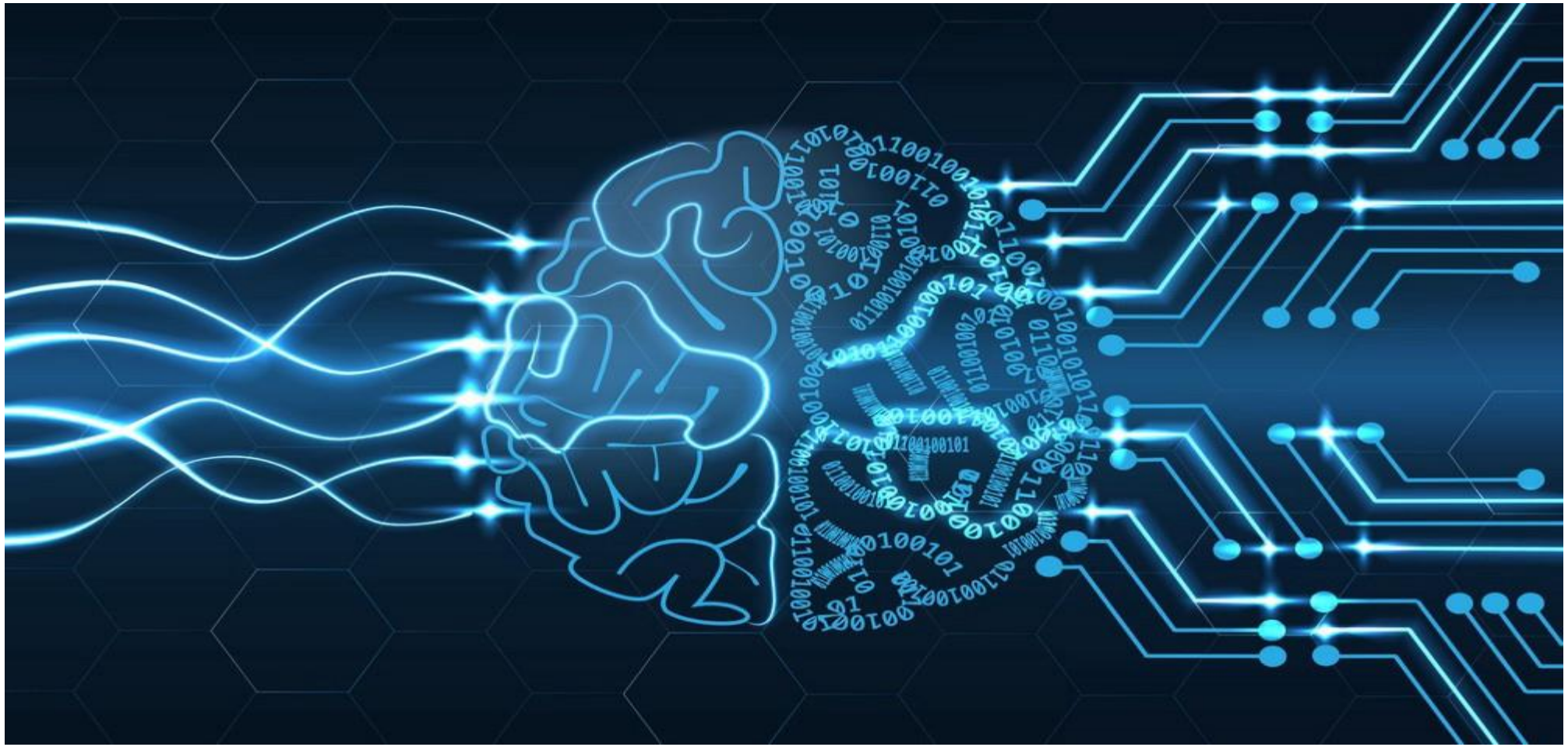
WEEK 8

Machine Learning (cont)

