ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Báo cáo đề tài mở rộng

Phân tích và thiết kế giải thuật

SVM - kNN - Decision Tree

GVHD: Nguyễn An Khương
Sinh viên: Bùi Ngô Hoàng Long 1810283

THÀNH PHỐ HỒ CHÍ MINH, 5/2021

Mục lục

1 Support Vector Machine (SVM)

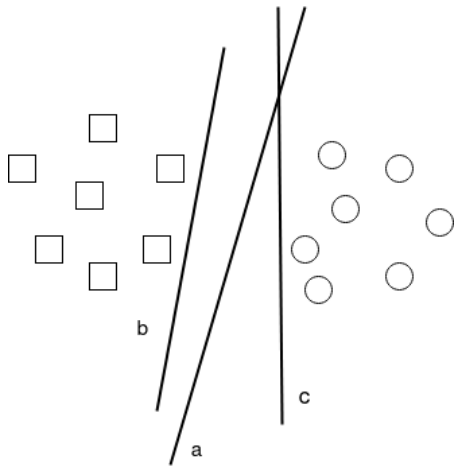
1.1 Bài toán mở đầu

Trong cuộc sống, chúng ta thường sử dụng các đường biên, ranh giới để phân loại sự vật hiện tượng, phân định ranh giới địa lý, lãnh thổ,... Ví dụ một người đang ở trong lãnh thổ Việt Nam thì có thể dự đoán khả năng cao người này là người Việt Nam, ngược lại nếu người này ở ngoài lãnh thổ Việt Nam thì khả năng cao đây không phải là người Việt Nam (mặc dù có người Việt sống ở nước ngoài và có người nước ngoài sống ở Việt Nam nhưng số lượng này là không đáng kể và có thể xem là điểm dữ liệu nhiễu). Nói chung việc lấy một đường biên hay một ranh giới để phân loại sự vật hiện tượng là cách suy nghĩ rất tự nhiên của con người. Trong *học máy* (*machine learning* [Mit+97]) cũng có một giải thuật lấy ý tưởng tương tự đó là *Support Vector Machine* (SVM [CV95]).

1.2 Khái niệm

Support Vector Machine (SVM) là một thuật toán *phân loại nhị phân* (*binary classification*) hoạt động bằng việc xây dựng một *siêu phẳng* (*hyperplane*) có (n - 1) chiều trong không gian n chiều của dữ liệu sao cho siêu phẳng này phân phân chia không gian thành 2 phần tương ứng 2 lớp với mỗi lớp nằm ở một *nửa không gian* (*half space*). Trong không gian 2 chiều thì siêu phẳng này là 1 đường thẳng, trong không gian 3 chiều thì siêu phẳng là mặt phẳng. Ngoài việc xây dựng đường phân chia 2 lớp, SVM còn đảm bảo đường phân chia này là tối ưu nhất.

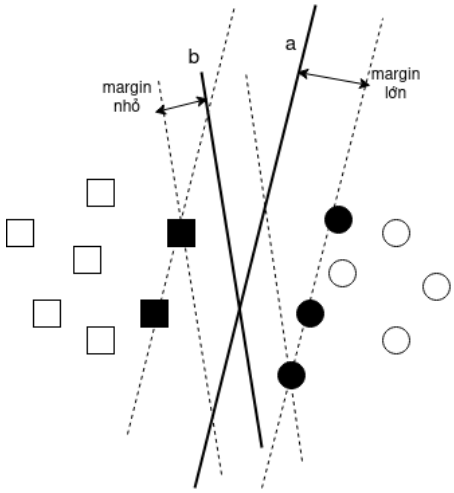
Để thuận tiện, bài viết sẽ xét bài toán trong không gian 2 chiều. Đối với bài toán tổng quát n chiều, các phương pháp tương tự cũng sẽ được sử dụng. Hình 1 biểu diễn việc phân loại 2 lớp tròn và vuông bằng cách xây dựng đường thẳng phân tách không gian thành 2 lớp ứng với lớp vuông và lớp tròn. một điểm dữ liệu mới bất kỳ nếu nằm trong nửa mặt phẳng của lớp vuông sẽ được phân loại vào lớp vuông, ngược lại sẽ phân loại vào lớp tròn.



Hình 1: Phân loại 2 lớp bằng các đường tuyến tính

Trong Hình 1, ta thấy 3 đường thẳng a,b,c đều là những đường phân chia hợp lý. Tuy nhiên, làm thế nào để xác định đâu là đường thẳng tối ưu ? Để đánh giá thế nào là đường thẳng tối ưu, ta

có thể dựa trên nhận xét sau: Nếu đường phân chia nằm quá gần lớp tròn (đường c) thì có khả năng các điểm dữ liệu ở biên của lớp tròn sẽ bị phân loại nhầm vào lớp vuông. Ngược lại, nếu đường phân chia nằm quá gần lớp vuông (đường b) thì những điểm dữ liệu biên của lớp vuông có thể sẽ bị phân loại nhầm vào lớp tròn. Vì vậy, chúng ta cần một đường phân chia sao cho khoảng cách từ điểm gần nhất của mỗi lớp (các điểm được khoanh tròn) tới đường phân chia là như nhau. Khoảng cách như nhau này được gọi là *margin* (*lề*). Các điểm gần nhất của mỗi lớp đến đường phân chia tạo thành 2 *support vector*.



Hình 2: Đường phân chia với margin lớn/nhỏ

Trong Hình 2, cả 2 đường a và b đều "cách đều" 2 lớp dữ liệu, nhưng đường a sẽ tốt hơn vì có margin rộng hơn, tạo ranh giới rạch ròi giữa các lớp. Vậy đường phân chia tối ưu là đường "cách đều" 2 lớp dữ liệu và có margin lớn nhất.

1.3 Bài toán tối ưu cho SVM

Giả sử rằng các cặp dữ liệu của *tập huấn luyện* (*training set*) là $(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)$ với vector $x_i \in R^k$ là đầu vào của một điểm dữ liệu và y_i là *nhãn* (*label*) của điểm dữ liệu đó, k là số chiều của dữ liệu và N là số điểm dữ liệu. Gọi (d): $w^T x + b = 0$ là đường phân chia 2 lớp với w là vector k chiều, b là một số thực. Đường phân chia này chia không gian thành 2 lớp tương ứng với phía dương ($y_i = 1$) và phía âm ($y_i = -1$). Mục tiêu của SVM là tìm giá trị w và b để đây là đường phân chia tối ưu.
Với cặp dữ liệu (x_i,y_i) bất kỳ, khoảng cách từ điểm đó tới mặt phân chia là:

$$distance((x_i, y_i), d) = \frac{|w^T x_i + b|}{||w||_2} \tag{1}$$

Tuy nhiên, do y_i và $w^T x_i + b$ cùng dấu nên ta có:

$$distance((x_i, y_i), d) = \frac{y_i(w^T x_i + b)}{||w||_2} \tag{2}$$

Công thức (2) cho thấy khoảng cách từ một điểm đến đường phân chia không đổi khi đồng thời

tăng/giảm w và b một hệ số a khác không:

$$distance((x_i, y_i), d) = \frac{y_i(w^T x_i + b)}{||w||_2} = \frac{a y_i(w^T x_i + b)}{a||w||_2} = \frac{y_i((aw)^T x_i + (ab))}{||aw||_2} \tag{3}$$

