Chương 4

HỌC MÁY (MACHINE LEARNING) VÀ HỌC SÂU (DEEP LEARNING)

Chương 4 trình bày những vấn đề sau: Học máy, các bước xây dựng mô hình học máy, đánh giá mô hình và học sâu. Đồng thời một số công cụ thông dụng trong phát triển các mô hình học máy và học sâu cũng được giới thiệu.

4.1. TỔNG QUAN

Học máy (tiếng Anh: machine learning) là một lĩnh vực của trí tuệ nhân tạo liên quan đến việc nghiên cứu và xây dựng các kĩ thuật cho phép các hệ thống "học" tự động từ dữ liệu để giải quyết những vấn đề cụ thể.

Một số định nghĩa khác về học máy như sau:

- ✓ Một quá trình nhờ đó một hệ thống cải thiện hiệu suất (hiệu quả hoạt động) của nó
- ✓ Một quá trình mà một chương trình máy tính cải thiện hiệu suất của nó trong một công việc thông qua kinh nghiệm
- ✓ Việc lập trình các máy tính để tối ưu hóa một tiêu chí hiệu suất dựa trên các dữ liệu ví dụ hoặc kinh nghiệm trong quá khứ.

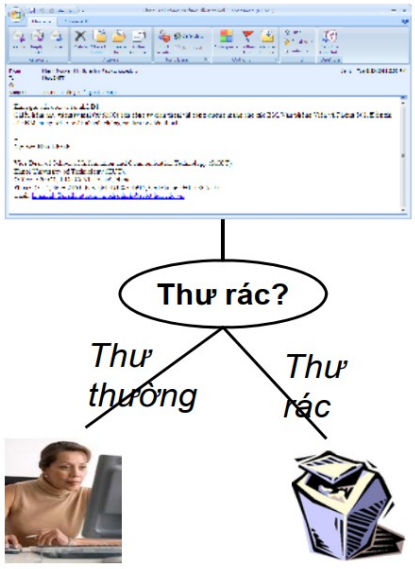
Một chương trình máy tính cần các quy tắc, luật lệ để có thể thực thi được một tác vụ nào đó như dán nhãn cho các email là thư rác nếu nội dung email có chứa từ khoá "quảng cáo". Nhưng với học máy, các máy tính có thể tự động phân lại các thư rác thành mà không cần chỉ trước bất kỳ quy tắc nào cả. Có thể hiểu đơn giản là nó giúp cho máy tính có được cảm quan và suy nghĩ được như con người. Nói một cách khác, học máy là phương pháp vẽ các đường thể hiện mối quan hệ của tập dữ liệu. Ví dụ như đường ngăn cách 2 loại dữ liệu cho nhãn khác nhau, đường thể hiện xu hướng của giá nhà phụ thuộc vào diện tích và trí hay các đường phân cụm dữ liệu.

Biểu diễn một bài toán học máy

Học máy = Cải thiện hiệu quả một công việc thông qua kinh nghiệm

- ✓ Một công việc (nhiệm vụ): T
- ✓ Các tiêu chí đánh giá hiệu năng (đánh giá mô hình): P
- ✓ Thông qua (sử dụng) kinh nghiệm: E

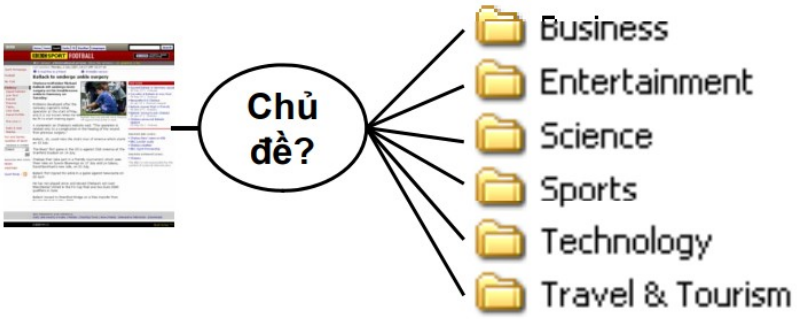
Ví dụ 4.1: Bài toán phân loại thư rác



Hình 4.1: Phân loại thư rác

- ✓ *T*: Dự đoán (để lọc) những thư điện tử nào là thư rác (p) spam email)
- ✓ *P*: Tỷ lệ % các thư điện tử gửi đến được phân loại chính xác Thư rác?
- ✓ *E*: Một tập các thư điện tử (emails) mẫu, mỗi thư điện tử được biểu diễn bằng một tập thuộc tính (vd: tập từ khóa) và nhãn lớp (thư thường/ thư rác) tương ứng

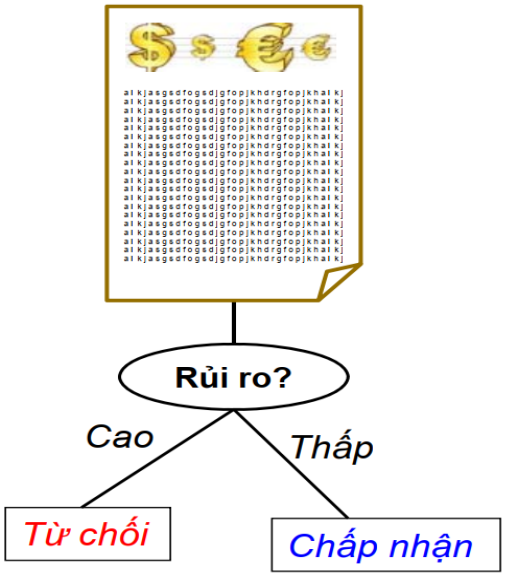
Ví dụ 4.2: Phân loại các trang Web



Hình 4.2: Phân loại trang web

- ✓ *T*: Phân loại các trang Web theo các chủ đề đã định trước
- ✓ *P*: Tỷ lệ (%) các trang Web được phân loại chính xác
- ✓ *E*: Một tập các trang Web, trong đó mỗi trang Web gắn với một chủ đề tương ứng

Ví dụ 4.3: Dự đoán rủi ro cho vay tài chính



Hình 4.3: Dự đoán rủi ro cho vay tài chính

- ✓ T: Xác định mức độ rủi ro (cao/thấp) đối với các hồ sơ xin vay tài chính
- ✓ P: Tỷ lệ (%) các hồ sơ xin vay có mức độ rủi ro cao (không trả lại tiền vay) được xác định chính xác
- ✓ E: Một tập các các hồ sơ xin vay bao gồm các thông tin liên quan và mức độ rủi ro.

Một nhánh nhỏ trong học máy gần đây rất được quan tâm là học sâu (*deep learning*). Học sâu là kỹ thuật sử dụng các mạng nơ-ron tương tự như các nơ-ron của não người để xây dựng hệ thống học máy. Đây là một sự kết hợp giữa toán học và khoa học thần kinh. Kết quả của nó mang lại cực kỳ to lớn, có thể coi là khởi nguyên của nhiều ngành công nghiệp mới. Tại thời điểm này, hầu hết các công ty lớn trong ngành công nghệ thông tin lẫn các ngành khác như ô tô, điện tử đều đang tập trung phát triển và ứng dụng kỹ thuật học sâu cho bài toán của mình. Ví dụ như AlphaGo của Google đã chiến thắng nhà vô địch cờ vây Lee Sedol vào tháng 3 năm 2016. Tính năng nhận diện khuôn mặt khá chính xác của Facebook được triển khai vào năm 2016. Trợ lý ảo Siri của Apple được giới thiệu từ năm 2006. Xe tự lái của Google được thử nghiệm chính thức trên đường phố vào năm 2015,…

4.2. PHÂN LOẠI KỸ THUẬT HỌC MÁY

Các giải thuật học máy được phân ra làm 2 loại chính là:

- ✓ **Học có giám sát** (*Supervised Learning*): Là phương pháp sử dụng những dữ liệu đã được gán nhãn từ trước để suy luận ra quan hệ giữa đầu vào và đầu ra. Các dữ liệu này được gọi là dữ liệu huấn luyện và chúng là cặp các đầu vào-đầu ra. Học có giám sát sẽ xem xét các tập huấn luyện này để từ đó có thể đưa ra dự đoán đầu ra cho 1 đầu vào mới chưa gặp bao giờ. Ví dụ: dự đoán giá nhà, phân loại email.
- ✓ **Học không giám sát** (*Unsupervised Learning*): Khác với học có giám sát, học phi giám sát sử dụng những dữ liệu chưa được gán nhãn từ trước để suy luận.

Phương pháp này thường được sử dụng để tìm cấu trúc của tập dữ liệu. Tuy nhiên lại không có phương pháp đánh giá được cấu trúc tìm ra được là đúng hay sai. Ví dụ như phân cụm dữ liệu, chiết xuất thành phần chính của một chất nào đó.

Ngoài ra còn có **học tăng cường** (*reinforcement learning*). Phương pháp học tăng cường tập trung vào việc làm sao để cho 1 tác tử trong môi trường có thể hành động sao cho lấy được phần thưởng nhiều nhất có thể. Khác với học có giám sát nó không có cặp dữ liệu gán nhãn trước làm đầu vào và cũng không có đánh giá các hành động là đúng hay sai.

4.3. XÂY DỰNG MÔ HÌNH DỰA TRÊN HỌC MÁY

4.3.1. Các bước xây dựng mô hình

Một bài toán học máy cần trải qua 03 bước chính:

- ✓ Chọn mô hình: Chọn một mô hình thống kê cho tập dữ liệu. Ví dụ như mô hình thống kê Bec-nu-li, mô hình phân phối chuẩn.
- ✓ Tìm tham số: Các mô hình thống kê có các tham số tương ứng, nhiệm vụ lúc này là tìm các tham số này sao cho phù hợp với tập dữ liệu nhất có thể.
- ✓ Suy luận: Sau khi có được mô hình và tham số, ta có thể dựa vào chúng để đưa ra suy luận cho một đầu vào mới nào đó.

Quy trình xây dựng mô hình học máy như sau:

- ✓ Thu thập dữ liệu: Thu thập dữ liệu để mô hình học
- ✓ Chuẩn bị dữ liệu: Xử lý và đưa dữ liệu về định dạng tối ưu, trích chọn đặc trưng hoặc giảm chiều dữ liệu
- ✓ Huấn luyện: Tại bước này, thuật toán học máy thực hiện việc học thông qua các ví dụ đã được thu thập và chuẩn bị từ hai bước trên
- ✓ Đánh giá: Kiểm thử mô hình để đánh giá xem chất lượng của mô hình tốt đến đâu
- ✓ Tinh chỉnh: Tinh chỉnh mô hình để tối ưu hiệu quả

Bất cứ một bài toán học máy nào cũng đều cần có dữ liệu để huấn luyện, ta có thể coi nó là điều kiện tiên quyết. Dữ liệu sau khi có được cần phải:

- ✓ **Chuẩn hoá**: Tất cả các dữ liệu đầu vào đều cần được chuẩn hoá để máy tính có thể xử lý được. Quá trình chuẩn hoá bao gồm số hoá dữ liệu, co giãn thông số cho phù hợp với bài toán. Việc chuẩn hoá này ảnh hưởng trực tiếp tới tốc độ huấn luyện cũng như cả hiệu quả huấn luyện.
- ✓ **Phân chia**: Việc mô hình được chọn rất khớp với tập dữ liệu đang có không có nghĩa là giả thuyết của ta là đúng mà có thể xảy ra tình huống dữ liệu thật lại không khớp. Vấn đề này trong học máy được gọi là khớp quá (*Overfitting*). Vì vậy khi huấn luyện người ta phải phân chia dữ liệu ra thành 3 loại để có thể kiểm chứng được phần nào mức độ tổng quát của mô hình. Cụ thể 3 loại đó là:
- ✓ **Tập huấn luyện** (*Training set*): Thường chiếm 60%, dùng để học khi huấn luyện.
- ✓ **Tập kiểm chứng** (*Cross validation set*): Thường chiếm 20%, dùng để kiểm chứng mô hình khi huấn luyện.
- ✓ **Tập kiểm tra** (*Test set*): Thường chiếm 20%, dùng để kiểm tra xem mô hình đã phù hợp chưa sau khi huấn luyện.

Lưu ý rằng, tập kiểm tra ta phải lọc riêng ra và không được sử dụng trong khi huấn luyện. Còn tập huấn luyện và tập kiểm chứng thì nên xáo trộn đổi cho nhau để mô hình của ta được huấn luyện với các mẫu ngẫu nhiên nhất có thể.

4.3.2. Một số vấn đề trong xây dựng mô hình

Giải thuật học máy (Learning algorithm):

- ✓ Những giải thuật học máy nào có thể học (xấp xỉ) một hàm mục tiêu cần học?
- ✓ Với những điều kiện nào, một giải thuật học máy đã chọn sẽ hội tụ (tiệm cận) hàm mục tiêu cần học?
- ✓ Đối với một lĩnh vực bài toán cụ thể và đối với một cách biểu diễn các ví dụ (đối tượng) cụ thể, giải thuật học máy nào thực hiện tốt nhất?

Các ví dụ học (Training examples):

- ✓ Bao nhiêu ví dụ học là đủ?
- ✓ Kích thước của tập học (tập huấn luyện) ảnh hưởng thế nào đối với độ chí h á nh xác của hàm mục tiê h u học được?
- ✓ Các ví dụ lỗi (nhiễu) và/hoặc các ví dụ thiếu giá trị thuộc tính (missing-value) ảnh hưởng thế nào đối với độ chính xác?

Quá trình học (Learning process):

- ✓ Chiến lược tối ưu cho việc lựa chọn thứ tự sử dụng (khai thác) các ví dụ học?
- ✓ Các chiến lược lựa chọn nào làm thay đổi mức độ phức tạp của bài toán học máy như thế nào?
- ✓ Các tri thức cụ thể của bài toán (ngoài các ví dụ học) có thể đóng góp thế nào đối với quá trình học?