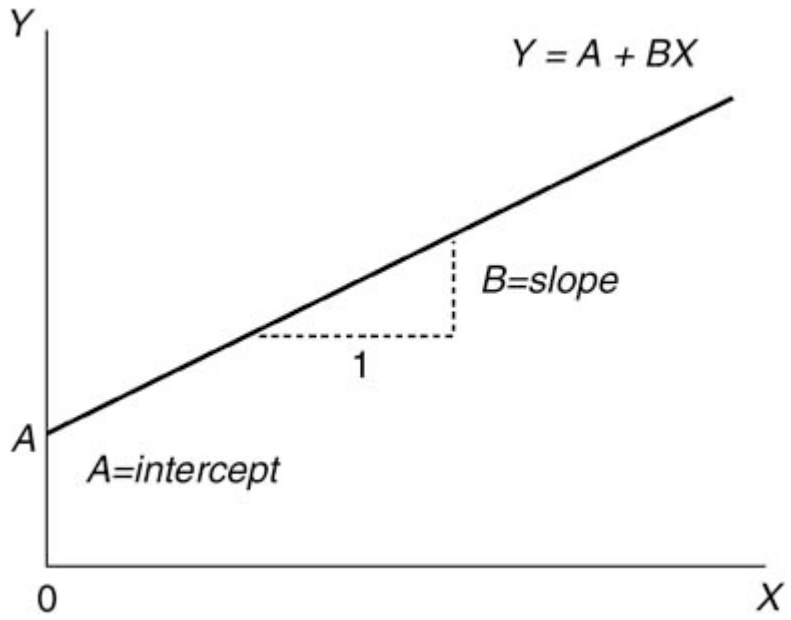


Linear Regression

Giả sử trong tập dữ liệu (dataset) của chúng ta có một biến độc lập (independent variable) là x và một biến phụ thuộc (dependent variable) là y. Từ đó chúng ta sẽ thiết lập mối liên hệ giữa chúng bằng một hàm số tuyến tính (linear function) như sau:

$$y = w_0 + w_1 x$$

Trong hàm số trên chúng ta có thể để ý thấy có 2 trọng số (weight) đó là w0 và w1. Trong 2 trọng số này, w0 chính là hệ số chặn (intercept), w1 là hệ số nghiêng (slope) của đường tuyến tính (linear)

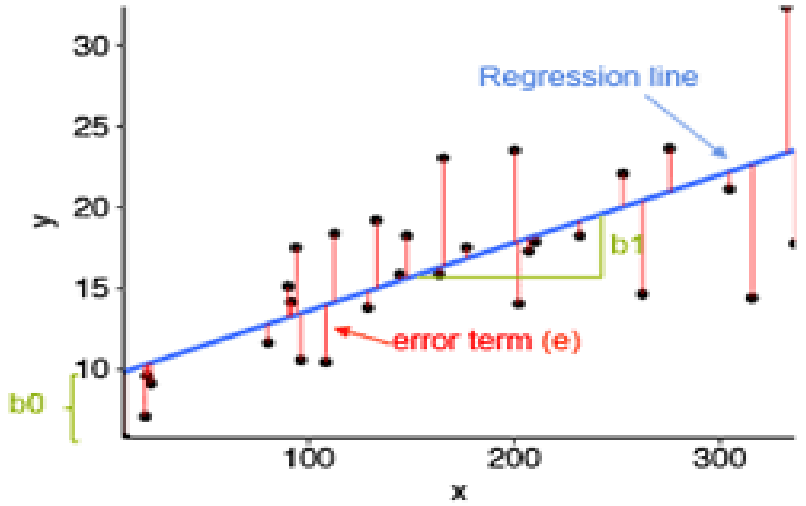


Linear Regression

Linear Regression model có nhiệm vụ tìm ra w0 và w1 phù hợp (fit) với tập dữ liệu (dataset). Hàm mất mát được sử dụng trong LR model đó là Residual Sum of Square (RSS), và được tính theo công thức sau:

$$RSS = \sum_1^n (y_i actual - y_i predict)^2$$

Trong công thức trên n chính là số cá thể (instance) trong tập dữ liệu (dataset). Phần dư thừa (residual) giữa kết quả thực tế và dự đoán này chính là sự sai lệch giữa giá trị thực tế và giá trị dự đoán. Để có thể nhìn thấy một cách trực quan hơn chúng ta hãy xem hình vẽ dưới đây:

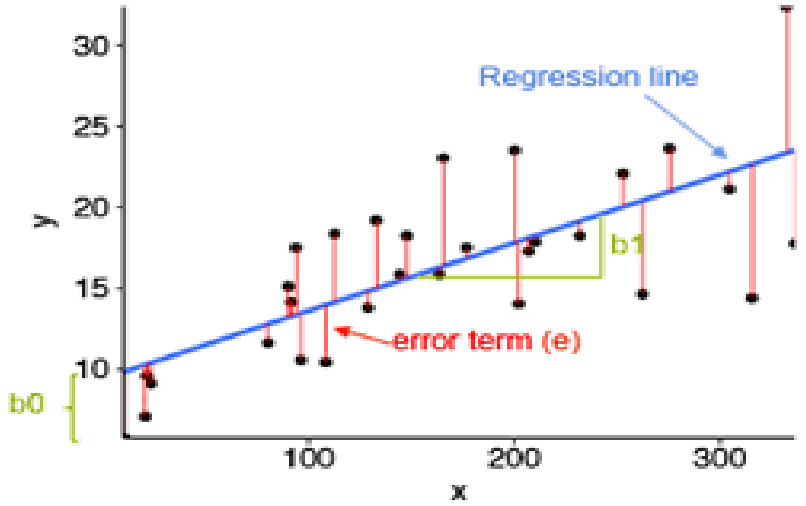


Linear Regression

Linear Regression model có nhiệm vụ tìm ra w0 và w1 phù hợp (fit) với tập dữ liệu (dataset). Hàm mất mát được sử dụng trong LR model đó là Residual Sum of Square (RSS), và được tính theo công thức sau:

$$RSS = \sum_1^n (y_i actual - y_i predict)^2$$

Trong công thức trên n chính là số cá thể (instance) trong tập dữ liệu (dataset). Phần dư thừa (residual) giữa kết quả thực tế và dự đoán này chính là sự sai lệch giữa giá trị thực tế và giá trị dự đoán. Để có thể nhìn thấy một cách trực quan hơn chúng ta hãy xem hình vẽ dưới đây:

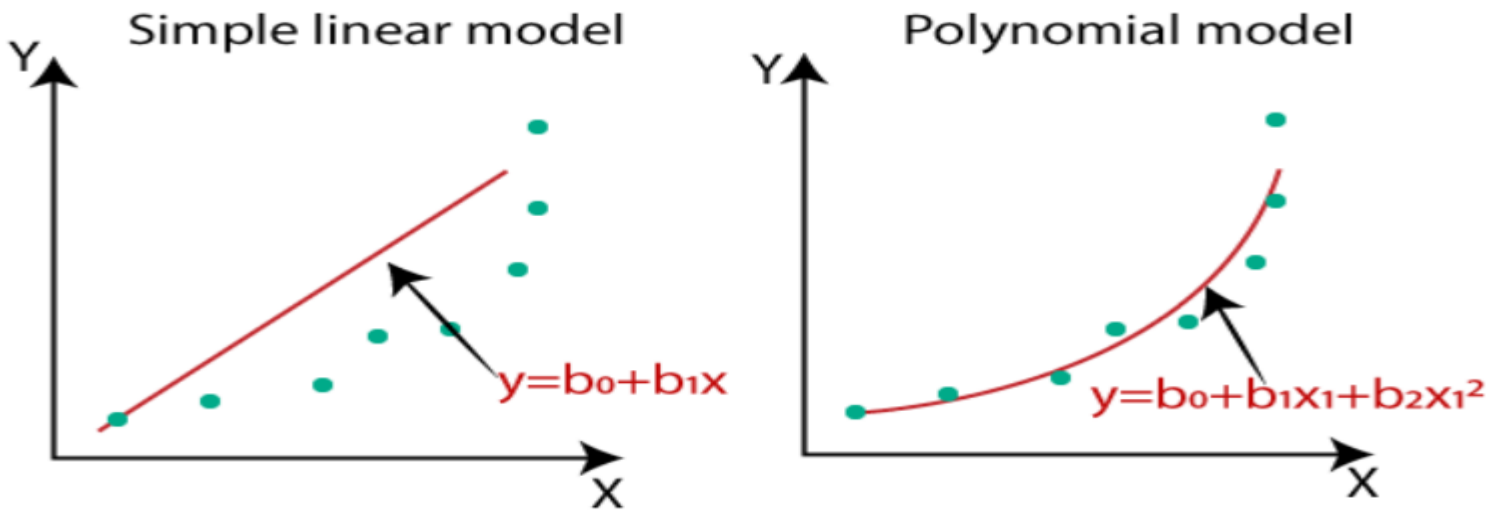


Hồi quy phi tuyến tính - Polynomial Linear Regression

Khi sự phân bố của các cá thể trong tập dữ liệu mang tính phi tuyến tính (nonlinear) chúng ta có thể sử dụng phương pháp biến đổi với các biến không phụ thuộc (independent variable) để chuyển Linear Regression sang Polynomial Regression nhằm giúp cải thiện hiệu năng (performance) của model. Hàm số đa thức (polynomial function) độ (degree) 2 có dạng như sau:

$$y = w_0 + w_1 x + w_2 x^2$$

Việc tạo thêm một tính năng (feature) x^2, vô hình chúng giúp tạo ra sự một phi tuyến tính (non linear) cho model. Dù vậy về bản chất x^2 được tạo ra là từ x nên các trọng số gắn với nó là w1 và w2 vẫn có mối quan hệ tuyến tính (linear relationship). Vì lý do đó Polynomial Regression được coi như một một trường hợp đặc biệt của Linear Regression.



Gradient Descent

Trong Machine Learning nói riêng và Toán Tối Ưu nói chung, chúng ta thường xuyên phải tìm giá trị nhỏ nhất (hoặc đôi khi là lớn nhất) của một hàm số nào đó

Nhìn chung, việc tìm global minimum của các hàm mất mát trong Machine Learning là rất phức tạp, thậm chí là bất khả thi. Thay vào đó, người ta thường cố gắng tìm các điểm local minimum, và ở một mức độ nào đó, coi đó là nghiệm cần tìm của bài toán.

Các điểm local minimum là nghiệm của phương trình đạo hàm bằng 0. Nếu bằng một cách nào đó có thể tìm được toàn bộ (hữu hạn) các điểm cực tiểu, ta chỉ cần thay từng điểm local minimum đó vào hàm số rồi tìm điểm làm cho hàm có giá trị nhỏ nhất.Tuy nhiên, trong hầu hết các trường hợp, việc giải phương trình đạo hàm bằng 0 là bất khả thi.

Hướng tiếp cận phổ biến nhất là xuất phát từ một điểm mà chúng ta coi là *gần* với nghiệm của bài toán, sau đó dùng một phép toán lặp để *tiến dần* đến điểm cần tìm, tức đến khi đạo hàm gần với 0. Gradient Descent (viết gọn là GD) và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất.