Margin là khoảng cách từ điểm gần nhất của mỗi lớp tới đường phân chia:

$$margin = \min_i \frac{y_i(w^T x_i + b)}{||w||_2} \tag{4}$$

Khi đó bài toán tối ưu của SVM là tìm w và b để $margin$ đạt giá trị lớn nhất:

$$(w, b) = arg \max_{w,b} \{\min_i \frac{y_i(w^T x_i + b)}{||w||_2}\} \tag{5}$$

với $(x_i, y_i) = arg \min_i \frac{y_i(w^T x_i + b)}{||w||_2}$ là các điểm gần đường phân chia nhất (điểm biên của 2 lớp). Các điểm biên này tạo thành 2 support vector như đã giới thiệu ở trên. Vì khoảng cách từ một điểm đến đường phân chia không đổi khi tăng/giảm đồng thời w và b hệ số a nên để thuận tiện trong tính toán ta có thể thêm ràng buộc $y_i(w^T x_i + b) = 1$ cho các điểm (x_i, y_i) trên support vector. Khi đó các điểm còn lại (có khoảng cách tới đường phân chia lớn hơn các điểm biên) sẽ có $y_i(w^T x_i + b) > 1$. Vậy ta có thể đưa bàu toán tối ưu (5) về bài toán tối ưu sau:

$$(w, b) = arg \max_{w,b} \frac{1}{||w||_2} \tag{6}$$

với ràng buộc:

$$y_i(w^T x_i + b) \geqslant 1, \forall i = 1, 2, .., N \tag{7}$$

Để thuận tiện tính toán, ta đưa bài toán (6) về bài toán tương đương sau:

$$(w, b) = arg \min_{w,b} \frac{1}{2}||w||_2^2 \tag{8}$$

với ràng buộc:

$$1 - y_i(w^T x_i + b) \leqslant 0, \forall i = 1, 2, .., N \tag{9}$$

Bằng cách nghịch đảo và bình phương $||w||_2$ ta được một hàm mục tiêu khả vi và bài toán cực đại hóa trở thành bài toán cực tiểu hóa. Trong bài toán (8), hàm mục tiêu là bình phương một *norm* ([KEK05]), nên là một *hàm lồi* ([BNO03]). Các hàm bất đẳng thức ràng buộc là các hàm tuyến tính theo w và b, nên chúng cũng là các hàm lồi. Vậy bài toán tối ưu (8) là một bài toán lồi. Hơn nữa, nó là một *quadratic programming*. Thậm chí, hàm mục tiêu là *strictly convex* vì $||w||_2^2 = w^T I w$ với I là ma trận đơn vị - là một *ma trận xác định dương* (*positive definite matrix* [HJ12]). Từ đây có thể suy ra nghiệm cho SVM là duy nhất.

1.4 Bài toán đối ngẫu cho SVM

Sử dụng phương pháp nhân tử Lagrange ([HBR09]) để giải bài toán (8), ta có Lagrangian của (8) là :

$$\mathcal{L}(w, b, \lambda) = \frac{1}{2}||w||_2^2 + \Sigma_{i=1}^N \lambda_i (1 - y_i(w^T x_i + b)) \tag{10}$$

với $\lambda = [\lambda_1, \lambda_2, ..., \lambda_N]^T$ được gọi là *biến đối ngẫu* (*dual variable*) hoặc *vector nhân tử Lagrange* (*Lagrange multiplier vector*) và $\lambda_i \geqslant 0, \forall i = 1, 2, ..., N$

Hàm đối ngẫu Lagrange ([BBV04]):

$$g(\lambda) = \min_{w,b} \mathcal{L}(w, b, \lambda) \tag{11}$$

với $\lambda \succeq 0$. Tìm hàm đối ngẫu Lagrange là bài toán tìm cực tiểu của $\mathcal{L}(w, b, \lambda)$ theo w và b. Bài toán này có thể giải bằng cách giải hệ phương trình:

$$\begin{cases} \dfrac{\partial \mathcal{L}(w,b,\lambda)}{\partial w} = w - \Sigma_{i=1}^{N} \lambda_i y_i x_i = 0 \\ \dfrac{\partial \mathcal{L}(w,b,\lambda)}{\partial b} = -\Sigma_{i=1}^{N} \lambda_i y_i = 0 \end{cases} \Leftrightarrow \begin{cases} w = \Sigma_{i=1}^{N} \lambda_i y_i x_i \\ \Sigma_{i=1}^{N} \lambda_i y_i = 0 \end{cases} \tag{12}$$

Từ (11) và (12), ta có:

$$g(\lambda) = -\frac{1}{2} \lambda^T K \lambda + Q^T \lambda \tag{13}$$

với $K = V^T V$ là *ma trận nửa xác định dương* (*positive semi definite matrix* [HJ12]) vì $\lambda^T K \lambda = \lambda^T V^T V \lambda = ||V\lambda||_2^2 \geqslant 0, \forall \lambda$. Trong đó: $V = [y_1 x_1, y_2 x_2, ..., y_N x_N]$, Q là vector cột N chiều với mỗi chiều đều mang giá trị 1. Vậy $g(\lambda)$ là một *hàm lõm* (*concave*).

Với bài toán tối ưu hóa có ràng buộc đẳng thức, nghiệm của bài toán tìm cực tiểu Lagrangian (tìm $g(\lambda)$) cũng là nghiệm của bài toán tối ưu gốc. Tuy nhiên ràng buộc (9) của bài tối ưu SVM là ràng buộc bất đẳng thức ($\leqslant$) nên giá trị cực tiểu của Lagrangian chưa phải là cực tiểu của bài toán SVM gốc:

$$g(\lambda) = \min_{w,b} \mathcal{L}(w, b, \lambda) \leqslant L(w_0, b_0, \lambda) \leqslant f(w_0, b_0) \tag{14}$$

với (w_0,b_0) là giá trị *khả thi* (*feasible*) bất kỳ, $f(w, b) = \frac{1}{2}||w||_2^2$ là hàm mục tiêu của bài toán gốc. Dấu $\leqslant$ thứ nhất là hiển nhiên vì cực tiểu một hàm luôn bé hơn hoặc bằng giá trị hàm đó tại điểm bất kỳ. Ta có dấu $\leqslant$ thứ hai từ Công thức (10) với $\Sigma_{i=1}^{N} \lambda_i (1 - y_i(w^T x_i + b)) \leqslant 0$ (vì $\lambda_i \geqslant 0$ và $1 - (y_i w^T x_i + b) \leqslant 0 \ \forall i = 1, ..., N$). Tại $(w_0, b_0) = (w^*, b^*)$ là nghiệm tối ưu của bài toán gốc, ta có:

$$g(\lambda) \leqslant f(w^*, b^*) = p^* \tag{15}$$

với p^* là cực tiểu của bài toán SVM gốc. Vậy nghiệm cực tiểu của (11) luôn bé hơn hoặc bằng nghiệm tối ưu của bài toán gốc. *Bài toán đối ngẫu* SVM là bài toán tìm cực đại $g(\lambda)$ theo λ để chênh lệch tối ưu của hai bài toán là nhỏ nhất có thể ($\max_\lambda g(\lambda)$ càng gần p^* thì nghiệm của (11) có thể dùng làm xấp xỉ nghiệm của bài toán gốc). Bài toán đối ngẫu được định nghĩa:

$$\lambda = arg \max_{\lambda} g(\lambda) \tag{16}$$

với ràng buộc:

$$\begin{cases} \lambda \succeq 0 \\ \Sigma_{i=1}^{N} \lambda_i y_i = 0 \end{cases} \tag{17}$$

Chúng ta biết rằng: nếu một bài toán lồi thoả mãn *tiêu chuẩn Slater* ([Sla14]) thì *strong duality* ([BL10]) thoả mãn ($max_\lambda g(\lambda) = p^*$). Và nếu strong duality thoả mãn thì nghiệm của bài toán chính là nghiệm của *hệ điều kiện Karush-Kuhn-Tucker* (KKT [BMS13]). Ta đã có bài toán tối

ưu (8) là bài toán lồi. Vậy nếu chứng minh bài toán này thỏa mãn tiêu chuẩn Slater thì strong duality thỏa mãn và ta sẽ tìm được nghiệm tối ưu của bài toán thông qua giải hệ điều kiên KKT.

