MACHINE LEARNING

Cao Văn Chung
cvanchung@hus.edu.vn
Informatics Dept., MIM, HUS, VNU Hanoi

Support Vector Machine
(Part 3: Kernel SVM for classification with nonlinear boundaries)

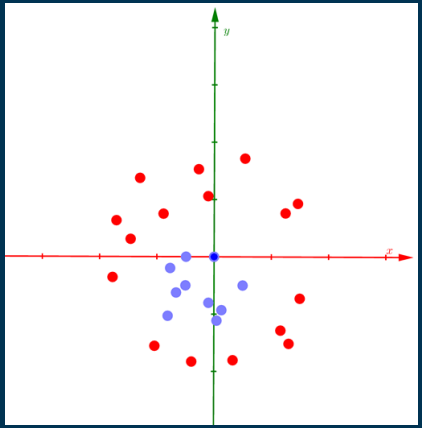
SVM vs. Otherwise

Các mô hình phân loại đã học:
Có sự tương đồng giữa nhóm phương pháp dạng SVM và hệ thống các phương pháp còn lại.

Neural Networks	Support Vector Machine	Đặc điểm chung
PLA	Hard Margin SVM	Binary classification Classes là tách được tuyến tính
Logistic Regression	Soft Margin SVM	Binary classification Dữ liệu gần như tách được tuyến tính
Softmax Regression	Multi-class SVM	Multi-classes (biên là tuyến tính)
Multi-layer Perceptron	Kernel SVM	Binary classification Các classes không tách tuyến tính

Non (Nearly) Linearly Separable Data

- Trường hợp dữ liệu tách được tuyến tính & gần như tách được tuyến tính, ta có Hard Margin & Soft Margin SVM.
- Xét trường hợp dữ liệu thực sự không tách được tuyến tính như trong hình $\Leftrightarrow$ không sử đụng được Hard & Soft Margin SVM

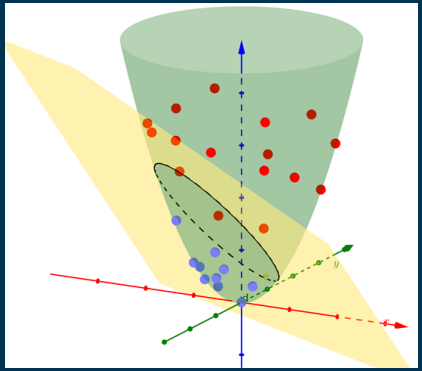


Non (Nearly) Linearly Separable Data

- Trick: Nếu bổ sung chiều thứ ba là hàm số của hai chiều đã có

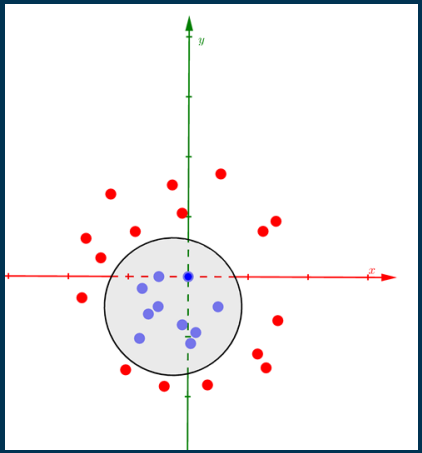
$$z = x^2 + y^2$$

- Dữ liệu mới được phân bố trong không gian 3D: Những điểm xa tâm sẽ có giá trị z lớn hơn => Các lớp dữ liệu là tách được tuyến tính trong chiều thứ ba.



Non (Nearly) Linearly Separable Data

- Các điểm dữ liệu 3D sẽ được phân bố trên một parabolic trong không gian 3D và đã trở nên phân biệt tuyến tính.
- Thực tế dữ liệu gốc (2D) là tách được nhưng đường biên là phi tuyến.



