựng/huấn luyện.

Phần 10: Tổng kết

58. Xây dựng một biệt đội siêu anh hùng - Hãy để đồng đội của bạn đọc điều này

Chúc mừng bạn đã hoàn thành quyển sách này!

Trong chương 2, chúng ta đã nói về việc quyển sách này có thể giúp bạn trở thành siêu anh hùng trong nhóm của bạn.



Điều duy nhất tuyệt vời hơn trở thành một siêu anh hùng là trở thành một phần của một biệt đội siêu anh hùng. Tôi hi vọng bạn sẽ giới thiệu bản sao của quyển sách này cho bạn bè và đồng đội của bạn và tạo ra những siêu anh hùng khác.