Bước tiếp theo, chúng ta sẽ chứng minh bài toán tối ưu (8) thoả mãn điều kiện Slater. Điều kiện Slater nói rằng : "Nếu tồn tại một điểm *strictly feasible* (và bài toán gốc là lồi), thì strong duality xảy ra.". Nói cách khác, nếu tồn tại w,b thoả mãn:

$$1 - y_i(w^T x_i + b) < 0, \forall i = 1, 2, ..., N \tag{18}$$

thì strong duality xảy ra. Vì 2 lớp là *phân biệt tuyến tính* (*linearly separable*) nên luôn luôn có một siêu phẳng phân chia 2 lớp, tức là bài toán có nghiệm. Khi đó, *feasible set* của bài toán tối ưu (8) phải khác rỗng, tức luôn luôn tồn tại cặp (w_0,b_0) sao cho:

$$1 - y_i(w_0^T x_i + b_0) \leqslant 0, \forall i = 1, 2, ..., N \tag{19}$$

$$\Leftrightarrow 2 - y_i((2w_0)^T x_i + (2b_0)) \leqslant 0, \forall i = 1, 2, \ldots, N \tag{20}$$

Vậy chỉ cần chọn $w_1 = 2w_0$ và $b_1 = 2b_0$, ta sẽ có: $1 - y_i(w_1^T x_i + b_1) \leqslant -1 < 0, \forall i = 1, 2, ..., N$. Vậy, điều kiện Slater thoả mãn. Nói cách khác strong duality xảy ra và nghiệm của bài toán tối ưu (8) là nghiệm của hệ phương trình KKT:

$$\begin{cases} 1 - y_i(w^T x_i + b) \leqslant 0, \forall i = 1, 2, ..., N & (21) \\ \lambda_i \geqslant 0, \forall i = 1, 2, ..., N & (22) \\ \lambda_i(1 - y_i(w^T x_i + b)) = 0, \forall i = 1, 2, ..., N & (23) \\ w = \Sigma_{i=1}^N \lambda_i y_i x_i & (24) \\ \Sigma_{i=1}^N \lambda_i y_i = 0 & (25) \end{cases}$$

Công thức (23) cho thấy, với $\lambda_i \neq 0$, ta có $y_i(w^T x_i + b) = 1$. Đây là những điểm nằm trên support vector. Vậy sau khi tìm được λ từ bài toán (16) ta chỉ cần quan tâm đến những điểm "biên" có $\lambda_i \neq 0$ để tính w (dựa vào (24)) và b (dựa vào (23) và (25)). Gọi tập hợp $S = \{i : \lambda_i \neq 0\}$ là tập hợp chứa các điểm "biên". Với mỗi $i \in S$, ta có: $1 = y_i(w^T x_i + b) \Leftrightarrow b = y_i - w^T x_i$ Mặc dù từ chỉ một cặp (x_i,y_i), ta có thể suy ra ngay được b nếu đã biết w, một phiên bản khác để tính b thường được sử dụng và được cho là ổn định hơn trong tính toán là:

$$b = \frac{1}{|S|}\Sigma_{i \in S}(y_i - w^T x_i) = \frac{1}{|S|}\Sigma_{i \in S}(y_i - \Sigma_{m \in S}\lambda_m y_m x_m^T x_i) \tag{26}$$

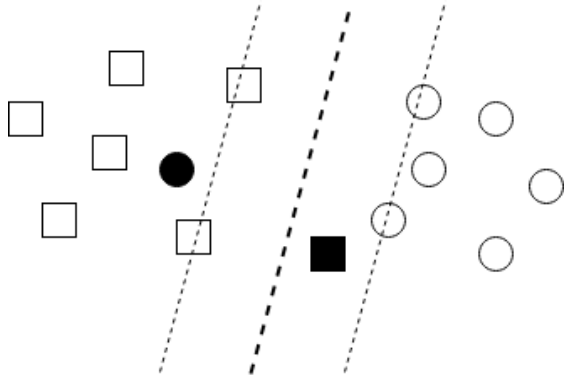
tức trung bình cộng của mọi cách tính b. Để xác định một điểm x mới thuộc vào lớp nào, ta dùng công thức:

$$class(x) = sign(w^T x + b) = sign(\Sigma_{m \in S}\lambda_m y_m x_m^T x + \frac{1}{|S|}\Sigma_{i \in S}(y_i - \Sigma_{m \in S}\lambda_m y_m x_m^T x_i)) \tag{27}$$

1.5 Soft Margin SVM

Trong bài toán xây dựng siêu phẳng phân chia 2 lớp được giới thiệu ở trên, ta đã sử dụng một giả thiết quan trọng là 2 lớp cần phân loại là phân biệt tuyến tính.

Trong Hình 3, không tồn tại đường thẳng phân chia 2 lớp vuông và tròn. Vậy bài toán SVM được giới thiệu ở trên là vô nghiệm. Tuy nhiên, nếu ta chấp nhận "hy sinh" (chấp nhận phân



Hình 3: Soft margin phân chia 2 lớp gần phân biệt tuyến tính

loại sai) một số điểm nhiễu (tô đen) thì ta có thể xây dựng một đường phân chia tương đối tốt. Trong Hình 3, margin tạo bởi đường phân chia và đường nét đứt mảnh còn được gọi là *biên mềm* (*soft margin*). Cũng theo cách gọi này, SVM thuần còn được gọi là SVM *biên cứng* (*hard margin SVM*). Vậy bài toán Soft Margin SVM là bài toán tìm soft margin thích hợp phân chia 2 lớp. Một cách để giải bài toán này là đưa bài toán về bài toán tối ưu có ràng buộc và giải dựa trên hàm đối ngẫu tương tự như bài toán SVM thông thường. Một phương pháp khác được trình bày ở phần này là sử dụng *gradient descent* ([Lem12]) giống như ý tưởng của các *mạng nơ-ron* (*neural network* [Pic04]).

Mục tiêu của bài toán là tìm giá trị w và b để được đường phân chia "tốt" nhất có thể. Vì vậy cần có một hàm số đánh giá độ "tốt". Hàm số này phải có tính chất càng tiến về cực trị (cực tiểu hoặc cực đại) nếu đường phân chia "tốt", và tiến ra xa cực trị nếu đường phân chia "không tốt". Soft margin SVM sử dụng *hàm mất mát hinge* (*hinge loss* [Ros+04]):

$$L_i(w, b) = \max(0, 1 - y_i(w^T x_i + b)) \tag{28}$$

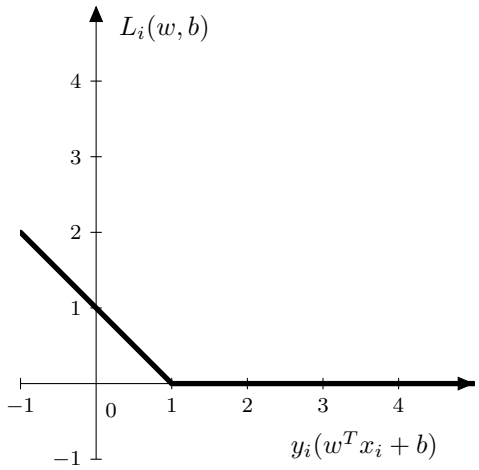
Đồ thị hàm hinge được cho bởi Hình 4.