Khả năng/giới hạn học (Learning capability):

- ✓ Hàm mục tiêu nào mà hệ thống cần học? Biểu diễn hàm mục tiêu: Khả năng biểu diễn (vd: hàm tuyến tính / hàm phi tuyến) và Độ phức tạp của giải thuật và quá trình học
- ✓ Các giới hạn (trên lý thuyết) đối với khả năng học của các giải thuật học máy?
- ✓ Khả năng khái quát hóa (generalize) của hệ thống từ các ví dụ học? Để tránh vấn đề "over-fitting" (đạt độ chính xác cao trên tập học, nhưng đạt độ chính xác thấp trên tập thử nghiệm). Vấn đề over-fitting thường do các nguyên nhân: Lỗi (nhiễu) trong tập huấn luyện (do quá trình thu thập/xây dựng tập dữ liệu); Số lượng các ví dụ học quá nhỏ, không đại diện cho toàn bộ tập (phân bố) của các ví dụ của bài toán học máy.
- ✓ Khả năng hệ thống tự động thay đổi (thích nghi) biểu diễn (cấu trúc) bên trong của nó? Để cải thiện khả năng (của hệ thống đối với việc) biểu diễn và học hàm mục tiêu

4.4. BÀI TOÁN PHÂN LỚP VÀ DỰ ĐOÁN

4.4.1. Khái niệm

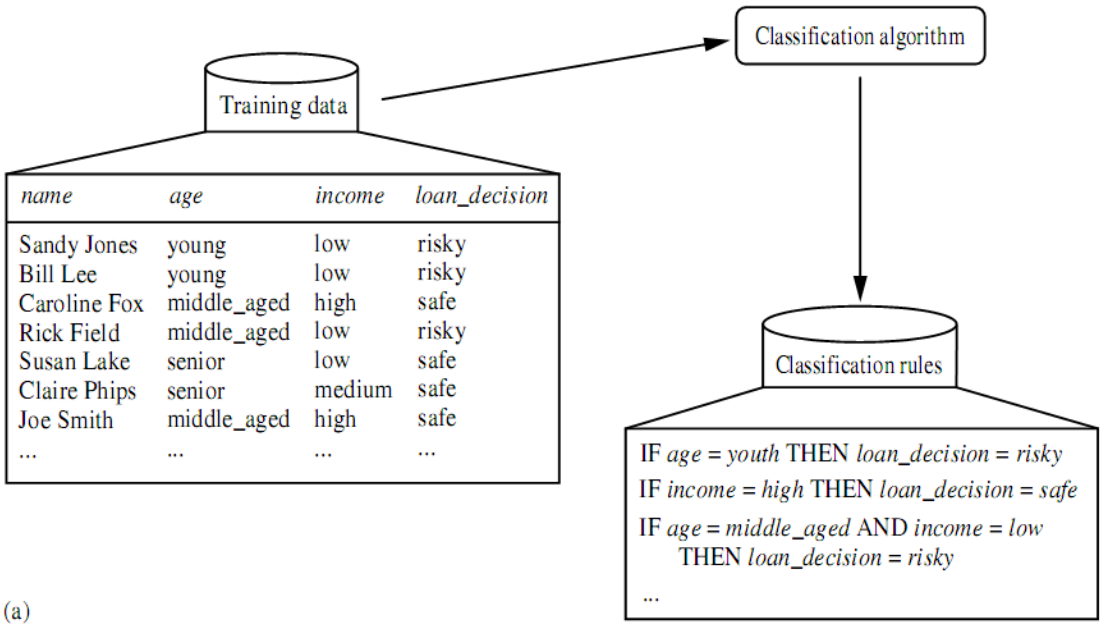
Tập dữ liệu luôn chứa rất nhiều các thông tin hữu ích có thể dùng cho việc ra các quyết định liên quan đến điều hành, định hướng của một đơn vị, tổ chức. Ví dụ chúng ta

có thể xây dựng một mô hình phân lớp để xác định một giao dịch cho vay của ngân hàng là an toàn hay có rủi ro, hoặc xây dựng mô hình dự đoán để phán đoán khả năng chi tiêu của các khách hàng tiềm năm dựa trên các thông tin liên quan đến thu nhập của họ. Phân lớp và dự đoán là hai dạng của quá trình phân tích dữ liệu được sử dụng để trích rút các mô hình biểu diễn các lớp dữ liệu quan trọng hoặc dự đoán các dữ liệu phát sinh trong tương lai. Nhiều các phương pháp phân lớp và dự đoán được nghiên cứu trong các lĩnh vực học máy. thước lớn.

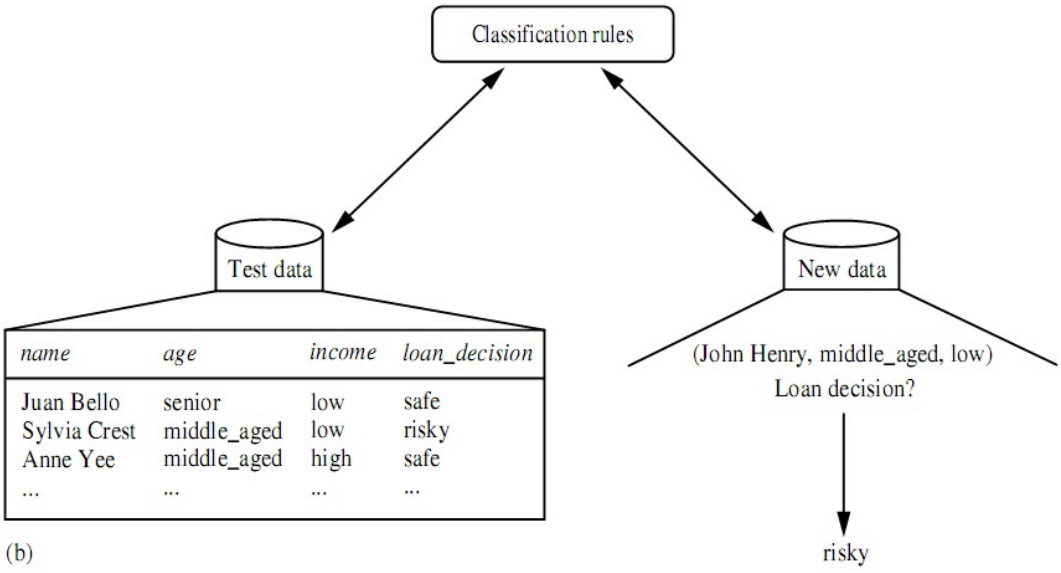
4.4.2. Phân lớp (classification)

Quá trình phân lớp thực hiện nhiệm vụ xây dựng mô hình các công cụ phân lớp giúp cho việc gán nhãn phân loại cho các dữ liệu. Ví dụ nhãn "An toàn" hoặc "Rủi ro" cho các yêu cầu vay vốn; "Có" hoặc "Không" cho các thông tin thị trường…. Các nhãn dùng phân loại được biểu diễn bằng các giá trị rời rạc trong đó việc sắp xếp chùng là không có ý nghĩa.

Phân lớp dữ liệu gồm hai quá trình. Trong quá trình thứ nhất một công cụ phân lớp sẽ được xây dựng để xem xét nguồn dữ liệu. Đây là quá trình học, trong đó một thuật toán phân lớp được xây dựng bằng cách phân tích hoặc "học" từ tập dữ liệu huấn luyện được xây dựng sẵn bao gồm nhiều bộ dữ liệu. Một bộ dữ liệu X biểu diễn bằng một vector n chiều, X = (x1, x2,…, xn) , đây là các giá trị cụ thể của một tập n thuộc tính của nguồn dữ liệu {A1, A2, …, An}. Mỗi bộ được giả sử rằng nó thuộc về một lớp được định nghĩa trước với các nhãn xác định.



Hình 4.4: Quá trình học



Hình 4.5: Quá trình phân lớp

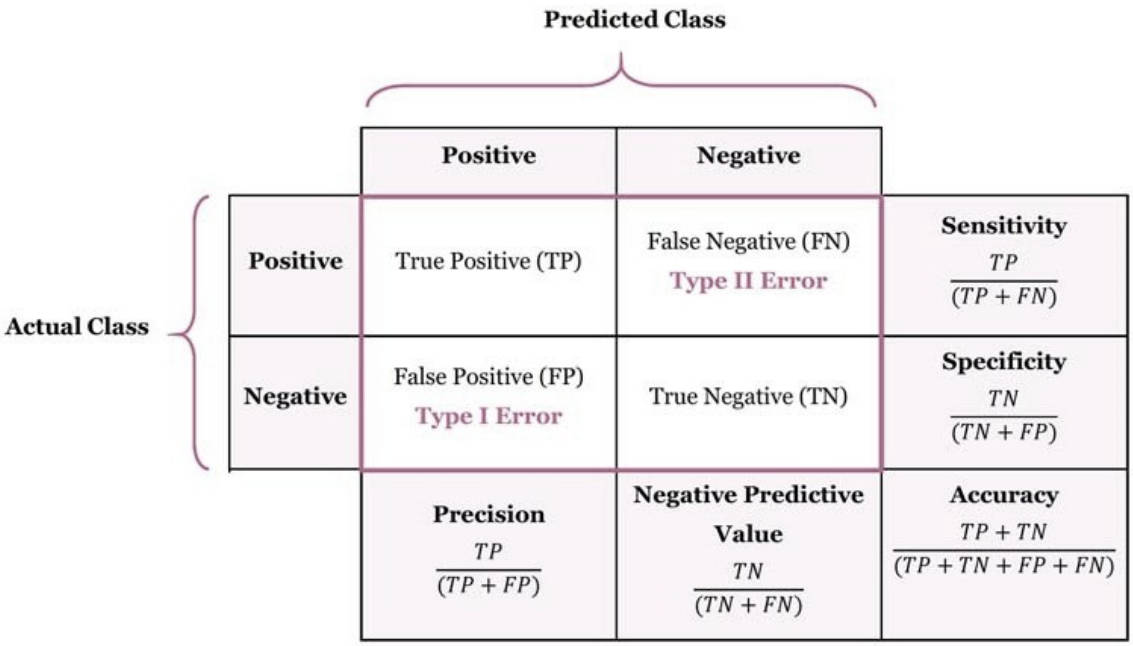
Quá trình đầu tiên của phân lớp có thể được xem như việc xác định ánh xạ hoặc hàm $y=f(X)$, hàm này có thể dự đoán nhãn y cho bộ X. Nghĩa là với mỗi lớp dữ liệu chúng ta cần học (xây dựng) một ánh xạ hoặc một hàm tương ứng.

Trong bước thứ hai, mô hình thu được sẽ được sử dụng để phân lớp. Để đảm bảo tính khách quan nên áp dụng mô hình này trên một tập kiểm thử hơn là làm trên tập dữ liệu huấn luyện ban dầu. Tính chính xác của mô hình phân lớp trên tập dữ liệu kiểm thử là số phần trăm các bộ dữ liệu kiểm tra được đánh nhãn đúng bằng cách so sánh chúng với các mẫu trong bộ dữ liệu huấn luyện. Nếu như độ chính xác của mô hình dự đoán là chấp nhận được thì chúng ta có thể sử dụng nó cho các bộ dữ liệu với thông tin nhãn phân lớp chưa xác định.

Đánh giá mô hình phân lớp

Để đánh giá mô hình phân lớp, một số các chỉ số được sử dụng trong nghiên cứu bao gồm: ma trận nhầm lẫn (confusion matrix) và độ chính xác tổng quát.

- Ma trận nhầm lẫn (confusion matrix): hay còn gọi là error matrix là một trong những phương pháp đánh giá mô hình phân loại quan trọng và phổ biến nhất, là cơ sở để hình thành cho các phương pháp đánh giá khác. Ma trận nhầm lẫn là một ma trận tổng quát thể hiện kết quả phân loại chính xác và kết quả phân loại sai được tạo ra bởi mô hình phân loại bằng cách so sánh với giá trị thật của biến mục tiêu (biến phân loại) của dữ liệu test. Ma trận có $N \times N$ với N là tổng số giá trị của biến mục tiêu (số class/nhóm của biến phân loại). Nếu chúng ta có bài toán phân loại chỉ bao gồm hai giá trị của biến mục tiêu là Có (1) hoặc Không (0), ta sẽ có một ma trận nhầm lẫn 2×2sau đây:



Hình 4.6: Ma trận nhầm lẫn

Chú ý: Positive/negative chỉ là tên gọi của 2 nhãn giá trị, mang tính quy ước và có ý nghĩa tương đối tùy theo mục tiêu của người dùng (Dương tính = điều ta quan tâm tìm kiếm và Âm tính = loại trừ (tất cả mọi) thứ còn lại).

- Độ chính xác tổng quát: là khả năng mô hình phân loại dự báo chính xác, hay xác định đúng class (nhóm, loại) cho dữ liệu cần phân loại. Độ chính xác tổng quát là tỉ lệ của tất cả trường hợp phân loại Đúng (không phân biệt negative/positive) trên toàn bộ trường hợp trong mẫu kiểm định.

$$Độ\ chính\ xác\ tổng\ quát = \frac{TP + TN}{(TP + TN + FP + FN)}$$

Ngoài ra, còn có các chỉ tiêu khác như độ nhạy (sensitivity) và độ đặc hiệu (specificity), …

4.4.3. Dự đoán (forecasting)

Dự đoán dữ liệu là một quá trình gồm hai bước, nó gần giống với quá trình phân lớp. Tuy nhiên để dự đoán, chúng ta bỏ qua khái niệm nhãn phân lớp bởi vì các giá trị được dự đoán là liên tục (được sắp xếp) hơn là các giá trị phân loại. Ví dụ thay vì phân loại xem một khoản vay có là an toàn hay rủi do thì chúng ta sẽ dự đoán xem tổng số tiền cho vay của một khoản vay là bao nhiêu thì khoản vay đó là an toàn.