Gradient Descent cho hàm 1 biến

1. Nếu đạo hàm của hàm số tại x_t: $f'(x_t) > 0$ thì x_t nằm về bên phải so với x^* (và ngược lại). Để điểm tiếp theo x_{t+1} gần với x^* hơn, chúng ta cần di chuyển x_t về phía bên trái, tức về phía *âm*. Nói các khác, **chúng ta cần di chuyển ngược dấu với đạo hàm**:

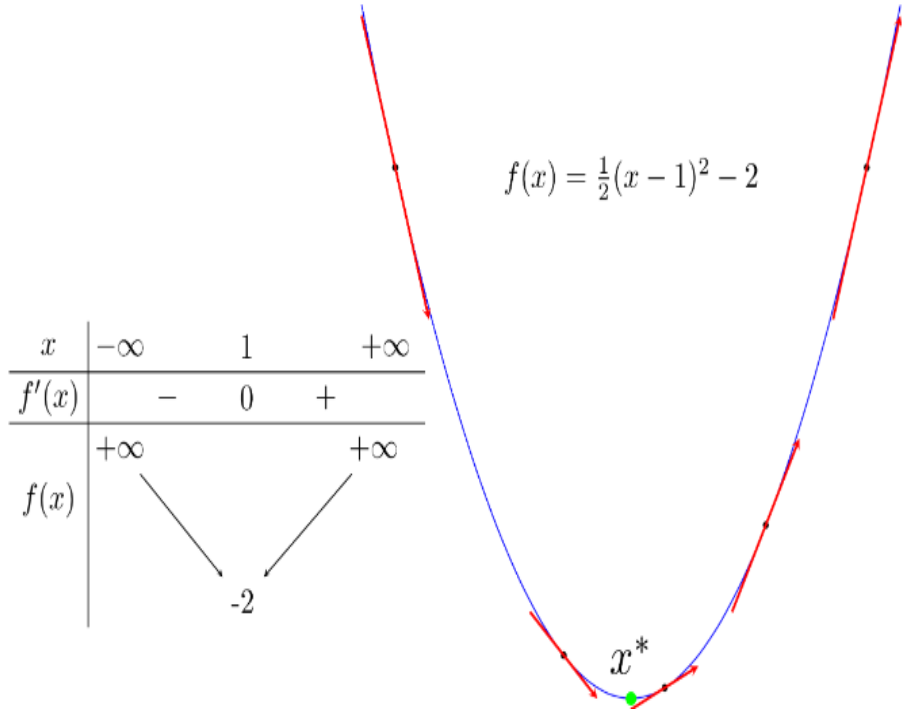
$$x_{t+1} = x_t + \Delta$$

Trong đó Δ là một đại lượng ngược dấu với đạo hàm $f'(x_t)$.

2. x_t càng xa x^* về phía bên phải thì $f'(x_t)$ càng lớn hơn 0 (và ngược lại). Vậy, lượng di chuyển Δ, một cách trực quan nhất, là tỉ lệ thuận với $-f'(x_t)$.

Hai nhận xét phía trên cho chúng ta một cách cập nhật đơn giản là:

$$x_{t+1} = x_t - \eta f'(x_t)$$



Gradient Descent: Phân loại

1. Batch Gradient Descent

Batch ở đây được hiểu là *tất cả*, tức khi cập nhật θ=wθ=w, chúng ta sử dụng **tất cả** các điểm dữ liệu xi

Cách làm này có một vài hạn chế đối với cơ sở dữ liệu có vô cùng nhiều điểm (hơn 1 tỉ người dùng của facebook chẳng hạn). Việc phải tính toán lại đạo hàm với tất cả các điểm này sau mỗi vòng lặp trở nên cồng kềnh và không hiệu quả. Thêm nữa, thuật toán này được coi là không hiệu quả với *online learning*.