Trong Hình 4, những điểm trong vùng "an toàn" (vùng mà khả năng chính xác của phép phân loại tương đối cao), tức những điểm nằm xa đường phân chia về phía lớp của chúng hơn các support vector, ứng với $y_i(w^T x_i + b) > 1$ sẽ có mất mát bằng 0. Những điểm nằm giữa support vector của lớp tương ứng và đường phân chia ứng với $0 < y_i(w^T x_i + b) < 1$ có mất mát tương đối nhỏ nằm trong khoảng $(0, 1)$. Những điểm bị phân loại nhầm bởi đường phân chia (nằm phía bờ bên kia của đường phân chia) ứng với $y_i(w^T x_i + b) < 0$ sẽ gây ra mất mát lớn hơn 1. Càng nằm xa đường phân chia về phía của lớp khác thì mất mát càng cao. Vì vậy hàm mất mát hinge có thể được dùng làm hàm đánh giá độ "tốt". Xét trên tập huấn luyện N phần tử, ta dùng *hàm chi phí* (*cost function*) là tổng mất mát của từng điểm dữ liệu:

$$J(w, b) = \Sigma_{i=1}^{N} L_i(w, b) \tag{29}$$

Vì đường phân chia không đổi nếu nhân với (w, b) một hệ số $a \neq 0$ nên (aw, ab) cũng là một nghiệm của bài toán. Khi đó bài toán có thể dẫn tới nghiệm không ổn định vì các hệ số của nghiệm có thể lớn tuỳ ý. Để tránh tình trạng này, số hạng *regularization* cần được thêm vào hàm chi phí:

$$J(w, b) = \Sigma_{i=1}^{N} L_i(w, b) + \lambda R(w, b) \tag{30}$$



Hình 4: Đồ thị hàm mất mát hinge

với $R(w,b)$ là *hàm regularization*, $\lambda > 0$ là *regularization parameter*.

Thông thường $R(w,b)$ được chọn là:

$$R(w,b) = \frac{1}{2}||w||_2^2 \tag{31}$$

Để thuận tiện trong tính toán, bộ trọng số w, b thường được gộp lại thành $\bar{w} = [w^T, b]^T$ và các điểm dữ liệu được thay bằng $\bar{x}_i = [x_i^T, 1]^T$. Khi đó hàm chi phí được viết gọn lại là:

$$J(\bar{w}) = \Sigma_{i=1}^{N} L_i(\bar{w}) + \lambda R(\bar{w}) \tag{32}$$

với

$$L(\bar{w}) = \Sigma_{i=1}^{N} \max(0, 1 - y_i \bar{w}^T \bar{x}_i) \tag{33}$$

$$R(\bar{w}) = \frac{1}{2}||w||_2^2 \tag{34}$$

Vậy bài toán cần giải quyết là tìm ("học") bộ trọng số $\bar{w}$ để cực tiểu hóa hàm mất mát (32). Quá trình "học" này được thực hiện bởi gradient descent. Ý tưởng cơ bản của gradient descent dựa trên nhận xét sau: *Cực tiểu địa phương* (*local optimal point*) của một hàm lồi cũng là *giá trị nhỏ nhất* của hàm đó (*global optimal point*). Khi lần lượt "đi" ngược chiều đạo hàm ta sẽ đến được điểm gần cực tiểu địa phương (cũng chính là cực tiểu toàn cục). Hai điều kiện của gradient descent là hàm chi phí (32) là hàm lồi và khả vi. Ta có $1 - y_i(\bar{w}^T \bar{x}_i)$ là hàm tuyến tính nên lồi, 0 là hằng số nên lồi, cực đại của 2 hàm lồi cũng là hàm lồi vậy từ (33) thì $L(\bar{w})$ cũng là hàm lồi. Đây cũng là 1 lý do soft margin SVM chọn hinge làm hàm mất mát. Bình phương norm 2 là một hàm lồi nên từ (34) ta cũng có $R(\bar{w})$ là hàm lồi. Vì hàm chi phí (32) là tổng của các hàm lồi nên cũng là hàm lồi. Về điều kiện khả vi, ta có $R(\bar{w})$ là bình phương norm 2 nên khả vi. Hàm mất mát hinge liên tục, và có đạo hàm tại gần như mọi nơi trừ điểm có hoành độ bằng 1. Nhưng tại 1, ta có thể coi như đạo hàm của nó bằng 0. Vậy hàm chi phí (32) cũng thỏa điều kiện khả vi.

Quá trình đi ngược chiều đạo hàm được thể hiện bởi:

$$\bar{w} = \bar{w} - \alpha \frac{\partial J(\bar{w})}{\partial \bar{w}} \tag{35}$$

với α là *tốc độ học* (*learning rate*) thể hiện tốc độ thay đổi của $\bar{w}$ trong mỗi lần lặp. Giá trị α phải được chọn phù hợp: nếu quá nhỏ thì sẽ tiêu tốn nhiều thời gian để đi đến điểm tối ưu (vì mỗi lần di chuyển 1 khoảng rất nhỏ), nếu quá lớn thì có thể sẽ đi vượt quá điểm tối ưu làm cho kết quả không hội tụ.

Ta có:

$$\frac{\partial L(\bar{w})}{\partial \bar{w}} = \begin{cases} 0 & ,1 - y_i \bar{w}^T \bar{x}_i \leqslant 0 \\ -y_i \bar{x}_i & ,1 - y_i \bar{w}^T \bar{x}_i > 0 \end{cases} \tag{36}$$

$$\frac{\partial R(\bar{w})}{\partial \bar{w}} = [w\ 0] \tag{37}$$

Vậy bộ trọng số sẽ được cập nhật theo công thức:

$$\bar{w} = \bar{w} - \alpha(\Sigma_{i \in S}(-y_i \bar{x}_i) + \lambda[w\ 0])$$

với S là tập hợp các điểm dữ liệu nằm ngoài support vector của lớp tướng ứng (có $1 - y_i \bar{w}^T \bar{x}_i > 0$).