Có thể xem xét việc dự đoán cũng là một hàm y = f(X), trong đó X là dữ liệu đầu vào, và đầu ra là một giá trị y liên tục hoặc sắp xếp được. Việc dự đoán và phân lớp có một vài điểm khác nhau khi sử dụng các phương pháp xây dựng mô hình. Giống với phân lớp, tập dữ liệu huấn luyện sử dụng để xây dựng mô hình dự đoán không được dùng để đánh giá tính chính xác. Tính chính xác của mô hình dự đoán được đánh giá dựa trên việc

tính độ lệch giá các giá trị dự đoán với các giá trị thực sự nhận được của mỗi bộ kiểm tra *X*.

Đánh giá mô hình dự đoán

Sai số dự báo là chênh lệch giữa giá trị thực (dữ liệu, giá trị mong muốn) và giá trị dự báo (giá trị đầu ra) nhằm đánh giá chất lượng hay sự phù hợp của mô hình dự báo tại cùng một thời điểm. Sai số dự báo cũng nhằm giúp điều chỉnh các thông số của mô hình dự báo.

Căn của sai số bình phương trung bình RMSE (Root Mean Squared Error):

$$RMSE=\sqrt{\frac{1}{m}\sum_{k=1}^{m}\left(t_k-y_k\right)^2}\ ,$$

Với t_k là giá trị mong muốn, y_k là giá trị dự báo của mô hình, m là tổng số mẫu.

Sai số tương đối trung bình MAPE (Mean Absolute Percent Error)

$$MAPE=\frac{1}{m}\sum_{k=1}^{m}\left|\frac{t_k-y_k}{t_k}\right|$$

Sai số tuyệt đối trung bình MAE (Mean Absolute Error)

$$MAE=\frac{1}{m}\sum_{k=1}^{m}\left|t_k-y_k\right|$$

Một mô hình dự báo được đánh giá tốt khi các sai số dự báo nhỏ. Ngoài ra tính ngẫu nhiên của sai số cũng là một tham số quan trọng để đánh giá độ chính xác của dự báo. Các tiêu chí MAE và MSE và RMSE có đặc tính, công năng như nhau và thường cho cùng một kết quả khi đánh giá. Tuy nhiên, nếu giá trị sai số $\varepsilon_t=t_k-y_t$ đều nhau thì nên chọn tiêu chí MSE để đánh giá.

Ngược lại, nếu các giá trị sai số $\varepsilon_t$quá khác biệt thì nên chọn tiêu chí MAE để đánh giá. Tiêu chí RMSE là căn bậc 2 của tiêu chí MSE nên hai tiêu chí về bản chất là một; điều khác biệt là giá trị của tiêu chí RMSE bé hơn.

Tiêu chí MAPE giúp đánh giá sai số một cách tương đối. Giả sử sai số trung bình là 1 đơn vị so với giá trị của dữ liệu là 100 thì vẫn là nhỏ (1%). Ngược lại, sai số trung bình 1 đơn vị so với giá trị của dữ liệu là 10 thì được xem là lớn (10%). Vậy nên khi đánh giá sai số dự báo với những bộ số liệu khác nhau thì nên sử dụng tiêu chí MAPE. Ngược lại, với cùng một bộ số liệu nhưng áp dụng nhiều phương pháp dự báo khác nhau thì không nên áp dụng tiêu chí MAPE vì tính phức tạp trong tính toán.

Hệ số tương quan R: Hệ số tương quan R đo lường mức độ phụ thuộc tuyến tính giữa giá trị thực tế và giá trị dự báo. Hệ số tương quan có giá trị từ -1 đến 1. Hệ số tương quan bằng 0 (hay gần 0) có nghĩa là hai biến số không có liên hệ gì với nhau; ngược lại nếu hệ số bằng -1 hay 1 có nghĩa là hai biến số có một mối liên hệ tuyệt đối. Nếu giá trị của hệ số tương quan là âm ($R<0$) có nghĩa là khi t tăng cao thì y giảm (và ngược lại,

khi t giảm thì y tăng); nếu giá trị hệ số tương quan là dương ($R>0$) có nghĩa là khi t tăng cao thì y cũng tăng, và khi y giảm cao thì y cũng giảm theo.

$$R=\frac{\sum_{k=1}^{m}\left(t_k-\bar{t}\right)\left(y_k-\bar{y}\right)}{\sqrt{\sum_{k=1}^{m}\left(t_k-\bar{t}\right)^2.\sum_{k=1}^{m}\left(y_k-\bar{y}\right)^2}}$$

,

Với $\bar{t}=\frac{1}{m}\sum_{k=1}^{m}t_k$ và $\bar{y}=\frac{1}{m}\sum_{k=1}^{m}y_k$ là các giá trị trung bình của t_k và y_k.

Chỉ số Theil U: Hệ số này được sử dụng để so sánh các mô hình dự báo, công thức như sau:

$$U=\frac{\sqrt{\sum_{k=1}^{m}\left(t_k-y_k\right)^2}}{\sum_{k=1}^{m}t_k^2+\sum_{k=1}^{m}y_k^2}$$

Giá trị U nằm trong khoảng từ 0 đến 1, U càng tiến về 0 thì mô hình dự báo càng chính xác.

4.5. MỘT SỐ KỸ THUẬT

4.5.1. k-NN(k-Nearest Neighbor) Thuật toán láng giềng gần nhất

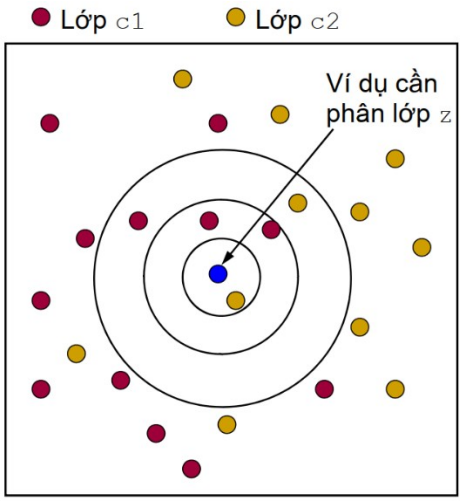
Mục tiêu của k-NN tìm ra k đối tượng – láng giềng "gần" với đối tượng X đang xét nhất. Từ k đối tượng tìm được, ta tính toán ước lược của X dựa trên các đánh giá của k đối tượng trên. Đại lượng "gần" ở đây có thể được đo bằng khoảng cách hoặc độ tương tự giữa các đối tượng với nhau.

Biểu diễn đầu vào của bài toán k-NN: Mỗi mẫu x được biểu diễn là một vectơ n chiều trong không gian các vectơ $X \in R_n$

$x=\left(x_1,x_2,\ldots,x_n\right)$, trong đó $x_i \in R$ là một số thực

Có thể áp dụng được với cả 2 kiểu bài toán học máy

- ✓ Bài toán phân lớp (classification): Hàm mục tiêu có giá trị rời rạc, Đầu ra của hệ thống là một trong số các giá trị rời rạc đã xác định trước (một trong các nhãn lớp)
- ✓ Bài toán dự đoán/hồi quy (prediction/regression): Hàm mục tiêu có giá trị liên tục (a continuous-valued target function). Đầu ra của hệ thống là một giá trị số thực.



Hình 4.7: Ví dụ bài toán phân lớp với thuật toán k-NN

Ví dụ xét bài toán phân lớp trong Hình 4.x.

- ✓ Xét 1 láng giềng gần nhất → Gán z vào lớp c2
- ✓ Xét 3 láng giềng gần nhất → Gán z vào lớp c1
- ✓ Xét 5 láng giềng gần nhất → Gán z vào lớp c1

Các bước của thuật toán k-NN như sau:

- ✓ Bước 1: Xác định tham số k là số láng giềng gần nhất.
- ✓ Bước 2: Tính toán khoảng cách giữa mẫu đang xét và những mẫu huấn luyện.
- ✓ Bước 3: Sắp xếp khoảng cách và xác định k khoảng cách nhỏ nhất.
- ✓ Bước 4: Thu thập giá trị thuộc tính của k láng giềng gần nhất
- ✓ Bước 5: Sử dụng giá trị trung bình của k láng giềng gần nhất để phán đoán giá trị của mẫu đang xét.

Việc phân lớp (hay dự đoán) chỉ dựa trên duy nhất một láng giềng gần nhất (là ví dụ học gần nhất với ví dụ cần phân lớp/dự đoán) thường không chính xác. Do nếu ví dụ học này là một mẫu bất thường, không điển hình (outlier), rất khác so với các mẫu khác. Nếu ví dụ học này có nhãn lớp (giá trị đầu ra) sai do lỗi trong quá trình thu thập (xây dựng) tập dữ liệu huấn luyện.

Thường xét $k(¿1)$ các ví dụ học (các láng giềng) gần nhất với ví dụ cần phân lớp/dự đoán.

Đối với bài toán phân lớp có 2 lớp, k thường được chọn là một số lẻ, để tránh cân bằng về tỷ lệ các ví dụ giữa 2 lớp, Ví dụ: $k=3,5,7,\ldots$

Hàm tính khoảng cách

Hàm tính khoảng cách d đóng vai trò rất quan trọng trong phương pháp học dựa trên các láng giềng gần nhất. Hàm tính khoảng cách thường được xác định trước, và không thay đổi trong suốt quá trình học và phân loại/dự đoán.

Lựa chọn hàm khoảng cách d:

Các hàm khoảng cách hình học (Geometry distance functions): Dành cho các bài toán có các thuộc tính đầu vào là kiểu số thực ($x_i \in R$).

- ✓ Hàm Minkowski (p-norm): $d(x,y)=\sum_{i=1}^{n}|x_i-z_i|$
- ✓ Hàm Manhattan (p=1): $d(x,y)=\sqrt{\sum_{i=1}^{n}(x_i-z_i)^2}$
- ✓ Hàm Euclid (p=2): $d(x,y)=\left(\sum_{i=1}^{n}(x_i-z_i)^p\right)^{\frac{1}{p}}$
- ✓ Hàm Chebyshev ($p=\infty$): $d(x,y)=\lim_{p\to\infty}(\sum_{i=1}^{n}|x_i-z_i|^p)^{\frac{1}{p}}=\max_i|x_i-z_i|$

Hàm khoảng cách Hamming: Dành cho các bài toán có các thuộc tính đầu vào là kiểu nhị phân ($x_i\in\{0,1\}$).

$$d(x,y)=\sum_{i=1}^{n}Difference(x_i,z_i)$$

$$Difference(a,b)=\begin{cases}1 & if(a\neq b)\\ 0 & if(a=b)\end{cases}$$

Hàm tính độ tương tự Cosine: Dành cho các bài toán phân lớp văn bản (x_i là giá trị trọng số TF/IDF của từ khóa thứ i).

$$d(x,y)=\frac{x.z}{||x||||z||}=\frac{\sum_{i=1}^{n}x_iz_i}{\sqrt{\sum_{i=1}^{n}x_i^2}\sqrt{\sum_{i=1}^{n}z_i^2}}$$

Các thuộc tính khác nhau có thể (nên) có mức độ ảnh hưởng n khác nhau đối với giá trị khoảng cách. Do đó, cần phải tích hợp (đưa vào) các giá trị trọng số của các thuộc tính trong hàm tính khoảng cách. Ví dụ: $d(x,y)=\sqrt{\sum_{i=1}^{n}w_i(x_i-z_i)^2}$, trong đó w_i là trọng số của thuộc tính i. Xác định trọng số của thuộc tính thông qua:

- ✓ Dựa trên các tri thức cụ thể của bài toán (vd: được chỉ định bởi các chuyên gia trong lĩnh vực của bài toán đang xét)
- ✓ Bằng một quá trình tối ưu hóa các giá trị trọng số (VD: sử dụng một tập học để học một bộ các giá trị trọng số tối ưu)

4.5.2. Naïve Bayes

Bộ phân lớp Bayes là một giải thuật thuộc lớp giải thuật thống kê, nó có thể dự đoán xác suất của một phần tử dữ liệu thuộc vào một lớp là bao nhiêu. Phân lớp Bayes được dựa trên định lý Bayes (Han, Kamber, & Pei, 2006). Giả sử có bộ dữ liệu huấn luyện (x_i,y_i), với $x_i=(x_{i1},x_{i2},\ldots,x_{in})$ là véc tơ n chiều và y_i là lớp tương ứng. Khi có một dữ liệu mới x_{tst}, để dự đoán lớp y_{tst} mà nó thuộc về, sẽ sử dụng công thức Bayes như sau:

$$y_{tst}=\underset{y}{\operatorname{argmax}}\,P(y|x_{tst})=\arg\max_y\frac{P(x_{tst}|y)P(y)}{P(x_{tst})}$$

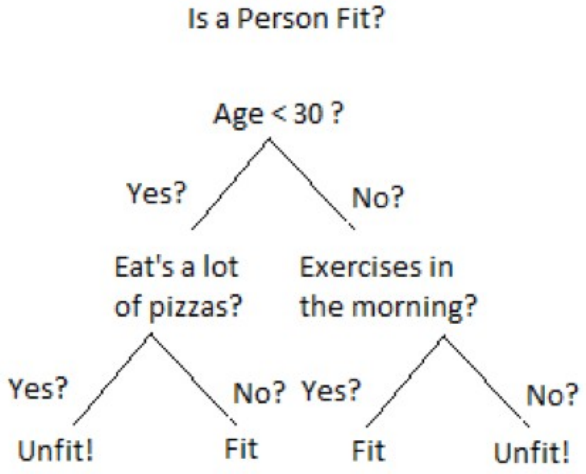
Tuy nhiên, trong công thức trên thì $P(x|y)$ trong nhiều trường hợp không xác định được hoặc chi phí tính toán lớn. Bộ phân lớp Naive Bayes làm giảm độ phức tạp bằng cách giả thiết các thuộc tính độc lập nhau, khi đó $P(x|y)$ có thể tính theo công thức sau:

$$P(x|y)=\prod_{j=1}^{n} P(x_j|y)$$

Trên thực tế thì ít khi tìm được dữ liệu mà các thành phần là hoàn toàn độc lập với nhau. Tuy nhiên giả thiết này giúp cách tính toán trở nên đơn giản, quá trình huấn luyện nhanh, đem lại hiệu quả với các lớp bài toán nhất định.