Online learning là khi cơ sở dữ liệu được cập nhật liên tục (thêm người dùng đăng ký hàng ngày chẳng hạn), mỗi lần thêm vài điểm dữ liệu mới. Kéo theo đó là mô hình của chúng ta cũng phải thay đổi một chút để phù hợp với các dữ liệu mới này. Nếu làm theo Batch Gradient Descent, tức tính lại đạo hàm của hàm mất mát tại tất cả các điểm dữ liệu, thì thời gian tính toán sẽ rất lâu, và thuật toán của chúng ta coi như không *online* nữa do mất quá nhiều thời gian tính toán.

Trên thực tế, có một thuật toán đơn giản hơn và tỏ ra rất hiệu quả, có tên gọi là Stochastic Gradient Descent (SGD).

Gradient Descent: Phân loại

2. Stochastic Gradient Descent.

Trong thuật toán này, tại 1 thời điểm, ta chỉ tính đạo hàm của hàm mất mát dựa trên *chỉ một* điểm dữ liệu xi rồi cập nhật θ dựa trên đạo hàm này. Việc này được thực hiện với từng điểm trên toàn bộ dữ liệu, sau đó lặp lại quá trình trên. Thuật toán rất đơn giản này trên thực tế lại làm việc rất hiệu quả.

Mỗi lần duyệt một lượt qua *tất cả* các điểm trên toàn bộ dữ liệu được gọi là một epoch. Với GD thông thường thì mỗi epoch ứng với 1 lần cập nhật θ, với SGD thì mỗi epoch ứng với N lần cập nhật θ với N là số điểm dữ liệu. Nhìn vào một mặt, việc cập nhật từng điểm một như thế này có thể làm giảm đi tốc độ thực hiện 1 epoch. Nhưng nhìn vào một mặt khác, SGD chỉ yêu cầu một lượng epoch rất nhỏ (thường là 10 cho lần đầu tiên, sau đó khi có dữ liệu mới thì chỉ cần chạy dưới một epoch là đã có nghiệm tốt). Vì vậy SGD phù hợp với các bài toán có lượng cơ sở dữ liệu lớn (chủ yếu là Deep Learning mà chúng ta sẽ thấy trong phần sau của blog) và các bài toán yêu cầu mô hình thay đổi liên tục, tức online learning

Gradient Descent: Phân loại

3. Mini-batch Gradient Descent

Khác với SGD, mini-batch sử dụng một số lượng n lớn hơn 1 (nhưng vẫn nhỏ hơn tổng số dữ liệu Nrất nhiều). Giống với SGD, Mini-batch Gradient Descent bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu rồi chia toàn bộ dữ liệu thành các *mini-batch*, mỗi *mini-batch* có n điểm dữ liệu (trừ mini-batch cuối có thể có ít hơn nếu N không chia hết cho n). Mỗi lần cập nhật, thuật toán này lấy ra một mini-batch để tính toán đạo hàm rồi cập nhật. Công thức có thể viết dưới dạng:

$$\theta = \theta - \eta \nabla_\theta J(\theta; \mathbf{x}_{i:i+n}; \mathbf{y}_{i:i+n})$$

Với $\mathbf{x}_{i:i+n}$ được hiểu là dữ liệu từ thứ i tới thứ $i+n-1$ (theo ký hiệu của Python). Dữ liệu này sau mỗi epoch là khác nhau vì chúng cần được xáo trộn. Một lần nữa, các thuật toán khác cho GD như Momentum, Adagrad, Adadelta,… cũng có thể được áp dụng vào đây.

Mini-batch GD được sử dụng trong hầu hết các thuật toán Machine Learning, đặc biệt là trong Deep Learning. Giá trị n thường được chọn là khoảng từ 50 đến 100.

Giới thiệu về SVM

SVM là viết tắt của cụm từ *support vector machine*. Đây là một thuật toán khá hiệu quả trong lớp các bài toán phân loại nhị phân và dự báo của học có giám sát. Thuật toán này có ưu điểm là hoạt động tốt đối với những mẫu dữ liệu có kích thước lớn và thường mang lại kết quả vượt trội so với lớp các thuật toán khác trong học có giám sát.