1.6 Mã giả

Input: X, y, α, λ, num_loop
Result: Xây dựng biên mềm phân chia 2 lớp
Khởi tạo $\bar{X}$;
Khởi tạo bộ trọng số cần học $\bar{w}$;
while *số lần lặp* $<$ *num_loop* **do**
  Khởi tạo tập S chứa các điểm có $1 - y_i \bar{w}^T \bar{x}_i > 0$;
  Tính hinge_gradient $= \Sigma_{i \in S} - y_i \bar{x}_i$;
  Tính regularization_gradient $= [w\ 0]$;
  $\bar{w} = \bar{w}$ - α*(hinge_gradient + λ*regularization_gradient)
end
return $\bar{w}$;

Algorithm 1: Giải thuật Soft Margin SVM

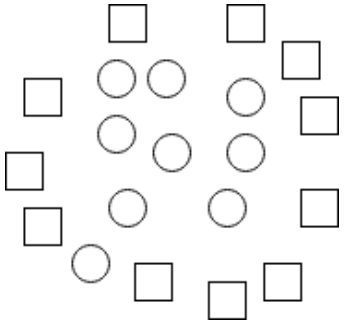
Giải thuật 1 là mã giả hiện thực Soft Margin SVM. Trong đó, X là ma trận kxN với k là số thuộc tính, N là số dữ liệu trong tập huấn luyện. y là ma trận 1xN chứa nhãn, α là tốc độ học, λ là hệ số regularization, num_loop là số lần cập nhật trọng số. $\bar{X}$ được tạo từ X bằng cách thêm vào X thuộc tính thứ k+1 có giá trị bằng 1. $\bar{w}$ là bộ trọng số cần học k+1 chiều với k chiều đầu ứng với w và chiều cuối ứng với b trong phương trình siêu phẳng phân chia 2 lớp.

Mã Python cho giải thuật trên được hiện thực trong file "svm.py" đính kèm.

1.7 Kernel trong SVM

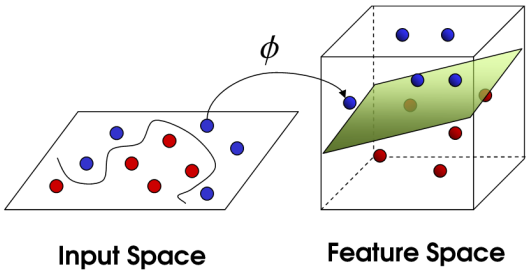
Trong thực tế thì các lớp cần phân loại là không phân biệt tuyến tính, thậm chí cũng không gần phân biệt tuyến tính, khi đó cả SVM và Soft Margin SVM đều không thể giải quyết được. Hình

5 là ví dụ về hai lớp không phân biệt tuyến tính: không thể phân chia lớp vuông và tròn bằng 1 đường thẳng.



Hình 5: 2 lớp dữ liệu không phân biệt tuyến tính

Trong phần này, ta sẽ sử dụng 1 phương pháp để biến 2 lớp không phân biệt tuyến tính thành phân biệt tuyến tính để áp dụng các giải thuật giới thiệu ở trên: dùng một hàm toán học để biến tập dữ liệu sang không gian mới nhiều chiều hơn sao cho 2 lớp trở nên phân biệt tuyến tính. Hình 6 ([Hon15]) mô tả tập dữ liệu được ánh xạ vào không gian mới để 2 lớp phân biệt tuyến tính.



Hình 6: Ánh xạ tập dữ liệu sang không gian mới để được 2 lớp phân biệt tuyến tính

Với ϕ là *ánh xạ đặc trưng* (*feature mapping*), ánh xạ các *thuộc tính* (*attribute*) sang các *đặc trưng* (*feature*). Thay vì áp dụng SVM với các thuộc tính x, ta có thể sử dụng các đặc trưng $\phi(x)$. Để làm việc này, ta chỉ cần thay tất cả x ở giải thuật trước đó thành $\phi(x)$.Tuy nhiên việc tính $\phi(x)$ cho từng x có thể tốn nhiều chi phí tính toán.

May mắn thay, trong giải thuật có thể tồn tại các thành phần là tích vô hướng x và z, nên ta cần thay thế chúng bằng các tích vô hướng $\phi(x)$, $\phi(z)$. Với mỗi ánh xạ đặc trưng ϕ, ta định nghĩa một *kernel* tương ứng:

$$K(x, z) = \phi(x)^T \phi(z) \tag{38}$$

Điều thú vị ở đây là, việc K(x, z) thường không khó khăn, kể cả khi bản thân $\phi(x)$ rất khó tính toán. Điều này có nghĩa là, ta có thể sử dụng SVM để học với các đặc trưng trong không gian nhiều chiều được cho bởi ϕ mà không cần biết rõ $\phi(x)$ chính xác là gì.

Sau đây là một số hàm kernel thông dụng được sử dụng rộng rãi trong các ứng dụng thực tế:

Linear:

$$K(x,z) = x^T z \tag{39}$$

Polynomial:

$$K(x,z) = (r + \gamma x^T z)^d \tag{40}$$

Radial Basic Function:

$$K(x,z) = e^{-\gamma||x-z||_2^2}, \gamma > 0 \tag{41}$$

Sigmoid:

$$K(x,z) = tanh(\gamma x^T z + r) \tag{42}$$

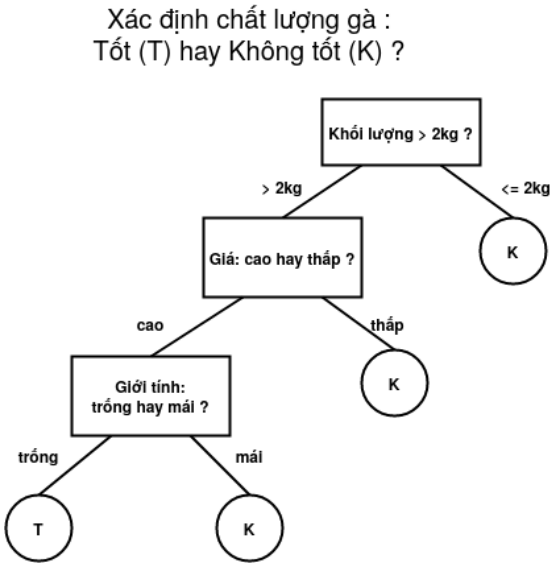
Ngoài các hàm thông dụng trên, còn có nhiều hàm khác có thể sử dụng làm kernel. Tuy nhiên các hàm kernel đều phải thỏa *định lý Mercer* ([GKS16]): Cho $K : R^n$ x $R^n \longmapsto R$. K được gọi là kernel hợp lệ (valid kernel) khi và chỉ khi với bất kì $x^{(1)}, ..., x^{(m)} (m < \infty)$, *ma trận kernel* tương ứng (*kernel matrix*) là đối xứng nửa xác định dương. Trong đó, ma trận kernel được định nghĩa là một mà trận vuông kích thước m x m, với mỗi phần tử tại (i, j) được tính bằng $K_{ij} = K(x^{(i)}, x^{(j)})$.

2 Decision Tree và Random Forest

2.1 Decision Tree

2.1.1 Bài toán mở đầu

Một ông chủ tiệm phở gà sau nhiều năm kinh nghiệm trong nghề đưa ra quy tắc xác định gà ngon dựa trên 3 tiêu chí cân nặng, giá cả và giới tính như sau: gà chỉ ngon khi lớn hơn 2kg, có giá cao và là là trống. Những trường hợp còn lại không phải là gà ngon. Để dễ dàng truyền đạt cho nhân viên, quy tắc này được trình bày bằng sơ đồ ở Hình 7:



Hình 7: Sơ đồ xác định chất lượng gà

Trong đó việc đưa ra quyết định cuối cùng (T hay K) ở các ô tròn phụ thuộc vào việc trả lời câu hỏi ở các ô hình chữ nhật. Ví dụ này cho thấy việc đưa ra các quyết định của con người trong cuộc sống thường được dựa trên việc trả lời các câu hỏi về những thuộc tính của sự vật, hiện tượng có liên quan. Hệ thống các câu hỏi - trả lời này có thể được mô phỏng dưới dạng sơ đồ hình cây. Tương tự, trong học máy cũng có một mô hình ra quyết định dựa trên các câu hỏi. Mô hình này có tên là *Decision Tree* ([KJS18]).