4.5.3. Cây quyết định (Decision Tree)

Cây quyết định (decision tree) là một trong những hình thức mô tả dữ liệu trực quan nhất, dễ hiểu nhất đối với người dùng. Cấu trúc của một cây quyết định bao gồm các nút và các nhánh. Nút dưới cùng được gọi là nút lá, trong mô hình phân lớp dữ liệu chính là các giá trị của các nhãn lớp (gọi tắt là nhãn). Các nút khác nút lá được gọi là các nút con, đây còn là các thuộc tính của tập dữ liệu, hiển nhiên các thuộc tính này phải khác thuộc tính phân lớp. Mỗi một nhánh của cây xuất phát từ một nút p nào đó ứng với một phép so sánh dựa trên miền giá trị của nút đó. Nút đầu tiên được gọi là nút gốc của cây. Hình 4.X là ví dụ một cây quyết định.



Hình 4.8: Ví dụ về cây quyết định

Mục đích của việc xây dựng một cây quyết định là khám phá ra một tập luật, từ đó có thể sử dụng để dự báo giá trị đầu ra từ những biến đầu vào. Cây quyết định có hai loại: cây hồi quy (Regression tree) ước lượng các hàm giá có giá trị là số thực và cây phân loại (Classification tree), nếu đầu ra là một biến phân loại như kết quả của một trận đấu (thắng hay thua). Cây quyết định giúp biểu diễn dữ liệu phức tạp thành một cấu trúc đơn giản hơn.

J.Ross Quinlan đã phát triển một thuật toán sinh cây quyết định. Đây là một tiếp cận tham lam, trong đó nó xác định một cây quyết dịnh được xây dựng từ trên xuống một cách đệ quy theo hướng chia để trị. Hầu hết các thuật toán sinh cây quyết định đều dựa trên tiếp cận top-down trình bày sau đây, trong đó nó bắt đầu từ một tập các bộ huấn luyện và các nhãn phân lớp của chúng. Tập huấn luyện được chia nhỏ một các đệ quy thành các tập con trong quá trình cây được xây dựng.

Generate_decision_tree: Thuật toán sinh cây quyết định từ các bộ dữ liệu huấn luyện của nguồn dữ liệu D

Đầu vào:

- Nguồn dữ liệu D, trong đó có chứa các bộ dữ liệu huấn luyện và các nhãn phân lớp
- Attribute_list - danh sách các thuộc tính
- Attribute_selection_method, một thủ tục để xác định tiêu chí phân chia các bộ dữ liệu một các tốt nhất thành các lớp. Tiêu chí này bao gồm một thuộc tính phân chia splitting_attribute, điểm chia split_point và tập phân chia splitting_subset.

Đầu ra: Một cây quyết định

Nội dung thuật toán:

1. Tạo nút N
2. **If** các bộ trong D đều có nhãn lớp C **then**
3. Trả về N thành một nút lá với nhãn lớp C
4. **If** danh sách thuộc tính attribute_list là rỗng **then**
5. Trả về N thành một nút là với nhãn là lớp chiếm đa số trong D (Việc này thực hiện qua gọi hàm **Attribute_selection_method**(D, attribute_list) để tìm ra tiêu chí phân chia tốt nhất splitting_criterion và gán nhãn cho N tiêu chí đó)
6. **If** splitting_attribute là một giá trị rời rạc và có nhiều cách chia **then**
7. Attribute_list = attribute_list – splitting_attribute // Loại bỏ thuộc tính splitting_attribute
8. **Foreach** j **in** splitting_criterion

   // Phân chia các bộ xây dựng cây cho các phân chia đó
9. Đặt D_j là tập các bộ trong D phù hợp với tiêu chí j
10. **If** D_j là rỗng **then**
11. Gắn nhãn cho nút N với nhãn phổ biến trong D
12. **Else** Gắn nút được trả về bởi hàm Generate_decision_tree(D_j, attribute_list) cho nút N
13. **Endfor**
14. **Return** N

Lựa chọn thuộc tính

Việc lựa chọn thuộc tính thực hiện nhờ việc lựa chọn các tiêu chí phân chia sao cho việc phân nguồn dữ liệu D đã cho một cách tốt nhất thành các lớp phân biệt. Nếu chúng ta chia D thành các vùng nhỏ hơn dựa trên các kết quả tìm được của tiêu chí phân chia, thì mỗi vùng sẽ khá là thuần chủng (Nghĩa là các tập các vùng đã phân chia có thể hoàn toàn thuộc về cùng một lớp). Điều này giúp xác định cách các bộ giá trị tại một nút xác định sẽ được chia thế nào. Cây được tạo cho phân vùng D được gán nhãn với tiêu chí phân chia, các nhánh của nó được hình thành căn cứ vào các kết quả phân chia của các bộ.

Giả sử D là một phân vùng dữ liệu chứa các bộ huấn luyện được gán nhãn. Các nhãn có m giá trị phân biệt xác định m lớp, Ci (với i = 1,..,m). Gọi Ci,D là tập các bộ của lớp Ci trong D

Thông tin cần thiết để phân lớp một bộ trong D cho bởi

$$Info(D) = -\sum_{i=1}^{N} p_i \log_2(p_i)$$

Trong đó pi là khả năng một bộ trong D thuộc về lớp Ci được xác định bởi |Ci,D| /| D|.

Giờ giả sử chúng ta phân chia các bộ D dựa trên một số thuộc tính A có v giá trị phân biệt {a1, .., av}. Thuộc tính A có thể dùng để chia D thành v phân vùng hoặc tập con {D1, D2, …, Dv} trong đó Dj chứa các bộ trong D có kết quả đầu ra aj. Các phân vùng đó sẽ tương đương với các nhánh của nút N.

Thông tin xác định xem việc phân chia đã gần tiếp cận đến một phân lớp được cho như sau

$$Info_A(D) = -\sum_{i=1}^{N} \frac{|D_j|}{|D|} \times Info(D_j)$$

$\frac{|D_j|}{|D|}$ là trọng lượng của phân vùng thứ j. InfoA(D) thể hiện thông tin cần thiết để phân lớp một bộ của D dựa trên phân lớp theo A. Giá trị thông tin nhỏ nhất sẽ cho ra phân vùng thuần túy tương ứng.

Độ đo thông tin thu được được cho

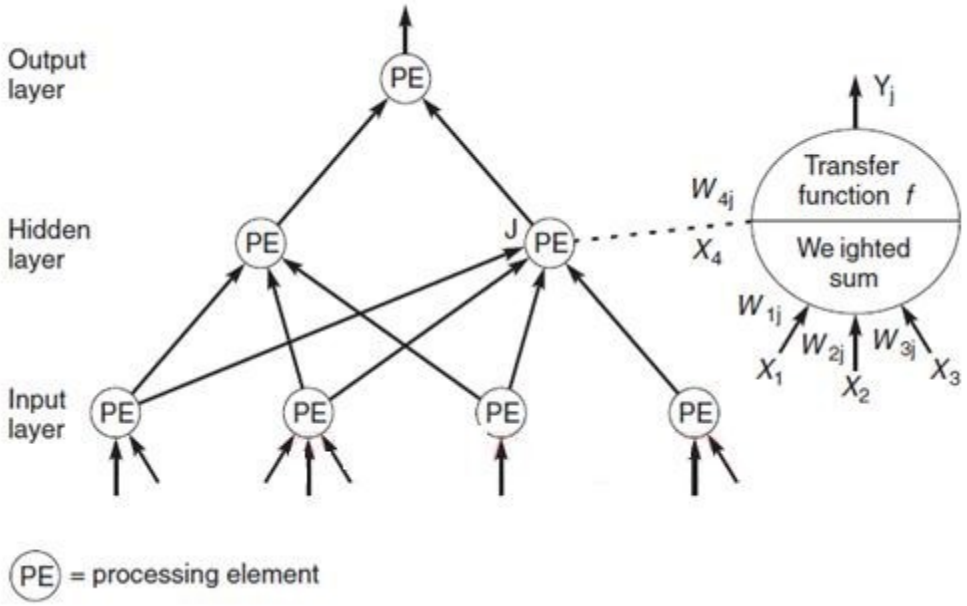
$$Gain(A) = Info(D) - Info_A(D)$$

Gain(A) sẽ cho chúng ta biết bao nhiêu nhánh có thể thu nhận được từ A. Thuộc tính A với độ đo thông tin thu được lớn nhất sẽ được dùng làm thuộc tính phân chia của nút N.

4.5.4. Mạng nơron nhân tạo (Artificial Neural Network - ANN)

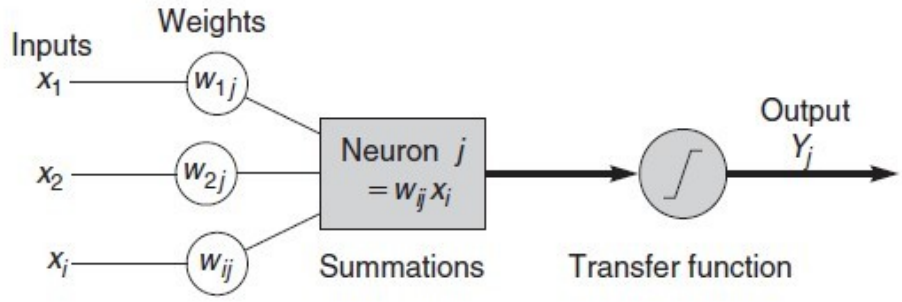
Mạng nơ-ron nhân tạo (Artificial Neural Network – ANN) là một mô phỏng xử lý thông tin hay thường được gọi ngắn gọn là mạng nơ-ron, được nghiên cứu dựa trên hệ thống thần kinh của sinh vật, giống như một bộ não con người để xử lý các thông tin. ANN bao gồm một số lượng lớn các mối gắn kết để xử lý các yếu tố làm việc trong mối quan hệ giải quyết vấn đề rõ ràng. Hoạt động của ANN giống như hoạt động bộ não của con người, được học hỏi bởi kinh nghiệm thông qua việc đào tạo, huấn luyện, có khả năng lưu giữ được các tri thức và sử dụng những tri thức đó trong việc phán đoán các dữ

liệu mới, chưa biết (unseen data). Processing Elements (PE) của ANN gọi là nơ-ron, nhận các dữ liệu vào (inputs) xử lý chúng và cho ra một kết quả (output) duy nhất. Kết quả xử lý của một nơ-ron có thể làm input cho các nơ-ron khác.



Hình 4.9: Kiến trúc mạng nơ-ron nhân tạo

Kiến trúc chung của ANN gồm 3 thành phần đó là input layer, hidden layer và output layer. Trong đó, lớp ẩn (hidden layer) gồm các nơ-ron, nhận dữ liệu input từ các nơ-ron ở lớp trước đó và chuyển đổi các input này cho các lớp xử lý tiếp theo. Quá trình xử lý thông tin của một ANN như sau:



Hình 4.10: Quá trình xử lý thông tin của một mạng nơ-ron nhân tạo

Trong đó, mỗi input tương ứng với 1 thuộc tính của dữ liệu. Ví dụ như trong ứng dụng của ngân hàng xem xét có chấp nhận cho khách hàng vay tiền hay không thì mỗi input là một thuộc tính của khách hàng như thu nhập, nghề nghiệp, tuổi, số con…Output

là một giải pháp cho một vấn đề, ví dụ như với bài toán xem xét chấp nhận cho khách hàng vay tiền hay không thì output là yes - cho vay hoặc no - không cho vay. Trọng số liên kết (Connection Weights) là thành phần rất quan trọng của một ANN, nó thể hiện mức độ quan trọng hay có thể hiểu là độ mạnh của dữ liệu đầu vào đối với quá trình xử lý thông tin, chuyển đổi dữ liệu từ layer này sang layer khác. Quá trình học (Learning Processing) của ANN thực ra là quá trình điều chỉnh các trọng số (Weight) của các input data để có được kết quả mong muốn. Hàm tổng (Summation Function) cho phép tính tổng trọng số của tất cả các input được đưa vào mỗi nơ-ron. Hàm tổng của một nơ-ron đối với n input được tính theo công thức sau:

$$Y = \sum_{i=1}^{n} X_i W_i$$

Kết quả trên cho biết khả năng kích hoạt của nơ-ron đó. Các nơ-ron này có thể sinh ra một output hoặc không trong ANN, hay nói cách khác rằng có thể output của 1 nơ-ron có thể được chuyển đến layer tiếp trong mạng nơ-ron hoặc không là do ảnh hưởng bởi hàm chuyển đổi (Transfer Function). Việc lựa chọn Transfer Function có tác động lớn đến kết quả của ANN. Vì kết quả xử lý tại các nơ-ron là hàm tính tổng nên đôi khi rất lớn, nên transfer function được sử dụng để xử lý output này trước khi chuyển đến layer tiếp theo. Hàm chuyển đổi phi tuyến được sử dụng phổ biến trong ANN là sigmoid (logical activation) function.