Ưu điểm của SVM đó là:

- Đây là thuật toán hoạt động hiệu quả với không gian cao chiều (*high dimensional spaces*).
- Thuật toán tiêu tốn ít bộ nhớ vì chỉ sử dụng các điểm trong *tập hỗ trợ* để dự báo trong *hàm quyết định*.
- Chúng ta có thể tạo ra nhiều *hàm quyết định* từ những hàm kernel khác nhau. Thậm chí sử dụng đúng kernel có thể giúp cải thiện thuật toán lên đáng kể.

Giới thiệu về SVM

Chính vì tính hiệu quả mà SVM thường được áp dụng nhiều trong các tác vụ phân loại và dự báo, cũng như được nhiều công ty ứng dụng và triển khai trên môi trường production. *Chúng ta có thể liệt kê một số ứng dụng của thuật toán SVM đó là:*

•Mô hình chuẩn đoán bệnh. Dựa vào biến mục tiêu là những chỉ số xét nghiệm lâm sàng, thuật toán đưa ra dự báo về một số bệnh như tiểu đường, suy thận, máu nhiễm mỡ,…

•Trước khi thuật toán CNN và Deep Learning bùng nổ thì SVM là lớp mô hình cực kì phổ biến trong phân loại ảnh.

•Mô hình phân loại tin tức. Xác định chủ đề của một đoạn văn bản, phân loại cảm xúc văn bản, phân loại thư rác.

•Mô hình phát hiện gian lận.

7.1. Hàm mất mát của SVM

7.1.1. Góc nhìn từ hồi qui Logistic

Trong hồi qui Logistic chúng ta đã làm quen với *hàm mất mát* (*loss function*) dạng:

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} -[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Bản chất của hàm mất mát trong hồi qui Logistic là một *thước đo* về sự tương quan giữa phân phối xác suất dự báo với *ground truth*.

Trong đó phân phối xác suất được ước tính dựa trên hàm `Sigmoid` theo công thức $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$.

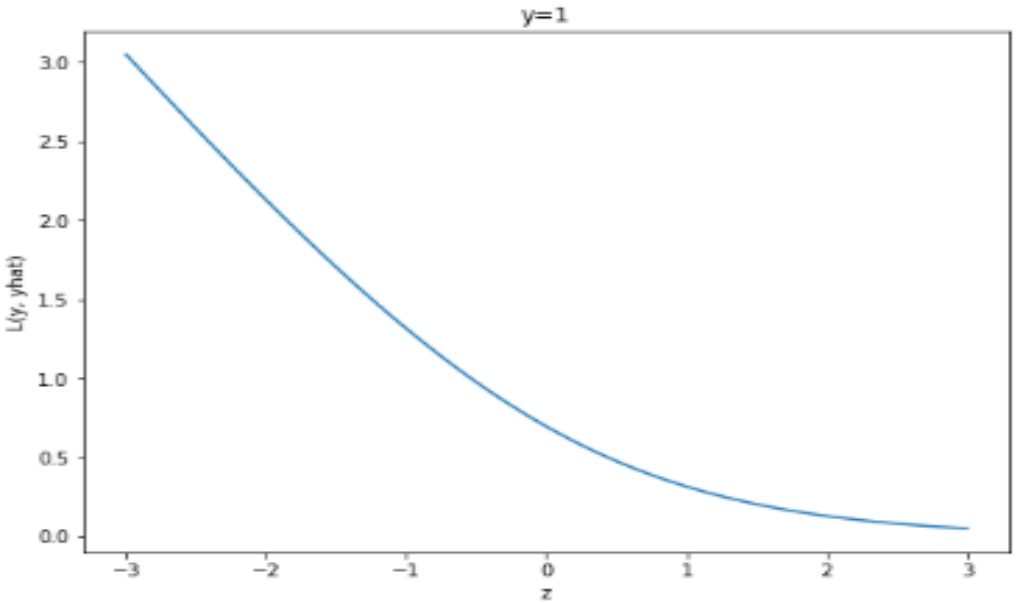
Ta cũng biết rằng đường biên phân loại của hồi qui Logistic là một siêu phẳng có phương trình $\mathbf{w}^\intercal \mathbf{x}$.