2.1.2 Khái niệm

Decision Tree là một mô hình *học có giám sát* (*supervised learning* [RN02]), có thể được áp dụng vào cả hai bài toán *phân lớp* (*classification*) và *hồi quy* (*regression*). Tuy nhiên mô hình này hiệu quả và phổ biến hơn cho các bài toán phân lớp. Tài liệu này chỉ giới thiệu Decision Tree trong bài toán phân lớp. Decision Tree được thể hiện dưới dạng *lưu đồ* (*flowchart*) có cấu trúc hình cây, gồm các thành phần:

- **Nút lá** (*leaf node*) các node không có con (các ô hình tròn trong Hình 7) chứa nhãn thể hiện loại mà điểm dữ liệu được phân vào dựa theo các thuộc tính đã phân tích.
- **Nút nội** (*internal node*) các node có ít nhất 2 con (các ô hình nhữ nhật trong Hình 7) thể hiện một tiêu chí, thường gọi là thuộc tính để đánh giá. Thuộc tính có thể ở dạng *rời rạc* (*categorical*) như màu sắc, đẹp hay xấu,... và cũng có thể ở dạng *liên tục* (*continuous*) như chiều dài, khối lượng, nhiệt độ,...
- **Nút gốc** (*root node*) là node không có cha (nút nội đầu tiên).
- **Nhánh** (*branch*) đoạn thẳng nối các node thể hiện giá trị của một thuộc tính. Các *con đường* (*path*) từ nút gốc qua các nút nội và đến các nút lá thể hiện các quy tắc phân lớp.

Việc xây dựng một Decision Tree trên dữ liệu huấn luyện cho trước là việc đi xác định các câu hỏi và thứ tự của chúng. Các câu hỏi thường được áp dụng lên từng thuộc tính, hoặc một tổ hợp tuyến tính của các thuộc tính, nhưng cách thứ nhất phổ biến hơn vì tính đơn giản của nó. Các câu hỏi thường ở dạng: Nó rơi vào loại nào? (đối với thuộc tính dạng rời rạc) hay Nó nằm trong ngưỡng nào? (đối với thuộc tính dạng liên tục). Thứ tự câu hỏi sẽ được sắp xếp sao cho hoạt động *dự đoán* (*predict*) sau đó sẽ chính xác và nhanh chóng nhất.

2.1.3 Giải thuật ID3

ID3 (viết đầy đủ là Iterative Dichotomiser 3 [Qui86]) liên tục *lặp lại* (*iterative*) việc *phân chia* (*dichotomize*) tập dữ liệu thành 2 hay nhiều nhóm nhỏ hơn tại mỗi bước dựa trên các câu hỏi ở mỗi nút nội. Câu hỏi ở đây chính là một thuộc tính, câu trả lời chính là giá trị của thuộc tính. Tập dữ liệu lúc này sẽ được phân chia thành các tập dữ liệu nhỏ hơn ứng với từng giá trị thuộc tính đó. Để xác định thứ tự hiệu quả của các câu hỏi, tại mỗi bước (ở mỗi nút nội), ta sẽ chọn câu hỏi ứng với thuộc tính "tốt nhất" sao cho sau khi phân chia thì *độ tinh khiết* (*purity*) của các tập con tăng nhiều nhất (hay *độ vẩn đục* (*impurity*) giảm nhiều nhất). Trong đó, một tập dữ liệu gọi là càng tinh khiết nếu tập đó càng ít hỗn loạn (chứa ít phần tử thuộc các lớp khác nhau), và ngược lại càng vẩn đục nếu độ hỗn loạn càng cao (chứa nhiều phần tử thuộc các lớp khác nhau). Vì vậy ta cần phải có một thước đo độ hỗn loạn hay độ vẩn đục của một tập dữ liệu, đó là hàm *Entropy* ([Sha48]).

Hàm Entropy định lượng độ hỗn loạn hay độ vẩn đục của một tập dữ liệu và được định nghĩa như sau :

$$H(p) = -\sum_{i=1}^{n} p_i \log_2(p_i) \tag{43}$$

Trong đó, $p = (p_1, p_2, ..., p_n)$ là một phân phối xác suất của một biến rời rạc x có thể nhận n giá trị khác nhau $x_1, x_2, ..., x_n$ (trong trường hợp này ứng với n lớp) với $p_i = p(x = x_i)$ (là xác suất một điểm dữ liệu thuộc về lớp i), $0 \leq p_i \leq 1$, và $\sum_{i=1}^{n} p_i = 1$. Lưu ý: quy ước $0 \log_2(0) = 0$.

Hàm Entropy $H(p)$ càng lớn thì độ hỗn loạn càng lớn. $H(p)$ đạt giá trị nhỏ nhất bằng 0 nếu tồn

tại 1 giá trị $p_i = 1$ (các giá trị các bằng 0). Khi đó toàn bộ điểm trong tập dữ liệu thuộc về cùng một lớp. Ngược lại, $H(p)$ lớn nhất (dữ liệu hỗn loạn nhất) khi tất cả các p_i bằng nhau (và bằng $\frac{1}{n}$).

Trở lại bài toán xây dựng Decision Tree với n lớp. Giả sử ta đang ở nút nội với tập S chứa các điểm dữ liệu. Đại lượng *Information Gain* ([Qui86]) ứng với một thuộc tính x (có tập giá trị $\{x_1, x_2, ..., x_m\}$) được định nghĩa như sau:

$$G(x, S) = H(S) - \sum_{j=1}^{m} \frac{|S_j|}{|S|} H(S_j) \tag{44}$$

Trong đó S_j là tập các điểm dữ liệu con có $x = x_j$ và $H(S)$, $H(S_j)$ là Entropy của các tập S, S_j tương ứng.

Do Information Gain đo độ giảm Entropy (giảm hỗn loạn, hay giảm vẩn đục) khi thực hiện phân loại trên một thuộc tính x nên Information Gain càng lớn thì phép phân loại trên thuộc tính này càng "tốt". Vậy thuộc tính "tốt nhất" được chọn để phân loại là thuộc tính x^* với:

$$x^* = \arg\max_x G(x, S) \tag{45}$$

2.1.4 Giải thuật CART

Bên cạnh ID3 chia các node dựa vào Information Gain, giải thuật *CART* (Classification and Regression Tree [SG14]) dùng một thước đo khác để chia các node, đó là *Gini Index* ([RS04]). Về độ chính xác phân loại thì cả hai giải thuật tương tự nhau nhưng giải thuật với Gini Index sẽ nhanh hơn Information Gain vì giảm phép toán log. Tuy nhiên một nhược điểm của CART là mỗi lần chỉ phân được tối đa 2 nhánh.

Gini Index là công thức đo lường tần suất một điểm dữ liệu ngẫu nhiên được phân loại không chính xác. Điều này nghĩa là thuộc tính với Gini Index thấp sẽ được ưu tiên chọn làm tiêu chí phân chia node. Gini Index được định nghĩa như sau:

$$Gini(a = a_j) = 1 - \sum_{i=1}^{c} (p(i|j))^2 \tag{46}$$

với a là thuộc tính đang xét, c là số lớp (c thường bằng 2) và $p(i|j)$ là xác suất điểm dữ liệu có thuộc tính $a = a_j$ rơi vào lớp i.