Kernel

- **Tổng quát:**
  - Kernel SVM $\Leftrightarrow$ Tìm hàm $\Phi(x)$ biến đổi dữ liệu X từ không gian *feature* ban đầu thành dữ liệu trong không gian mới.
  - Trong ví dụ, $\Phi(\cdot)$ bổ sung một chiều dữ liệu mới (hay một *feature* mới), là hàm số của các *feature* đã biết.
- **Mục đích của $\Phi(\cdot)$**
  - $\Phi(\cdot$ cần thỏa mãn: Trong không gian mới, dữ liệu của các classes là tách được tuyến tính hoặc gần như tách được tuyến tính.
  - Khi đó, có thể dùng các bộ phân lớp tuyến tính thông thường như PLA, Logistic Regression, hay Hard/Soft Margin SVM.

Kernel

- **So sánh tương đối:**
  - Hàm Kernel $\Phi(\cdot)$ tương đồng với hàm *ACTIVATION* trong nhóm phương pháp dạng *NEURAL NETWORKS*.
  - Cả hai hàm đều có mục đích chuyển đổi giữa bài toán phân loại phi tuyến và bài toán phân loại tuyến tính.
- **Điểm khác biệt**
  - Nhiệm vụ của activation function là phá vỡ tính tuyến tính của mô hình.
  - Trong khi đó, hàm $\Phi(\cdot)$ biến dữ liệu không phân biệt tuyến tính thành phân biệt tuyến tính.

Kernel

- $\Phi(\cdot)$ **thường tạo ra dữ liệu mới có số chiều cao hơn số chiều của dữ liệu ban đầu.**
  - Nếu tính toán trực tiếp cho từng điểm dữ liệu, sẽ dẫn tới các vấn đề về bộ nhớ và hiệu năng tính toán.
  - Cách tiếp cận: sử dụng các kernel functions mô tả quan hệ giữa hai điểm dữ liệu bất kỳ trong không gian mới.
  - Cụ thể: Thiết lập lại công thức của các bài toán tối ưu trong không gian mới sau khi đã sử dụng Kernel function.
  - Kỹ thuật này được xây dựng dựa trên quan sát về bài toán đối ngẫu của SVM.

Kernel

Nhắc lại bài toán đối ngẫu trong Soft Margin SVM cho dữ liệu hầu như tách được tuyến tính (*nearly linearly separable*):

$$\lambda = \arg\max_{\lambda} \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \mathbf{x}_n^T \mathbf{x}_m \tag{1}$$

$$\text{subject to: } \sum_{n=1}^{N} \lambda_n y_n = 0; \qquad 0 \leq \lambda_n \leq C, \ \forall n = 1, 2, \ldots, N$$

Kernel

Trong đó:

- N : số cặp điểm dữ liệu trong tập training.
- x_n : feature vector của dữ liệu thứ n trong tập training.
- y_n : nhãn của dữ liệu thứ n, bằng 1 hoặc -1.
- λ_n : nhân tử Lagrange ứng với điểm dữ liệu thứ n.
- $C > 0$: hằng số cân đối độ lớn của margin và các điểm phải chấp nhận nằm trong vùng không an toàn. Khi $C = \infty$ (thực tế chọn số rất lớn), Soft Margin SVM trở thành Hard Margin SVM.

Kernel

Sau khi giải được λ từ bài toán (1), với một điểm dữ liệu mới x, ta xác định nhãn y ứng với x bởi biểu thức:

$$y := \text{sign}\left\{\sum_{m\in\mathcal{S}} \lambda_m y_m \mathbf{x}_m^T \mathbf{x} + \frac{1}{N_\mathcal{M}} \sum_{n\in\mathcal{M}} \left(y_n - \sum_{m\in\mathcal{S}} \lambda_m y_m \mathbf{x}_m^T \mathbf{x}_n\right)\right\} \qquad (2)$$

Trong đó

- $\mathcal{M} = \{n : 0 < \lambda_n < C\}$: là tập hợp những điểm nằm trên margin.
- $\mathcal{S} = \{n : 0 < \lambda_n\}$: là tập hợp các điểm support.
- $N_\mathcal{M}$: là số phần tử của $\mathcal{M}$.