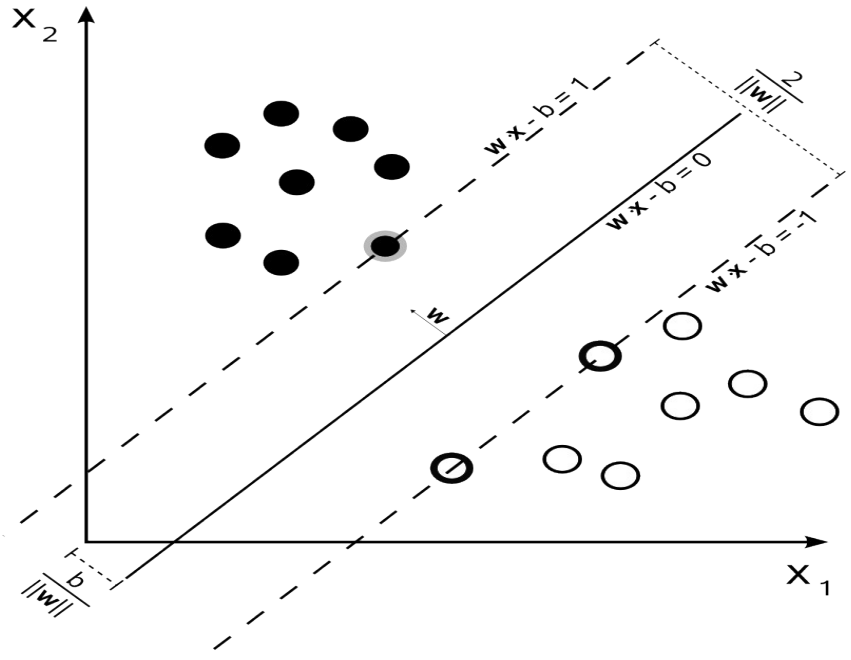
$$Y_T = \frac{1}{1+e^{-Y}}$$

Kết quả của Sigmoid Function thuộc khoảng [0, 1] nên còn gọi là hàm chuẩn hóa (Normalized Function). Đôi khi thay vì sử dụng hàm chuyển đổi, ta sử dụng giá trị ngưỡng (Threshold value) để kiểm soát các output của các nơ-ron tại một layer nào đó trước khi chuyển các output này đến các layer tiếp theo. Nếu output của một nơ-ron nào đó nhỏ hơn Threshold thì nó sẽ không được chuyển đến Layer tiếp theo. Ứng dụng thực tế của mạng nơ-ron thường được sử dụng trong các bài toán nhận dạng mẫu như nhận dạng chữ cái quang học (Optical character recognition), nhận dạng chữ viết tay, nhận dạng tiếng nói, nhận dang khuôn mặt.

4.5.5. Máy vectơ hỗ trợ (Support Vector Machine – SVM)

Máy véctơ hỗ trợ (Support Vectors Machine - SVM) là một mô hình học có giám sát. SVM thường được sử dụng để phân lớp dữ liệu (classification), hoặc phân tích hồi quy (regression analysis). SVM là nền tảng cho nhiều thuật toán trong khai thác dữ liệu. SVM nhận dữ liệu vào và phân loại chúng vào hai lớp khác nhau. Nguyên lý của SVM là xây dựng một bề mặt siêu phẳng (hyperplane) để phân lớp (classify) tập dữ liệu thành các lớp riêng biệt.

Bài toán cơ bản của SVM là bài toán phân loại hai lớp: Cho trước r điểm trong không gian n chiều (mỗi điểm thuộc vào một lớp kí hiệu là +1 hoặc –1), mục đích của giải thuật SVM là tìm một siêu phẳng (hyperplane) phân hoạch tối ưu cho phép chia các điểm này thành hai phần sao cho các điểm cùng một lớp nằm về một phía với siêu phẳng này. Hình 2.4 cho một minh họa phân lớp với SVM trong mặt phẳng.



Hình 4.11: Ví dụ siêu phẳng với lề cực đại trong không gian R2

Xét tập r mẫu huấn luyện $\{(x_1, y_1), (x_2, y_2), \ldots, (x_r, y_r)\}$. Trong đó xi là một vector đầu vào được biểu diễn trong không gian $X \in R$, y_i là một nhãn lớp; $y_i \in \{1, -1\}$. Siêu phẳng tối ưu phân tập dữ liệu này thành hai lớp là siêu phẳng có thể tách rời dữ liệu thành hai lớp riêng biệt với lề (margin) lớn nhất. Tức là, cần tìm siêu phẳng $H_0: y = w.x + b = 0$ và hai siêu phẳng $H_{+¿, H_{-¿¿}}$ hỗ trợ song song với H_0 và có cùng khoảng cách đến H_0. Với điều kiện không có phần tử nào của tập mẫu nằm giữa $H_{+¿¿}$ và $H_{-¿¿}$, khi đó:

$H_{+¿: w.x+b \geq +1¿}$ với $y = +1$

$H_{-¿: w.x+b \geq -1¿}$ với $y = -1$

Kết hợp hai điều kiện trên, có $y.(w.x + b) \geq 1$.

Khoảng cách của siêu phẳng $H_{+¿¿}$ và $H_{-¿¿}$ đến H_0 là $\frac{1}{\|w\|}$. Cần tìm siêu phẳng H_0 có lề lớn nhất, là giải bài toán tối ưu tìm w và b sao cho $\frac{2}{\|w\|}$ đạt cực đại với ràng buộc $y_i(w.x_i + b) \geq 1$. Tương đương với bài toán cực tiểu hóa $\frac{w.w}{2}$ với điều kiện $y_i(w.x_i + b) \geq 1$, mọi $i = 1 \ldots r$. Lời giải cho bài toán tối ưu này là cực tiểu hóa hàm Lagrange:

$$L(w, b, \alpha) = \frac{1}{2}\|w.w\| - \sum_{i=1}^{r} \alpha_i \left[y_i (\|w.x_i\| + b) - 1 \right]$$

Trong đó α là các hệ số Lagrange, $\alpha \geq 0$.

Lời giải tìm siêu phẳng tối ưu trên có thể mở rộng trong trường hợp dữ liệu không thể tách rời tuyến tính bằng cách ánh xạ dữ liệu vào một không gian có số chiều lớn hơn bằng cách sử dụng một hàm nhân K (Kernel). Một số hàm nhân thường dùng như:

Hàm tuyến tính có dạng $K(x, y) = x.y$

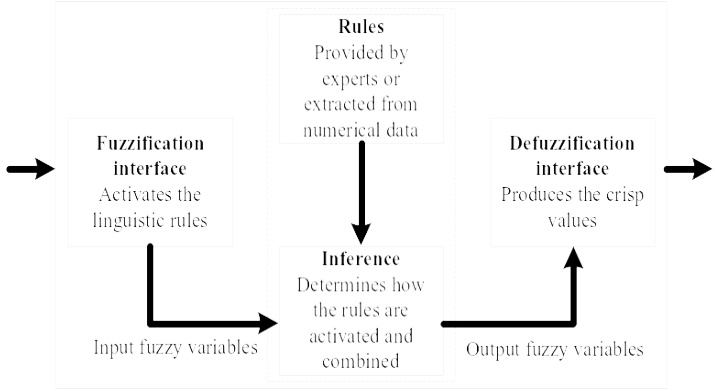
Hàm đa thức có dạng $K(x, y) = (x.y + 1)^d$

Hàm RBF có dạng: $K(x,y)=e^{\frac{-|x-y|^2}{2\alpha^2}}$

Với khả năng vượt trội của SVM về tính hiệu quả, độ chính xác, khả năng xử lý các bộ dữ liệu một cách linh hoạt, việc sử dụng máy vectơ hỗ trợ SVM đã và đang là sự lựa chọn tối ưu nhất trong việc giải quyết các bài toán phân loại/dự báo trong một số các lĩnh vực.

4.5.6. Hệ suy diễn mờ neural thích nghi (Adaptive neuro fuzzy inference system - ANFIS)

Mạng nơron và lý thuyết mờ là các công nghệ tính toán mềm được sử dụng để xây dựng các hệ thống thông minh. Hệ suy diễn mờ (FIS) sử dụng các luật mờ if-then khi thu thập tri thức từ các chuyên gia. Các hệ suy diễn mờ được sử dụng rộng rãi trong nhiều lĩnh vực như tối ưu hóa, điều khiển và xác định hệ thống. Một hệ suy diễn mờ FIS đơn giản được trình bày trong Hình 4.12. Tuy nhiên, hệ mờ không có khả năng tự học hỏi và điều chỉnh. Trong khi đó, mạng nơron có khả năng học hỏi từ môi trường, tự tổ chức và thích nghi. Chính vì lý do đó, việc kết hợp hệ suy diễn mờ với mạng nơron trong một hệ nơron mờ được sử dụng để tạo ra các hệ cơ sở luật mờ. Trong khi nhiệm vụ của các quy tắc if-then mờ là mô hình hóa kiến thức chuyên gia, thì mạng nơron thần kinh tối ưu hóa các hàm thành viên để giảm thiểu tỷ lệ lỗi trong đầu ra. Kiến trúc của hệ nơron mờ được đề xuất bởi Takagi and Hayashi. Trong các hệ nơron mờ, ANFIS, đề xuất bởi Jang, là phương pháp phổ biến nhất. Trong các hệ suy diễn mờ, các luật if-then mờ được xác định bởi chuyên gia thì trong ANFIS, nó được tự động tạo ra bởi dữ liệu đầu vào, đầu ra, và khả năng học của mạng nơron.



Hình 4.12: Hệ suy diễn mờ

ANFIS là một mạng nơ ron truyền thẳng nhiều lớp. Một ANFIS bao gồm năm lớp và mỗi lớp được hình thành bởi một số nút và chức năng nút. Có hai loại nút: nút thích ứng và nút cố định. Các nút thích ứng được đánh dấu bằng các ô vuông đại diện cho các bộ tham số, có thể được điều chỉnh. Các nút cố định được đánh dấu bằng các vòng tròn và các bộ tham số của chúng được cố định trong hệ thống. Kiến trúc của ANFIS gồm năm lớp: lớp mờ hóa, lớp luật, lớp chuẩn hóa, lớp giải mờ và một nút tổng hợp. Giả thiết một ANFIS hai đầu vào, x và y, hai luật, và một đầu ra, f, như trong Hình 4.13. Mỗi nút

trong cùng một lớp thực hiện chức năng giống nhau. Một FIS có hai đầu vào và hai luật mờ if-then được biểu diễn như sau:

Luật 1: *If* x *is* A_1 *and* y *is* B_1 *then* $f_1 = p_1 x + q_1 y + r_1$

Luật 2: *If* x *is* A_2 *and* y *is* B_2 *then* $f_2 = p_2 x + q_2 y + r_2$,

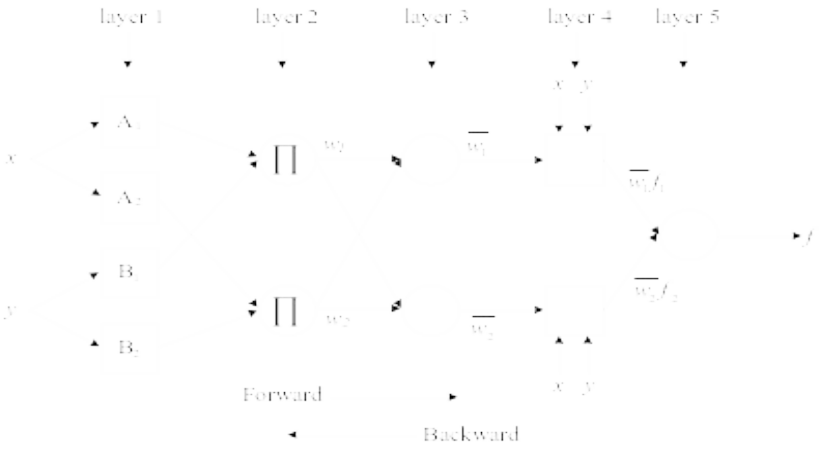
Trong đó x và y là các đầu vào; A_1, A_2, B_1, B_2 là các biến ngôn ngữ; p_i, q_i, và r_i, (i=1 or 2) là các tham số kết quả được xác định trong quá trình huấn luyện; và f_1 và f_2 là các giá trị đầu ra mờ. Công thức (2.2) thể hiện dạng 1 của luật mờ if-then trong đó đầu ra là một hàm tuyến tính. Giá trị đầu ra cũng có thể là một hằng số [26], như sau:

Luật 1: *If* x *is* A_1 *and* y *is* B_1 *then* $f_1 = C_1$

Luật 2: *If* x *is* A_2 *and* y *is* B_2 *then* $f_2 = C_2$,

Trong đó C_i (i=1 hoặc 2) là các giá trị hằng số. Công thức trên chính là dạng 2 của luật mờ if-then.

Đối với các bài toán phức tạp thì dạng 2 được sử dụng để xác định mối quan hệ giữa đầu vào và đầu ra.



Hình 4.13: Kiến trúc ANFIS với hai đầu vào và hai luật

Chức năng của các lớp trong Hình 4.13 được giải thích như sau:

Lớp 1 – lớp mờ hóa: Lớp đầu tiên chứa các nút thích nghi được đại diện bởi i. Các nút tạo các giá trị thành viên. Đầu ra của các nút được xác định qua công thức sau:

$O_{1,i} = \mu A_i(x), i=1, 2$ và

$O_{1,i} = \mu B_{i-2}(y), i=3, 4$

Với $O_{1,i}$ là đầu ra của nút i trong lớp 1, và $\mu A_i(x)$ và $\mu B_{i-2}(y)$ là các hàm thành viên mờ của A_i và B_{i-2}. Các hàm thành viên mờ có thể là dạng hình tam giác, hình thang, hàm Gaussian.

Lớp 2 – Lớp thứ hai là lớp quy tắc. Mỗi nút trong lớp này có hình tròn, có nhãn là Π, được gọi là các nút quy tắc. Một đầu ra từ các nút quy tắc đại diện cho một sản phẩm của các tín hiệu đầu vào. Nghĩa là, nút cố định nhận các đầu vào từ các nút thích nghi tương ứng, và mỗi giá trị đầu ra của nút biểu diễn cường độ của một quy tắc đã cho:

$$O_{2,i}=w_i=\mu A_i(x)\times\mu B_i(y),\ i=1,2$$

Với $O_{2,i}$ là đầu ra của nút i trong lớp 2 và w_i là cường độ một quy tắc.