Giả sử chia node đang xét dựa trên thuộc tính a. $Gini(a = a_j)$ đạt giá trị nhỏ nhất bằng 0 khi có một giá trị $p(i|j) = 1$ (các giá trị còn lại bằng 0). Điều này đồng nghĩa với mọi điểm trong node con có thuộc tính $a = a_j$ đều thuộc cùng một lớp i (hay nói cách khác là node này tinh khiết 100%). Ngược lại, $Gini(a = a_j)$ sẽ lớn nhất nếu các $p(i|j)$ đều bằng nhau và bằng $\frac{1}{n}$ khi các điểm trong node này phân phối đều vào các lớp. Như vậy công thức (46) có thể được dùng để đo độ tinh khiết (đồng nhất) của một node: *Gini* càng nhỏ thì càng tinh khiết.

Vậy để chọn thuộc tính tối ưu nhất cho việc phân chia một node, ta dựa vào giá trị sau:

$$Gini(a) = \sum_{i=1}^{m} \frac{n_i}{n} Gini(a = a_i) \tag{47}$$

với m là số phần tử trong tập giá trị của thuộc tính a, n là số điểm dữ liệu trong node cần chia, n_i là số phần tử trong node con thứ i (có $a = a_i$). Thuộc tính a^* có $Gini(a^*)$ nhỏ nhất sẽ được chọn để chia node vì hệ số này càng nhỏ thì phép chia càng tối ưu.

Bên cạnh, Information Gain và Gini Index đã trình bày, việc phân chia một node khi xây dựng Decision Tree từ tập dữ liệu huấn luyện còn có thể dựa trên các thước đo khác như *Gain Ratio*, *Reduction in Variance*, hay *Chi-Square*.

2.1.5 Điều kiện dừng

Một vấn đề chung đối với các thuật toán Decision Tree là cây tạo ra sẽ cho 100% dự đoán đúng đối với tập dữ liệu huấn luyện nếu ta tiếp tục phân chia các node chưa tinh khiết mà không đưa ra một giới hạn hợp lý. Khi đó, cây không những có thể sẽ rất phức tạp (nhiều node) với nhiều nút lá chỉ có một vài điểm dữ liệu, mà còn ảnh hưởng đến tính đúng đắn khi dự đoán những mẫu dữ liệu không nằm trong tập huấn luyện ban đầu do *quá khớp* (*overfitting*).

Để tránh quá khớp, một trong số các phương pháp sau có thể được sử dụng để làm điều kiện dừng quá trình phân chia node:

- node có Entropy bằng 0
- node phải có số phần tử nhỏ hơn một *ngưỡng* (*threshold*) nào đó
- node có khoảng cách đến nút gốc phải nhỏ hơn ngưỡng nào đó (giới hạn chiều sâu của cây)
- giới hạn số nút lá tối đa của cây
- .....

Trong đó, 2 phương pháp hiệu quả và phổ biến nhất là *pruning* (sẽ được trình bày ngay trong mục này) và Random Forest ([Ho95]) (trình bày ở phần 2.2).

Trong pruning, các cành và lá của cây sẽ được "cắt tỉa" sao cho không ảnh hưởng nhiều đến kết quả dự đoán của cây đối với tập dữ liệu huấn luyện. Một phương pháp pruning thường được dùng đó là *Reduced Error Pruning*. Phương pháp này tách tập huấn luyện ban đầu thành một tập huấn luyện nhỏ hơn và một tập kiểm tra (validation set). Decision Tree được xây dựng trên tập huấn luyện cho tới khi mọi điểm trong tập huấn luyện được phân lớp đúng. Sau đó cắt tỉa các nút lá và giữ lại node bố mẹ (parents node) nếu độ chính xác trên tập kiểm tra được cải thiện. Lúc này, các node bố mẹ đó trở thành nút lá với lớp được gán cho nó là lớp có số điểm dữ liệu chiếm đa số trong node. Quá trình này lặp lại cho đến khi độ chính xác trên tập kiểm tra không được cải thiện nữa thì dừng.

2.1.6 Mã giả

Result: Xây dựng được Decision Tree
Khởi tạo nút gốc với tập S là tập dữ liệu huấn luyện ban đầu;
while *số thuộc tính chưa chọn > 0* **do**
  Khởi tạo biến selected_attribute để lưu thuộc tính được chọn;
  Khởi tạo biến max_IG = 0;
  for *từng thuộc tính* **do**
    Tính Information Gain IG cho thuộc tính hiện tại;
    if *IG > max_IG* **then**
      max_IG = IG;
      selected_attribute = thuộc tính hiện tại;
    end
  end
  Chia tập S bởi thuộc tính selected_attribute;
  Xây dựng các cây con một cách đệ quy;
end

Algorithm 2: Giải thuật Decision Tree

Mã Python cho giải thuật trên được hiện thực trong file "decisiontree.py" đính kèm.

2.1.7 Ví dụ

Trong phần này, ta sẽ từng bước xây dựng Decision Tree từ dữ liệu huấn luyện cho ở Bảng 1. Bảng dữ liệu này mô tả mối quan hệ giữa 4 thuộc tính Giống, Khối lượng, Giới tính, và Giá và Chất lượng gà.

Giống	Khối lượng	Giới tính	Giá	Chất lượng
ri	nhẹ	trống	thấp	không tốt
ri	nhẹ	trống	cao	không tốt
tre	nhẹ	trống	thấp	tốt
tam hoàng	trung bình	trống	thấp	tốt
tam hoàng	nặng	mái	thấp	tốt
tam hoàng	nặng	mái	cao	không tốt
tre	nặng	mái	cao	tốt
ri	trung bình	trống	thấp	không tốt
ri	nặng	mái	thấp	tốt
tam hoàng	trung bình	mái	thấp	tốt
ri	trung bình	mái	cao	tốt
tre	trung bình	trống	cao	tốt
tre	nhẹ	mái	thấp	tốt
tam hoàng	trung bình	trống	cao	không tốt

Bảng 1: Bảng dữ liệu huấn luyên cho Ví dụ 2.1.7

Mặc dù trên thực tế thuộc tính Khối lượng và Giá có thể là các biến liên tục nhưng ta vẫn có thể

áp dụng tương tự các thuộc tính rời rạc nếu đặt ra các ngưỡng phân loại ví dụ: Khối lượng nhỏ hơn 1kg thì xếp vào loại nhẹ, từ 1 đến 2 kg thì trung bình, và trên 2kg thì xem là nặng. Tương tự, ta cũng đặt một ngưỡng cho thuộc tính Giá, nếu Giá nhỏ hơn ngưỡng này thì xếp loại thấp, không thì vào loại cao. Từ tập dữ liệu huấn luyện với 4 thuộc tính này ta sẽ phải xây dựng Decision Tree dự đoán chất lượng gà: phân loại gà vào 1 trong 2 lớp với nhãn là "tốt" và "không tốt".