Kernel

- Thực tế, các phân lớp dữ liệu có thể có đường biên phi tuyến, tức là không hầu như tách được tuyến tính (*non - nearly linearly separable*).
- Trong trường hợp trên, nghiệm của (1) không phản ánh đúng phân lớp dữ liệu.
- Từ ý tưởng của ví dụ trên, ta tìm $\Phi(\cdot) : x \mapsto \Phi(x)$ - điểm dữ liệu trong không gian mới. Trong không gian này, các lớp dữ liệu (sau biến đổi) là hầu như tách được tuyến tính.

Kernel

Bài toán đối ngẫu (1) trong không gian mới (sau biến đổi bởi Φ) trở thành:

$$\lambda = \arg\max_{\lambda} \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \Phi(\mathbf{x}_n)^T \Phi(\mathbf{x}_m) \tag{3}$$

$$\text{subject to: } \sum_{n=1}^{N} \lambda_n y_n = 0; \qquad 0 \leq \lambda_n \leq C, \ \forall n = 1, 2, \ldots, N$$

Kernel

Và nhãn y của mẫu dữ liệu mới x sẽ được xác định bởi

$$
\begin{aligned}
y &:= \text{sign}\left\{\mathbf{w}^T\Phi(\mathbf{x}) + b\right\} \\
&= \left\{\sum_{m\in\mathcal{S}} \lambda_m y_m \Phi(\mathbf{x}_m)^T\Phi(\mathbf{x}) + \frac{1}{N_\mathcal{M}}\sum_{n\in\mathcal{M}}\left(y_n - \sum_{m\in\mathcal{S}}\lambda_m y_m \Phi(\mathbf{x}_m)^T\Phi(\mathbf{x}_n)\right)\right\}
\end{aligned} \tag{4}
$$

- $\mathcal{M} = \{n : 0 < \lambda_n < C\}$: là tập hợp những điểm nằm trên margin.
- $\mathcal{S} = \{n : 0 < \lambda_n\}$: là tập hợp các điểm support.
- $N_\mathcal{M}$: là số phần tử của $\mathcal{M}$.

Kernel

Nhận xét:

- Số chiều của $\Phi(x)$ thường là rất lớn và nếu tính các $\Phi(x)$ dẫn tới cần tính tiếp tích vô hướng $\Phi(\mathbf{x}_m)^T\Phi(\mathbf{x}_n)$; $\Phi(\mathbf{x}_m)^T\Phi(\mathbf{x})$.
- Trong bài toán (3) và biểu thức (4), chúng ta không cần tính trực tiếp $\Phi(\mathbf{x})$ cho mọi điểm dữ liệu.
- Ta chỉ thực sự cần tính $\Phi(\mathbf{x})^T\Phi(\mathbf{z})$ cho mọi cặp dữ liệu **xz** bất kỳ.

Kernel trick: *Những phương pháp dựa trên kỹ thuật này, tức thay vì trực tiếp tính tọa độ của một điểm trong không gian mới, ta đi tính tích vô hướng giữa hai điểm trong không gian mới.*

Kernel

Định nghĩa hàm *Kernel*: $k(\mathbf{x}, \mathbf{z}) = \Phi(\mathbf{x})^T \Phi(\mathbf{z})$, lúc đó bài toán (3) trở thành

$$\begin{aligned} \lambda &= \arg\max_{\lambda} \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m k(\mathbf{x}_n, \mathbf{x}_m) \\ \text{subject to: } & \sum_{n=1}^{N} \lambda_n y_n = 0; \qquad 0 \leq \lambda_n \leq C, \ \forall n = 1, 2, \ldots, N \end{aligned} \tag{5}$$

và biểu thức (4)có thể viết lại dưới dạng

$$y = \text{sign} \left\{ \sum_{m \in \mathcal{S}} \lambda_m y_m k(\mathbf{x}_m, \mathbf{x}) + \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} \left(y_n - \sum_{m \in \mathcal{S}} \lambda_m y_m k(\mathbf{x}_m, \mathbf{x}_n) \right) \right\} \tag{6}$$