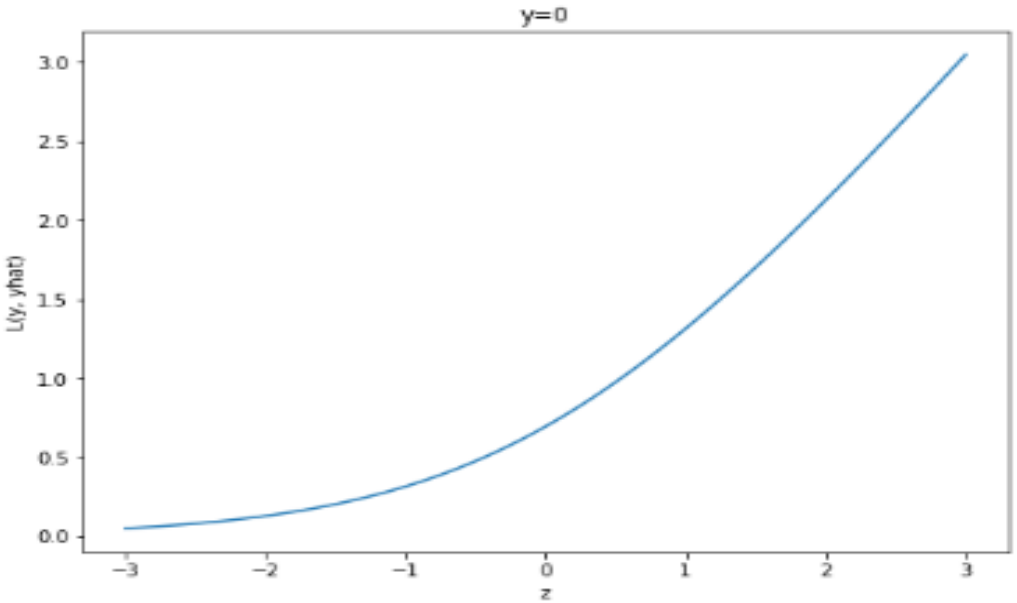
$$\hat{y} = \begin{cases} 1 \text{ if } \mathbf{w}^\intercal \mathbf{x} > 0 \\ 0 \text{ if } \mathbf{w}^\intercal \mathbf{x} \leq 0 \end{cases}$$

Tiếp theo chúng ta sẽ cùng phân tích *hàm mất mát* của mô hình trong hai trường hợp $y = 0$ và $y = 1$:

$$\mathcal{L}(\mathbf{w}) = \begin{cases} -\log(\hat{y}_i) & \text{if } y_i = 1 \\ -\log(1 - \hat{y}_i) & \text{if } y_i = 0 \end{cases}$$

Giới thiệu về SVM





Ta nhận thấy hình dạng của *hàm mất mát* trong hai trường hợp tương ứng với $y = 1$ và $y = 0$ là trái ngược nhau:

- Đối với trường hợp nhãn $y = 1$: Khi giá trị của z càng lớn thì hàm mất mát sẽ tiệm cận 0. Điều đó đồng nghĩa với mô hình sẽ phạt ít những trường hợp z lớn và có nhãn 0. Những trường hợp này tương ứng với những điểm nằm cách xa đường biên phân chia.
- Đối với nhãn $y = 0$ thì trái lại, mô hình có xu hướng phạt ít với những giá trị z nhỏ. Khi đó những điểm này sẽ nằm cách xa đường biên về phía nửa mặt phẳng $y = 1$.