Lớp 3 – chuẩn hóa: Trong lớp thứ ba, mỗi nút là một nút cố định hình tròn. Số lượng các nút trong lớp này là các nút số giống nhau trong lớp quy tắc. Nút thứ i trong lớp này được tính là tỷ lệ của cường độ quy tắc của nút thứ i so với tổng tất cả các cường độ của các quy tắc:

$$O_{3,i}=\overline{w_i}=\frac{w_i}{w_1+w_2},i=1,2$$

Với $O_{3,i}$ là đầu ra của nút i trong lớp 3, và $\overline{w_i}$ là cường độ quy tắc được chuẩn hóa.

Lớp 4 – giải mờ: Trong lớp thứ tư, mỗi nút là một nút thích nghi hình vuông. Số lượng các nút trong lớp này giống như số nút trong lớp thứ ba. Đầu ra từ mỗi nút là giá trị kết quả trọng số của một quy tắc nhất định:

$$O_{4,i}=\overline{w_i}f_i,\ i=1,2$$

Trong đó $O_{4,i}$ là đầu ra của nút i trong lớp 4, $\overline{w_i}$ là đầu ra của lớp 3, với $\{p_i,q_i,r_i\}$ là tập các giá trị. Các giá trị trong lớp này được gọi là các giá trị kết quả của mô hình Sugeno mờ.

Lớp 5 – tổng: Lớp thứ năm chỉ chứa một nút đầu ra, và được gọi là lớp tổng kết. Nút đơn này là một nút hình tròn và được biểu thị là ∑. Nút này tính tổng sản lượng của ANFIS, là tổng của các đầu ra của tất cả các nút thích nghi trong lớp thứ tư:

$$O_{5,i}=\sum_i \overline{w_i}f_i=\frac{\sum_i w_i f_i}{\sum_i w_i},\ i=1,2$$

Với $O_{5,i}$ là đầu ra của nút i trong lớp 5.

ANFIS sử dụng một thuật toán học lai để hiệu chỉnh mạng. Việc kết hợp thuật toán hồi phục lại với thuật toán xấp xỉ hoặc thuật toán truyền lại được sử dụng trong thuật toán học lai ghép để tối ưu hóa các tham số trong các lớp 1 và 4. Thuật toán lai ghép là

việc kết hợp hai phương pháp lan truyền ngược và phương pháp sai số bình phương nhỏ nhất.

Đối với mô hình mờ, mối quan hệ phi tuyến vào-ra phụ thuộc rất nhiều vào các phân vùng mờ của không gian vào-ra. Do đó việc điều chỉnh các tham số hàm liên thuộc (hàm thành viên) trong các mô hình mờ trở nên rất quan trọng. Trong mạng nơron mờ việc điều chỉnh này có thể xem như là vấn đề tối ưu dùng giải thuật học để giải quyết. Đầu tiên ta giả định các hàm liên thuộc có một hình dạng nhất định. Sau đó ta thay đổi các thông số của hình dạng đó qua quá trình học (huấn luyện) bằng mạng nơron. Như vậy ta cần một tập dữ liệu ở dạng các cặp vào-ra mong muốn để cho mạng nơron học và cũng cần phái có một bảng các luật ban đầu dựa trên các hàm phụ thuộc đó.

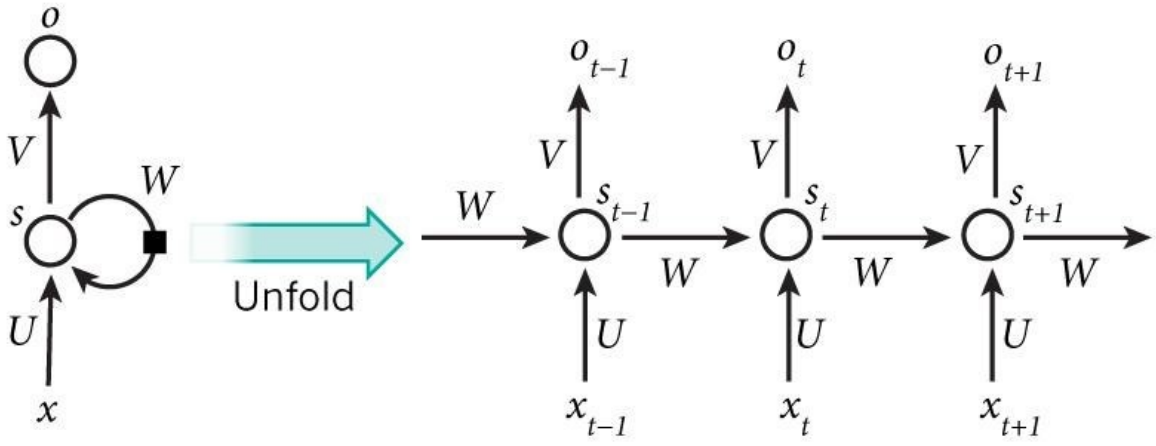
4.6. HỌC SÂU

4.6.1. Giới thiệu

Mạng nơ-ron hồi quy (RNN) là một trong những mô hình Deep Learning được đánh giá là có nhiều lợi thế trong các tác vụ xử lý ngôn ngữ tự nhiên (NLP). Ý tưởng của RNN là thiết kế một mạng lưới thần kinh có khả năng xử lý thông tin giống như chuỗi như một câu hỏi. Lặp lại có nghĩa là thực hiện cùng một nhiệm vụ lặp đi lặp lại cho từng yếu tố trong chuỗi. Cụ thể, đầu ra ở thời điểm hiện tại phụ thuộc vào kết quả tính toán của các thành phần ở thời điểm trước. Nói cách khác, RNN là một mô hình có bộ nhớ, có thể nhớ thông tin được tính toán trước đó, không giống như các mạng thần kinh truyền thống là thông tin đầu vào hoàn toàn độc lập với thông tin đầu ra. Về mặt lý thuyết, RNN có thể nhớ thông tin chuỗi có độ dài bất kỳ, nhưng trong thực tế mô hình này chỉ có thể nhớ thông tin từ một vài bước trước đó.

Các ứng dụng của RNN khá đa dạng trong các lĩnh vực như mô hình ngôn ngữ và tạo văn bản (Generating text). Mô hình ngôn ngữ cho chúng ta biết xác suất của câu trong ngôn ngữ là gì. Đây cũng là vấn đề dự đoán xác suất của từ tiếp theo của một câu nhất định. Từ vấn đề này, chúng ta có thể mở rộng sang vấn đề tạo văn bản (generative model/generating text). Mô hình này cho phép chúng tôi tạo văn bản mới dựa trên bộ dữ liệu đào tạo. Ví dụ, khi đào tạo mô hình này với dữ liệu tư vấn bán hàng, có thể tạo câu trả lời cho các câu hỏi liên quan đến thương mại điện tử. Tùy thuộc vào loại dữ liệu đào tạo, chúng tôi sẽ có nhiều loại ứng dụng khác nhau. Trong mô hình ngôn ngữ, đầu vào là một chuỗi các từ (được mã hóa thành one-hot vector), đầu ra là một chuỗi các từ được dự đoán từ mô hình này. Một lĩnh vực khác của RNN là Dịch máy. Vấn đề dịch máy tương tự như mô hình ngôn ngữ. Cụ thể, đầu vào là chuỗi các từ của ngôn ngữ nguồn (ví dụ: tiếng Việt), đầu ra là chuỗi các từ của ngôn ngữ đích (ví dụ: tiếng Anh). Sự khác biệt ở đây là đầu ra chỉ có thể dự đoán được khi đầu vào đã được phân tích hoàn toàn. Điều này là do từ được dịch phải chứa tất cả thông tin từ từ trước đó. Hoặc RNN có thể áp dụng cho các vấn đề toán học được mô tả cho hình ảnh Generating Image Descriptions). RNN kết hợp với Convolution Neural Netwokrs có thể tạo văn bản mô tả cho hình ảnh. Mô hình này hoạt động bằng cách tạo các câu mô tả từ các tính năng được trích xuất trong hình ảnh.

Huấn luyện RNN tương tự như đào tạo ANN truyền thống. Giá trị ở mỗi đầu ra không chỉ phụ thuộc vào kết quả tính toán của bước hiện tại mà còn phụ thuộc vào kết quả tính toán của các bước trước đó.



Hình 4.14: Mạng RNN

RNN có khả năng biểu diễn mối quan hệ phụ thuộc giữa các thành phần trong chuỗi. Ví dụ, nếu chuỗi đầu vào là một câu có 5 từ thì RNN này sẽ unfold (dàn ra) thành RNN có 5 layer, mỗi layer tương ứng với mỗi từ, chỉ số của các từ được đánh từ 0 tới 4. Trong hình vẽ ở trên, x_t là input (one-hot vector) tại thời điểm thứ t. S_t là hidden state (memory) tại thời điểm thứ t, được tính dựa trên các hidden state trước đó kết hợp với input của thời điểm hiện tại với công thức:

$$S_t = \tanh\left(U_{x_t} + W_{S_{t-1}}\right)$$

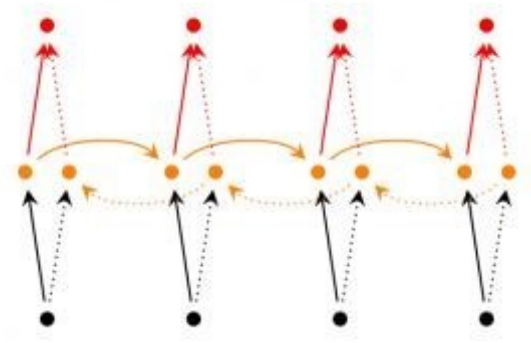
S_{t-1} là hidden state được khởi tạo là một vector không. $O_t$là output tại thời điểm thứ t, là một vector chứa xác suất của toàn bộ các từ trong từ điển.

$$O_t = softmax\left(V_{S_t}\right)$$

Không giống như ANN truyền thống, ở mỗi lớp cần sử dụng một tham số khác, RNN chỉ sử dụng một bộ tham số (U, V, W) cho tất cả các bước. Về mặt lý thuyết, RNN có thể xử lý và lưu trữ thông tin của một chuỗi dữ liệu có độ dài bất kỳ. Tuy nhiên, trong thực tế, RNN chỉ hiệu quả đối với chuỗi dữ liệu có độ dài không quá dài (short-term memory hay còn gọi là long-term dependency problem). Nguyên nhân của vấn đề này là do vấn đề độ dốc biến mất (gradient được sử dụng để cập nhật giá trị của weight matrix trong RNN và nó có giá trị nhỏ dần theo từng layer khi thực hiện back propagation). Khi độ dốc trở nên rất nhỏ (với giá trị gần bằng 0), giá trị của ma trận trọng số sẽ không được cập nhật nữa và do đó, mạng nơ-ron sẽ ngừng học ở lớp này. Đây cũng là lý do tại sao RNN không thể lưu trữ thông tin của các dấu thời gian đầu tiên trong một chuỗi dữ liệu dài. Trong vài năm qua, các nhà khoa học đã nghiên cứu và phát triển nhiều RNN ngày càng tinh vi để giải quyết các hạn chế của RNN.

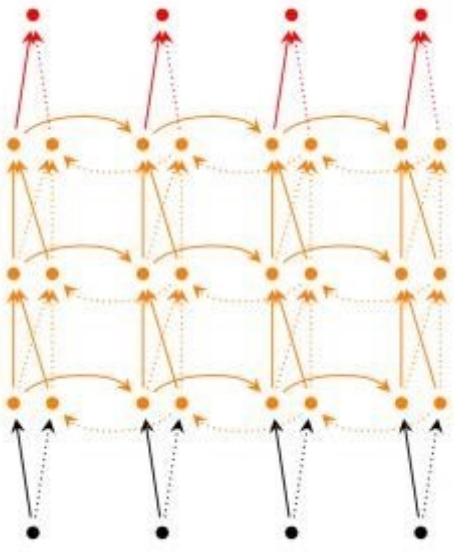
Bidirectinal RNN (2 chiều): dựa trên ý tưởng rằng đầu ra tại thời điểm t không chỉ phụ thuộc vào các thành phần trước mà còn phụ thuộc vào các thành phần trong tương lai. Ví dụ, để dự đoán một từ còn thiếu trong chuỗi, chúng ta cần xem xét các từ trái và

phải xung quanh từ đó. Mô hình này chỉ bao gồm hai RNN chồng chéo. Cụ thể, trạng thái ẩn được tính toán dựa trên cả hai thành phần bên trái và bên phải của mạng.



Hình 4.15: Mạng RNN 2 chiều

Deep RNN: tương tự như Bidirectional RNN, sự khác biệt là mô hình này bao gồm nhiều lớp RNN hai chiều tại một thời điểm. Mô hình này sẽ cho chúng ta khả năng thực hiện các tính toán nâng cao nhưng yêu cầu đào tạo phải đủ lớn.



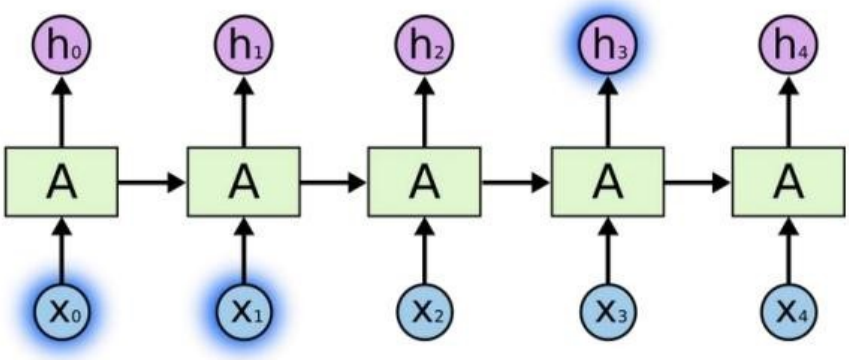
Hình 4.16: Mạng RNN nhiều tầng

Long short-term memory network (LSTM): mô hình này có cấu trúc giống như mạng RNN nhưng có cách tính khác cho các trạng thái ẩn. Bộ nhớ trong LSTM được gọi là các tế bào. Chúng ta có thể thấy đây là một hộp đen nhận thông tin đầu vào bao gồm cả

trạng thái và giá trị ẩn. Trong các hạt nhân này, họ xác định thông tin nào sẽ lưu trữ và thông tin nào cần xóa, để mô hình có thể lưu trữ thông tin dài hạn.