Kernel

Ví dụ: xét dữ liệu trong không gian 02 chiều, điểm dữ liệu $\mathbf{x} = (x_1, x_2)$, với $\Phi(\cdot) : \mathbb{R}^2 \mapsto \mathbb{R}^3$, $\Phi(\mathbf{x}) = \Phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$ - thuộc không gian 03 chiều. Hàm *Kernel*:

$$\begin{aligned} k(\mathbf{x}, \mathbf{z}) &= \Phi(\mathbf{x})^T \Phi(\mathbf{z}) = (x_1, x_2, x_1^2 + x_2^2)(z_1, z_2, z_1^2 + z_2^2)^T \\ &= x_1 z_1 + x_2 z_2 + (x_1^2 + x_2^2)(z_1^2 + z_2^2). \end{aligned}$$

Dễ thấy tính toán bằng công thức này sẽ nhanh hơn so với tính $\Phi(\mathbf{x})$, $\Phi(\mathbf{z})$ và sau đó mới tính $\Phi(\mathbf{x})^T \Phi(\mathbf{z})$.

Kernel

Tính chất của hàm Kernel:

- Đối xứng: $k(\mathbf{x}, \mathbf{z}) = k(\mathbf{z}, \mathbf{x})$. Điều này suy ra từ tích vô hướng $\Phi(\mathbf{x})^T \Phi(\mathbf{z})$ là đối xứng.
- Về *lý thuyết*, hàm kerrnel cần thỏa mãn điều kiện Mercer.

$$\sum_{n=1}^{N} \sum_{m=1}^{N} k(\mathbf{x}_m, \mathbf{x}_n) c_n c_m \geq 0, \quad \forall c_i \in \mathbb{R}, i = 1, 2, \ldots, N \tag{7}$$

- Tuy nhiên trong thực tế, một số hàm $k(\mathbf{x}, \mathbf{z})$ không thỏa mãn điều kiện Mercer nhưng vẫn cho kết quả chấp nhận được và vẫn được sử dụng.

Kernel

Tính chất của hàm Kernel:

- Với hàm kernel $k(\mathbf{x}, \mathbf{z})$ đối xứng và thỏa mãn điều kiện (7), đặt

$$\mathcal{K} \in \mathbb{R}^{N \times N}: \quad \mathcal{K} = \left(k_{m,n}\right)_{m,n=1}^{N}, \quad \text{trong đó} \quad k_{m,n} := k(\mathbf{x}_m, \mathbf{x}_n).$$

- Lúc đó với mọi $c \in \mathbb{R}^N$ ta có

$$c^T \mathcal{K} c = \sum_{n=1}^{N} \sum_{m=1}^{N} k(\mathbf{x}_m, \mathbf{x}_n) c_n c_m \geq 0.$$

- Vậy bản chất điều kiện Mercer là để đảm bảo ánh xạ $\mathcal{K}$ là đối xứng nửa xác định dương.

Kernel

Giải bài toán tối ưu với hàm Kernel:

- Quay lại bài toán (5) với hàm kernel $k(\mathbf{x}, \mathbf{z})$ đối xứng, thỏa mãn (7), đặt $c_n = y_n\lambda_n$, ta thu được với mọi $\lambda \in \mathbb{R}^N$:

$$\lambda^T\mathbf{K}\lambda = \sum_{n=1}^{N}\sum_{m=1}^{N} k(\mathbf{x}_m, \mathbf{x}_n)y_n y_m\lambda_n\lambda_m = \sum_{n=1}^{N}\sum_{m=1}^{N} k(\mathbf{x}_m, \mathbf{x}_n)c_n c_m \geq 0 \quad (8)$$

- Ở đây $\mathbf{K}$ là một ma trận đối xứng với các phần tử $k_{nm} = y_n y_m k(\mathbf{x}_n, \mathbf{x}_m)$.
- Từ (8) suy ra ma trận $\mathbf{K}$ đối xứng nửa xác định dương.
- Vậy, bài toán tối ưu (5) có ràng buộc lồi & hàm mục tiêu lồi (*quadratic form*).
- Do đó nó có thể được giải tương tự như trong phương pháp SoftMargin SVM.

Kernel

Giải bài toán tối ưu với hàm Kernel:

- Bài toán (5) có dạng quadratic nên sẽ có nghiệm toàn cục, và có thể giải bằng một số thư viện, ví dụ CVXOPT. Sau khi tìm được các λ_n, với mỗi mẫu dữ liệu $\mathbf{x}$ mới ta dự đoán nhãn (phân lớp) của nó theo (6).
- Nhắc lại việc sử dụng CVXOPT solver trong việc giải bài toán quadratic programming. Bài toán dạng:

$$\begin{aligned} \min_{x} \left\{ \frac{1}{2} x^T P x + q x \right\} \\ \text{subject to} \quad Gx \preceq h \\ Ax = b. \end{aligned}$$

Kernel

Giải bài toán tối ưu với hàm Kernel:

So sánh với (5) ta thấy các toán tử trong bài toán QP được xác định bởi

- $P = \mathbf{K} = (k_{nm}) : k_{nm} = y_n y_m k(\mathbf{x}_n, \mathbf{x}_m)$ là ma trận đã được nêu ở trên.
- $q = -\mathbf{l}$ (ma trận đơn vị).
- $h = (0, \ldots, 0, C, \ldots, C)^T$ - vector $2N$ phần tử và

$$G = \begin{pmatrix} -\mathbf{l} & 0 \\ 0 & \mathbf{l} \end{pmatrix}$$

- $A = (\mathbf{y})^T$ và $b = 0$.

Sau khi giải được λ từ (5), ta xác định nhãn của mẫu dữ liệu mới $\mathbf{x}$ theo (6).

Examples of SVM Kernels: Linear

Hàm Kernel tuyến tính:

- Trường hợp đơn giản với kernel chính là tích vô hướng của hai vector:

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}$$

- Có thể chứng minh hàm này thỏa mãn điều kiện Mercer.
- Khi sử dụng hàm sklearn.svm.SVC, kernel này được chọn bằng cách đặt kernel = 'linear'

Examples of SVM Kernels: Polynomial

Hàm Kernel dạng đa thức:

- Công thức tổng quát của kernel:

$$k(\mathbf{x}, \mathbf{z}) = (r + \gamma \mathbf{x}^T \mathbf{z})^d$$

- d là bậc của đa thức; r là hệ số tự do; $\gamma > 0$.
- Khi sử dụng thư viện sklearn, kerrnel này được chọn bằng cách đặt kernel = 'poly'

Examples of SVM Kernels: RBF

Radial Basic Function (RBF) kernel hay Gaussian kernel :

- Đây là dạng được sử dụng nhiều nhất trong thực tế. Công thức tổng quát của kernel:

$$k(\mathbf{x}, \mathbf{z}) = \exp(-\gamma ||\mathbf{x} - \mathbf{z}||_2^2),$$

- Tham số $\gamma > 0$.
- Khi sử dụng thư viện sklearn, rbf là lựa chọn mặc định kernel = 'rbf'.

Examples of SVM Kernels: Sigmoid

Hàm Sigmoid cũng được sử dụng làm kernel:

- Công thức tổng quát của kernel:

$$k(\mathbf{x}, \mathbf{z}) = \tanh(\gamma \mathbf{x}^T \mathbf{z} + r),$$

- Khi sử dụng thư viện sklearn, lựa chọn sigmoid thông qua kernel = 'sigmoid'.

Cách gọi SVM với Kernel từ thư viện sklearn:

```
from sklearn import svm

# ... prepare dataset X and targets Y

# fit the model
for kernel in ('sigmoid', 'poly', 'rbf'):
    clf = svm.SVC(kernel=kernel, gamma=4, coef0 = 0)
    clf.fit(X, Y)
```

Examples of SVM Kernels: User defined

Hàm kernel tự định nghĩa:
Ví dụ về cách gọi SVM với Kernel tự định nghĩa:

```
mport numpy as np
from sklearn import svm
def my_kernel(X, Y):
    return np.dot(X, Y.T)

clf = svm.SVC(kernel=my_kernel)
```