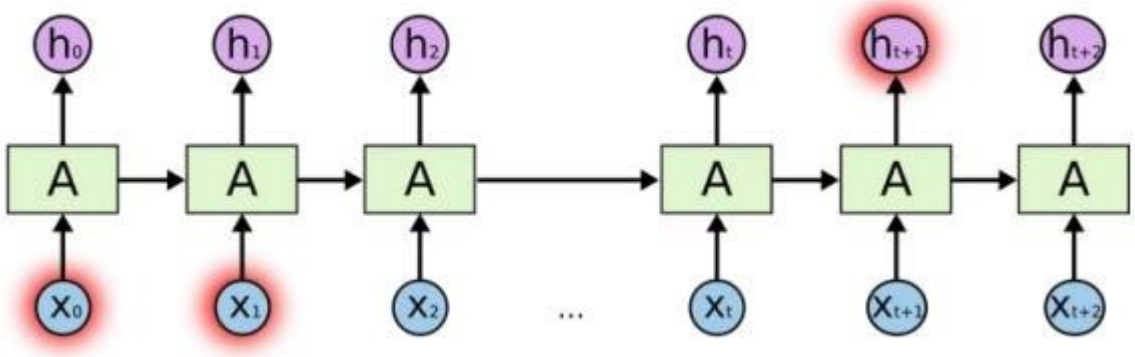
4.6.2. Vấn đề phụ thuộc quá dài

Ý tưởng ban đầu của RNN là liên kết thông tin trước đó để hỗ trợ các quy trình hiện tại. Nhưng đôi khi, chỉ cần dựa vào một số thông tin gần nhất để thực hiện nhiệm vụ hiện tại. Ví dụ, trong mô hình hóa ngôn ngữ, chúng tôi cố gắng dự đoán từ tiếp theo dựa trên các từ trước đó. Nếu chúng ta dự đoán từ cuối cùng trong câu "mây bay trên bầu trời", thì chúng ta không cần phải tìm kiếm quá nhiều từ trước đó, chúng ta có thể đoán từ tiếp theo sẽ là "bầu trời". Trong trường hợp này, khoảng cách đến thông tin liên quan được rút ngắn, nạng RNN có thể tìm hiểu và sử dụng thông tin trong quá khứ.



Hình 4.17: RNN phụ thuộc short-term

Nhưng cũng có những trường hợp chúng ta cần thêm thông tin, đó là phụ thuộc vào ngữ cảnh. Chẳng hạn, khi dự đoán từ cuối cùng trong đoạn "Tôi sinh ra và lớn lên ở Việt Nam ... tôi có thể nói tiếng Việt trôi chảy". Từ thông tin mới nhất cho thấy từ tiếp theo là tên của một ngôn ngữ, nhưng khi chúng ta muốn biết ngôn ngữ cụ thể, chúng ta cần quay lại quá khứ xa hơn, để tìm bối cảnh của Việt Nam. Và do đó, RRN có thể phải tìm thông tin liên quan và số điểm trở nên rất lớn. Thật bất ngờ, RNN không thể học cách kết nối thông tin lại với nhau:



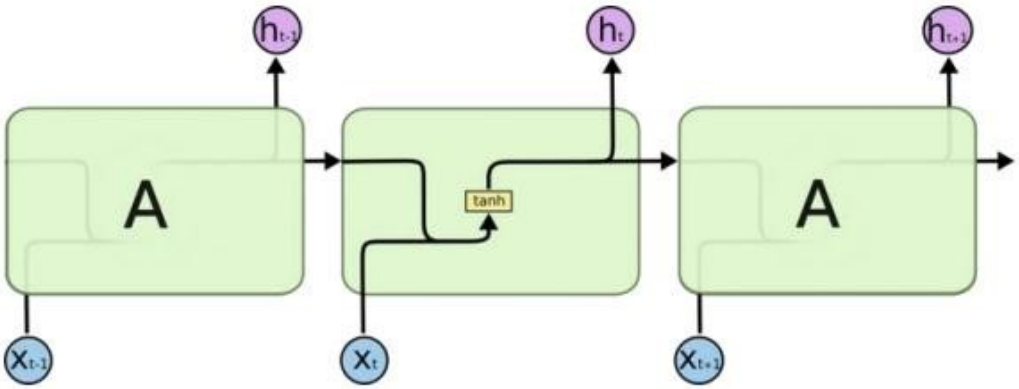
Hình 4.18: RNN phụ thuộc long-term

Về mặt lý thuyết, RNN hoàn toàn có khả năng xử lý "các phụ thuộc dài hạn", nghĩa là thông tin hiện tại thu được là do chuỗi thông tin trước đó. Đáng buồn thay, trong thực tế, RNN dường như không có khả năng này. Vấn đề này đã được đặt ra bởi Hochreiter (1991) [German] và và công sự đặt ra một thách thức cho mô hình RNN.

4.6.3 Kiến trúc mạng LSTM

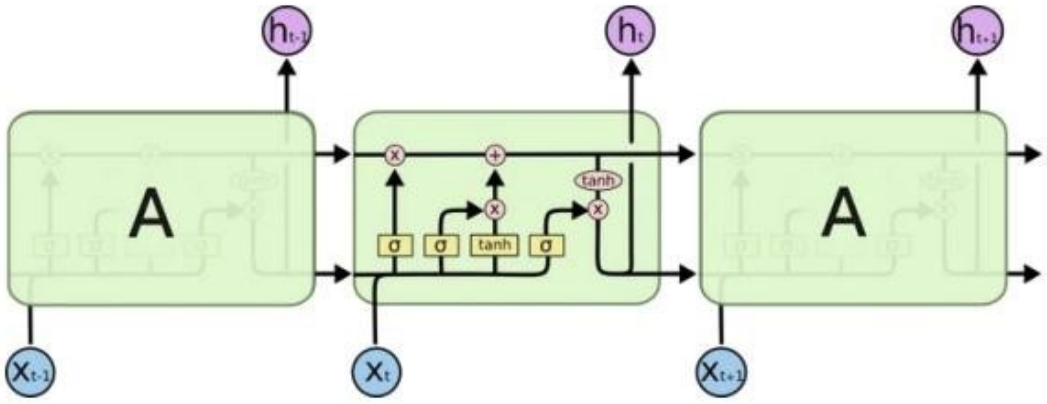
Long Short Term Memory network (LSTM) là trường hợp đặc biệt của RNN, có khả năng học các phụ thuộc dài hạn. Mô hình này được giới thiệu bởi Hochreiter & Schmidhuber (1997), và được cải tiến một lần nữa. Sau đó, mô hình này dần trở nên phổ biến nhờ các công trình nghiên cứu gần đây. Mô hình này tương thích với nhiều vấn đề, vì vậy nó được sử dụng rộng rãi trong các ngành liên quan.

LSTM được thiết kế để loại bỏ các vấn đề phụ thuộc quá dài. Nhìn vào mô hình RNN bên dưới, các lớp được kết nối với nhau thành các mô đun mạng thần kinh. Trong RNN tiêu chuẩn, mô-đun lặp lại này có cấu trúc rất đơn giản bao gồm một lớp đơn giản tanh layer.



Hình 4.19: Các mô-đun lặp của mạng RNN chứa một layer

LSTM có cùng cấu trúc liên kết, nhưng các mô-đun lặp lại có cấu trúc khác. Thay vì chỉ có một lớp mạng thần kinh, LSTM có tới bốn lớp, tương tác với nhau theo một cấu trúc cụ thể.



Hình 4.20: Các mô-đun lặp của mạng LSTM chứa bốn layer

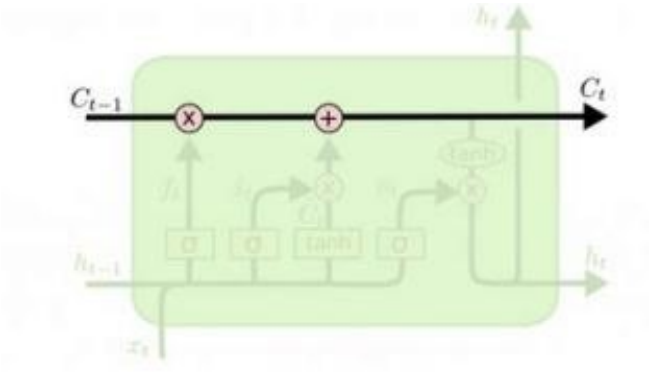
Trong đó, các ký hiệu sử dụng trong mạng LSTM được giải nghĩa như sau:

Neural Network Layer	Là các lớp ẩn của mạng nơ-ron
Pointwise Operation	Toán tử Pointwise, biểu diễn các phép toán như cộng, nhân vector
Vector Transfer	Vector chỉ đầu vào và đầu ra của một nút
Concatenate	Biểu thị phép nối các toán hạng
Copy	Biểu thị cho sự sao chép từ vị trí này sang vị trí khác

4.6.4. Phân tích mô hình LSTM

Mấu chốt của LSTM là trạng thái ô, đường ngang chạy dọc theo đỉnh của sơ đồ. Trạng thái tế bào giống như một băng chuyền. Nó chạy thẳng qua toàn bộ chuỗi, chỉ một

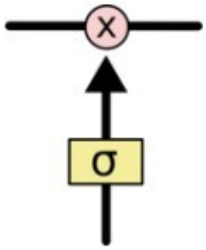
vài tương tác tuyến tính nhỏ được thực hiện. Điều này làm cho thông tin ít có khả năng thay đổi trong suốt quá trình lan truyền.



Hình 4.21: Tế bào trạng thái LSTM

LSTM có khả năng thêm hoặc bớt thông tin vào cell state, được quy định một cách cẩn thận bởi các cấu trúc gọi là cổng (gate). Các cổng này là một cách (tuỳ chọn) để định nghĩa thông tin băng qua. Chúng được tạo bởi hàm sigmoid và một toán tử nhân pointwise.

LSTM có khả năng thêm hoặc xóa thông tin về trạng thái tế bào, được điều chỉnh cẩn thận bởi các cấu trúc gọi là cổng (gate). Các cổng này là một cách (tùy chọn) để xác định thông tin truyền qua. Chúng được tạo bởi hàm sigmoid và toán tử pointwise.

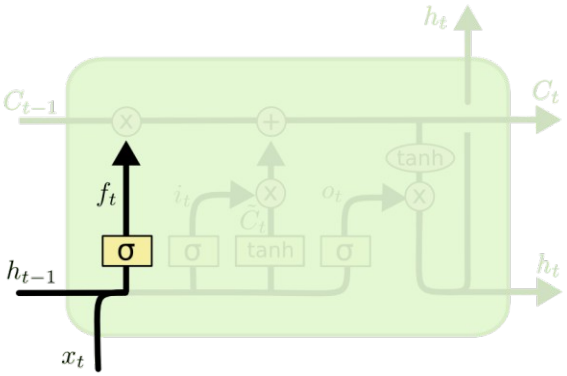


Hình 4.22: Cổng trạng thái LSTM

Hàm kích hoạt Sigmoid có giá trị từ [0 – 1], mô tả độ lớn thông tin được phép truyền qua tại mỗi lớp mạng. Nếu ta thu được 0 điều này có nghĩa là “không cho bất kỳ cái gì đi qua”, ngược lại nếu thu được giá trị là 1 thì có nghĩa là “cho phép mọi thứ đi qua”. Một LSTM có ba cổng như vậy để bảo vệ và điều khiển cell state.

4.6.5. Quá trình hoạt động của LSTM

Bước đầu tiên trong mô hình LSTM là quyết định thông tin nào chúng ta cần xóa khỏi trạng thái tế bào. Quá trình này được thực hiện thông qua một lớp sigmoid gọi là "lớp cổng chặn" - cổng chặn. Đầu vào là h_{t-1} và x_t, đầu ra là một giá trị trong phạm vi [0, 1] cho trạng thái ô C_{t-1}. 1 tương đương với "lưu giữ thông tin", 0 tương đương với "xóa thông tin"

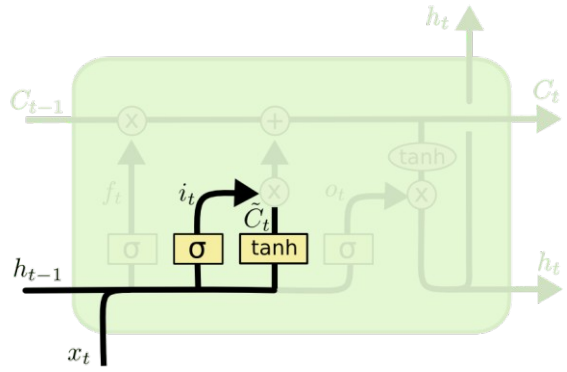


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Hình 4.23: LSTM focus f

Bước tiếp theo, ta cần quyết định thông tin nào cần được lưu lại tại cell state. Ta có hai phần. Một, single sigmoid layer được gọi là "input gate layer" quyết định các giá trị chúng ta sẽ cập nhật. Tiếp theo, một $tanh$ layer tạo ra một vector ứng viên mới, Ct được thêm vào trong ô trạng thái.

Trong bước tiếp theo, chúng ta cần quyết định thông tin nào cần được lưu trữ trong trạng thái tế bào. Chúng tôi có hai phần. Một, lớp sigmoid duy nhất được gọi là "lớp cổng đầu vào" xác định các giá trị chúng tôi sẽ cập nhật. Tiếp theo, một lớp creates tạo ra một vectơ ứng cử viên mới, Ct được thêm vào trong hộp trạng thái.

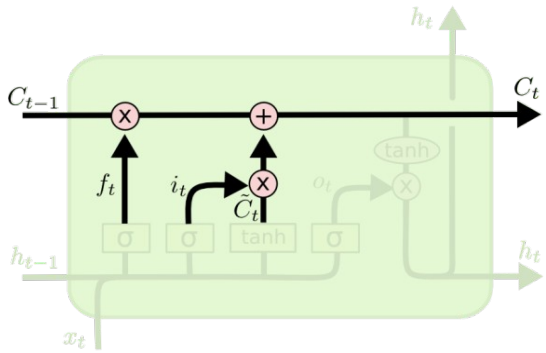


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Hình 4.24: LSTM focus I

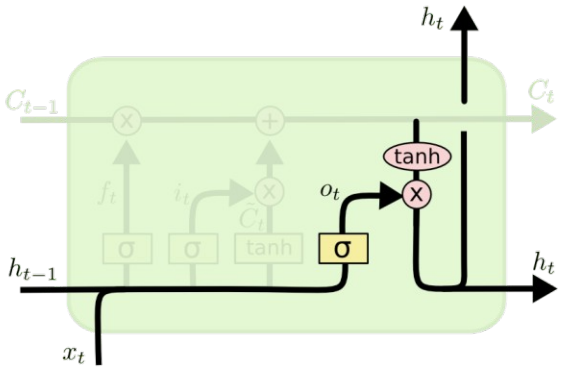
Ở bước tiếp theo, ta sẽ kết hợp hai thành phần này lại để cập nhật vào cell state. Lúc cập nhật vào cell state cũ, C_{t-1}, vào cell state mới C_t. Ta sẽ đưa state cũ hàm ff, để quên đi những gì trước đó. Sau đó, ta sẽ them $i_t * \widetilde{C}_t$. Đây là giá trị ứng viên mới, co giãn (scale) số lượng giá trị mà ta muốn cập nhật cho mỗi state.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Hình 4.25: LSTM focus c

Cuối cùng, ta cần quyết định xem thông tin output là gì. Output này cần dựa trên cell state của chúng ta, nhưng sẽ được lọc bớt thông tin. Đầu tiên, ta sẽ áp dụng single sigmoid layer để quyết định xem phần nào của cell state chúng ta dự định sẽ output. Sau đó, ta sẽ đẩy cell state qua tanh giá trị khoảng [-1 và 1] và nhân với một output sigmoid gate, để giữ lại những phần ta muốn output ra ngoài.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Hình 4.26: LSTM focus o

Ví dụ về một mô hình ngôn ngữ, chỉ cần nhìn vào chủ đề có thể cung cấp thông tin về một trạng từ theo sau. Ví dụ: nếu đầu ra của chủ ngữ là số ít hoặc số nhiều thì chúng ta có thể biết dạng trạng từ theo sau nó là gì.

4.7. MỘT SỐ CÔNG CỤ

4.7.1. Ngôn ngữ Python

4.7.1.1. Giới thiệu

Python là một ngôn ngữ lập trình đa mục đích được tạo ra vào cuối những năm 1980s, và được đặt tên theo nhóm kịch Monty Python, nó được sử dụng bởi hàng ngàn người để làm những việc từ kiểm thử vi mạch tại hãng Intel, sử dụng trong ứng dụng Instagram, cho tới xây dựng các video game với thư viện PyGame. Nó nhỏ và chặt chẽ như ngôn ngữ tiếng Anh, và có hàng trăm các thư viện của bên thứ ba (third-party).

Các ưu điểm nổi bật của ngôn ngữ lập trình Python:

- ✓ Cực mạnh trong việc xử lý các loại dữ liệu chuỗi, tập hợp. Thích hợp với ứng dụng bóc tách, chuyển đổi, phân tích dữ liệu: Big Data - Data Mining.
- ✓ Dễ học - dễ làm - dễ cài đặt.

- ✓ Chạy trên đa nền tảng: MacOSX, Windows, Linux.
- ✓ Lập trình gần như mọi thứ: Web (Django, Tornado), Game (pygame , kivy, piglet).
- ✓ Thư viện có sẵn nhiều. Ví dụ thư viện data mining Scikit-learn, Pandas...
- ✓ Tại thị trường tuyển dụng tại Việt nam: Python là hàng độc so với kỹ năng lập trình PHP, C#, Java.

Cú pháp Python rất dễ đọc

Python có điểm chặt chẽ rất giống với ngôn ngữ tiếng Anh, sử dụng những từ như 'not' và 'in' nên khi bạn đọc một chương trình, script, hoặc khi đọc to cho người khác nghe mà không cảm thấy giống như bạn đang nói một thứ ngôn ngữ bí mật nào đó. Điều này cũng được hỗ trợ bởi các quy tắc chấm phẩy câu rất nghiêm ngặt của Python, có nghĩa là bạn không có những dấu ngoặc nhọn ({}) trong code của bạn.

Ngoài ra, Python có một tập hợp các quy tắc, được gọi là PEP 8, để hướng dẫn mọi lập trình viên Python làm thế nào để định dạng code của họ. Điều này có nghĩa là bạn luôn biết được nơi để đặt những dòng mới, và quan trọng hơn, đó là mọi script Python mà bạn tham khảo, cho dù nó được viết bởi một "lính mới" hay bởi một chuyên gia dày dạn kinh nghiệm, sẽ nhìn rất giống nhau và rất dễ đọc. Thực tế rằng những dòng code Python do một lập trình viên có nhiều hơn 5 năm kinh nghiệm như chúng ta viết ra, trông cũng rất giống với code do tác giả của ngôn ngữ Python là Guido van Rossum viết ra vậy.

Các thư viện phong phú

Python đã tồn tại khoảng hơn 20 năm, vì vậy có rất nhiều code viết bằng Python được xây dựng qua nhiều thập kỷ, và là một ngôn ngữ mã nguồn mở, rất nhiều trong số này được phát hành cho người khác sử dụng. Hầu như tất cả chúng được tập hợp lại trên trang web https://pypi.python.org, bạn phát âm nó là "pie-pee-eye", hoặc còn được gọi bằng một cái tên phổ biến hơn là "the CheeseShop". Bạn có thể cài đặt phần mềm này lên hệ thống của bạn để sử dụng bởi các dự án của riêng bạn. Ví dụ, nếu bạn muốn sử dụng Python để xây dựng những script với các đối số dòng lệnh, bạn nên cài đặt thư viện "click" và sau đó import nó vào trong các script của bạn rồi sử dụng nó. Có những thư viện sử dụng được cho khá nhiều trường hợp từ thao tác với hình ảnh, cho tới tính toán khoa học, và tự động hóa máy chủ.

Python có một cộng đồng sử dụng lớn

Python có nhiều nhóm người sử dụng ở khắp mọi nơi, thường được gọi là các PUG, và họ tiến hành những cuộc hội thảo lớn trên tất cả mọi châu lục ngoại trừ Nam Cực. PyCon NA, hội nghị về Python lớn nhất ở Bắc Mỹ, đã bán ra 2.500 vé trong năm nay. Hội nghị này phản ánh cam kết đa dạng hóa của Python, vì có trên 30% diễn giả là phụ nữ. Việc trở thành một phần của một cộng đồng tích cực như vậy sẽ luôn tạo ra rất nhiều động lực cho bạn.

4.7.1.2. Thư viện Open CV

OpenCV là một thư viện mã nguồn mở hàng đầu cho thị giác máy tính (computer vision), xử lý ảnh và máy học, và các tính năng tăng tốc GPU trong hoạt động thời gian thực.

OpenCV được phát hành theo giấy phép BSD, do đó nó hoàn toàn miễn phí cho cả học thuật và thương mại. Nó có các interface C++, C, Python, Java và hỗ trợ Windows, Linux, Mac OS, iOS và Android. OpenCV được thiết kế để tính toán hiệu quả và với sự tập trung nhiều vào các ứng dụng thời gian thực. Được viết bằng tối ưu hóa C/C++, thư viện có thể tận dụng lợi thế của xử lý đa lõi. Được sử dụng trên khắp thế giới, OpenCV có cộng đồng hơn 47 nghìn người dùng và số lượng download vượt quá 6 triệu lần. Phạm vi sử dụng từ nghệ thuật tương tác, cho đến lĩnh vực khai thác mỏ, bản đồ trên web hoặc công nghệ robot.

Chức năng của OpenCV: Image/video I/O, xử lý, hiển thị (core, imgproc, highgui) Phát hiện các vật thể (objdetect, features2d, nonfree) Geometry-based monocular or stereo computer vision (calib3d, stitching, videostab) Computational photography (photo, video, superres) Machine learning & clustering (ml, flann) CUDA acceleration (gpu)

4.7.1.3. Thư viện Tensorflow

Tensorflow là một thư viện mã nguồn mở mạnh mẽ cho machine learning được phát triển bởi các chuyển gia của Google Company. Thư viện này có rất nhiều các hàm được xây dựng sẵn cho từng bài toán. Nó cho phép xây dựng nhiều mạng neural network khác nhau. Tensorflow cũng cho phép tính toán song song trên nhiều máy tính khác nhau và cũng có thể trên nhiều CPU, GPU trong cùng một máy. Tensorflow cung cấp các giao diện lập trình ứng dụng API làm việc với Python, C++.

4.7.1.4. Thư viện Keras

Keras là một library được phát triển vào năm 2015 bởi François Chollet, là một kỹ sư nghiên cứu deep learning tại google. Nó là một open source cho neural network được viết bởi ngôn ngữ python. keras là một API bậc cao có thể sử dụng chung với các thư viện deep learning nổi tiếng như tensorflow (được phát triển bởi Google), CNTK (được phát triển bởi microsoft),theano(người phát triển chính Yoshua Bengio). keras có một số ưu điểm như :

- ✓ Dễ sử dụng, xây dựng model nhanh.
- ✓ Có thể chạy trên cả CPU và GPU.
- ✓ Hỗ trợ xây dựng CNN , RNN và có thể kết hợp cả hai.

4.7.2. Ngôn ngữ R

4.7.2.1. Giới thiệu

R là một ngôn ngữ lập trình hàm cấp cao vừa là một môi trường dành cho tính toán thống kê. R hỗ trợ rất nhiều công cụ cho phân tích dữ liệu, khai phá tri thức và dữ liệu nhưng lại là phần mềm miễn phí mã nguồn mở. Hơn nữa R rất dễ học và có thể phát triển nhanh các ứng dụng tính toán xác suất thống kê, phân tích dữ liệu.

Các ưu điểm của R:

- ✓ R là một ngôn ngữ lập trình hoàn thiện định hướng cho tính toán thống kê, phân

tích dữ liệu. R cho phép xây dựng những hàm, những câu lệnh chỉ để giải quyết một nhóm các nhiệm vụ phân tích đặc thù nào đó.

- ✓ Với tư cách là một công cụ phân tích dữ liệu nói chung, R còn là một công cụ cho Data Mining, Big Data, và Machine Learning.

✓ Package của R là một tập hợp các chương trình, hàm được viết sẵn để xử lý một nhóm các phân tích hay một nhóm các bài toán nào đó. Trong nhiều trường hợp, các Packages này có thể bao gồm cả dữ liệu đi kèm. Đến thời điểm năm 2017, đã có gần 1000 Packages được cung cấp bởi cộng đồng phân tích dữ liệu.

4.7.2.2. Gói ggplot2

Ggplot2 là một package hỗ trợ visualization rất mạnh trong R. Dựa trên package này ta có thể vẽ được các đồ thị dạng barchart, line, plot, density, candle chart,pie,… và rất nhiều các đồ thị khác.Ngoài ra ggplot2 còn cho phép người dùng tùy chỉnh màu sắc, kích cỡ, theme, title, … để đồ thị được đẹp hơn. Cấu trúc của ggplot2 được chia rõ ràng làm 2 phần chính.

✓ ggplot(): phần này qui định đồ thị sẽ sử dụng data nào làm đầu vào. Lưu ý data phải có dạng data.frame. Dạng vector sẽ không được support.

✓ geom_(aes(x,y)): Phần này qui định kiểu đồ thị và các trục tọa độ từ dữ liệu đầu vào. Nếu chỉ có ggplot() mà không thêm geom_() thì chúng ta chỉ nhận được background mà không có đồ thị mặc dù data đã được khai báo. Trong geom_() chúng ta phải khai báo thêm trục tọa độ vào các arguments x và y của aes() chẳng hạn như geom_point(aes(x=bienx,y=bieny)).

Các kiểu đồ thị chính: geom_line(): biển diễn line geom_point(): biểu diễn point geom_chart(): biểu diễn chart geom_density(): biểu diễn dưới dạng density geom_vertical(): thêm trục vertical geom_abline(): thêm trục horizontal, vertical và diagonal geom_qq(): quantile and quantile plot geom_contour(): vẽ các đường đồng mức 2d và 3d geom_label() geom_text(): hiển thị text geom_raster(), geom_tile(), geom_rect(): biểu diễn dạng màu sắc mật độ.

4.7.2.3. Gói readr

Package này cung cấp các hàm với chức năng đọc dữ liệu như:

✓ read_csv đọc tệp tin dữ liệu có các trường phân cách bằng dấu phẩy ,

✓ read_csv2 đọc tệp tin dữ liệu có các trường phân cách bằng dấu chấm phẩy ;

✓ read_tsv đọc tệp tin dữ liệu có các trường phân cách bằng kí tự tab \t

✓ read_delim đọc tệp tin dữ liệu có các trường phân cách bằng kí tự khác

✓ read_table đọc tệp tin dữ liệu có kích thước của trường được quy định trước

CÂU HỎI, BÀI TẬP

1. Sử dụng kỹ thuật học máy để phân loại ký tự viết tay
2. Sử dụng kỹ thuật học máy để phân loại biển báo giao thông
3. Mô hình LSTM tốt hơn mô hình RNN ở điểm nào?
4. Sử dụng mô hình LSTM cho bài toán dự báo một số chỉ số tài chính (chứng khoán, giá vàng, dầu, kim loại quý,..)
5. Dùng mô hình LSTM cho bài toán dự đoán bitcoin.