Gọi A là tập dữ liệu huấn luyện ban đầu với 14 phần tử gồm 5 phần tử thuộc lớp không tốt và 9 phần tử thuộc lớp tốt. Vậy từ Công thức (43) ta tính được $H(A)$:

$$H(A) = -\frac{5}{14}\log_2\frac{5}{14} - \frac{9}{14}\log_2\frac{9}{14} = 0.94$$

Chia node A:

Xét thuộc tính Giống:
Nếu chia node A dựa trên Giống thì theo Công thức (43) các node con sẽ có Entropy:

$$\begin{aligned}
&\text{H(Giống = ri)} &&= -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.971\\
&\text{H(Giống = tre)} &&= -\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4} = 0\\
&\text{H(Giống = tam hoàng)} &&= -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.971
\end{aligned}$$

Vậy từ Công thức (44), ta có Information Gain khi chia node A theo Giống là:

$$\text{G(Giống, A)} = H(A) - \frac{5}{14}\text{H(Giống = ri)} - \frac{4}{14}\text{H(Giống = tre)} - \frac{5}{14}\text{H(Giống = tam hoàng)} = 0.246$$

Xét thuộc tính Khối lượng (viết tắt là KL):
Nếu chia node A dựa trên Khối lượng thì theo Công thức 43 các node con sẽ có Entropy:

$$\begin{aligned}
&\text{H(KL = nhẹ)} &&= -\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4} = 1\\
&\text{H(KL = trung bình)} &&= -\frac{2}{6}\log_2\frac{2}{6} - \frac{4}{6}\log_2\frac{4}{6} = 0.918\\
&\text{H(KL = nặng)} &&= -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.811
\end{aligned}$$

Vậy từ Công thức (44), ta có Information Gain khi chia node A theo Khối lượng là:

$$\text{G(KL, A)} = H(A) - \frac{4}{14}\text{H(KL = nhẹ)} - \frac{6}{14}\text{H(KL = trung bình)} - \frac{4}{14}\text{H(KL = nặng)} = 0.029$$

Xét thuộc tính Giới tính (viết tắt là GT):
Nếu chia node A dựa trên Giới tính thì theo Công thức (43) các node con sẽ có Entropy:

$$\begin{aligned}
&\text{H(GT = trống)} &&= -\frac{4}{7}\log_2\frac{4}{7} - \frac{3}{7}\log_2\frac{3}{7} = 0.985\\
&\text{H(GT = mái)} &&= -\frac{6}{7}\log_2\frac{6}{7} - \frac{1}{7}\log_2\frac{1}{7} = 0.592
\end{aligned}$$

Vậy từ Công thức (44), ta có Information Gain khi chia node A theo Giới tính là:

$$\text{G(GT, A)} = H(A) - \frac{7}{14}\text{H(GT = trống)} - \frac{7}{14}\text{H(GT = mái)} = 0.152$$

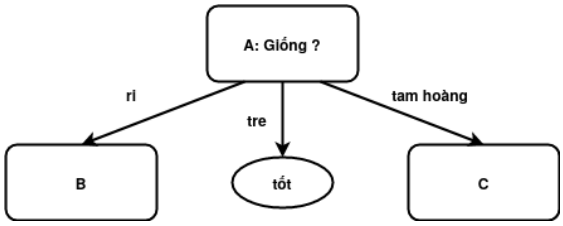
Xét thuộc tính Giá:
Nếu chia node A dựa trên Giá thì theo Công thức (43) các node con sẽ có Entropy:

$$\begin{aligned} \text{H(Giá = thấp)} &= -\frac{2}{8}\log_2\frac{2}{8} - \frac{6}{8}\log_2\frac{6}{8} = 0.811 \\ \text{H(Giá = cao)} &= -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 1 \end{aligned}$$

Vậy từ Công thức (44), ta có Information Gain khi chia node A theo Giá là:

$$\text{G(Giá, A)} = H(A) - \frac{8}{14}\text{H(Giá = thấp)} - \frac{6}{14}\text{H(Giá = cao)} = 0.048$$

Vì Information Gain của thuộc tính Giống là lớn nhất, nên theo Công thức (45) Giống sẽ được chọn làm thuộc tính để chia node A. Ta được hình cây như Hình 8:



Hình 8: Decision Tree sau khi chia node A

Trong đó tập dữ liệu của B và C được biểu diễn ở Bảng 2 và Bảng 3:

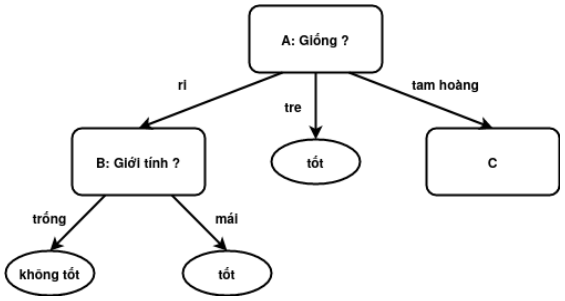
Khối lượng	Giới tính	Giá	Chất lượng
nhẹ	trống	thấp	không tốt
nhẹ	trống	cao	không tốt
trung bình	trống	thấp	không tốt
nặng	mái	thấp	tốt
nặng	mái	cao	tốt

Bảng 2: Tập dữ liệu của node B

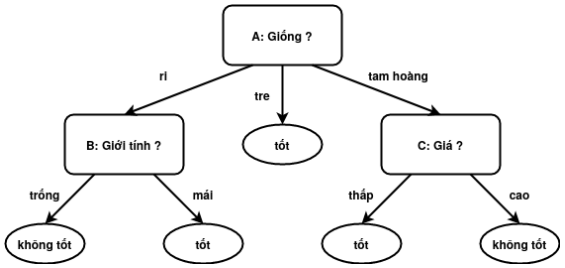
Khối lượng	Giới tính	Giá	Chất lượng
trung bình	trống	thấp	tốt
nặng	mái	thấp	tốt
nặng	mái	cao	không tốt
trung bình	mái	thấp	tốt
trung bình	trống	cao	không tốt

Bảng 3: Tập dữ liệu của node C

Tương tự sau khi chia node B và node C ta có cây như Hình 9 và Hình 10.
Sau các bước trên, ta đã xây dựng được một Decision Tree hoàn chỉnh, sẵn sàng dự đoán Chất lượng gà dựa trên các thuộc tính Giống, Khối lượng, Giới tính, và Giá được cung cấp. Ví dụ, cho gà có 4 thuộc tính: Giống là gà ri, Khối lượng là nhẹ, Giới tính là mái, và Giá cao, ta có thể dự đoán được Chất lượng dựa vào Hình 10, và theo trình tự sau:



Hình 9: Decision Tree sau khi chia node B



Hình 10: Decision Tree sau khi chia node C

1. Bắt đầu tại node A, ta có Giống là gà ri nên sẽ đi theo nhánh bên trái vào node B.
2. Tại node B, ta có Giới tính là mái nên sẽ đi theo nhánh phải và vào node có nhãn tốt.

Vậy dự đoán của ta là gà này có Chất lượng tốt.

2.2 Random Forest

2.2.1 Khái niệm

Random Forest là một mô hình học có giám sát , có thể được áp dụng vào cả hai bài toán phân lớp và hồi quy. Tương tự với rừng (forest) thì phải gồm nhiều cây (tree), Random Forest gồm nhiều Decision Tree được xây dựng từ các tập dữ liệu huấn luyện được xây dựng ngẫu nhiên (random) dựa trên tập huấn luyện ban đầu. Dự đoán từ Random Forest có được bằng cách chọn ra dự đoán chiếm đa số trong các dự đoán từ các Decision Tree. Ý tưởng xây dựng Random Forest dựa trên kinh nghiệm: việc đưa ra một quyết định, lựa chọn dựa trên số đông theo hình thức bỏ phiếu thì kết quả sẽ chính xác hơn so với ý kiến chủ quan của một người dễ sai lệch. Bằng cách này, Random Forest sẽ đưa ra kết quả chính xác hơn mô hình Decision Tree giới thiệu ở phần 2.1 vì tránh được sự quá khớp.

"Ngẫu nhiên" (Random) trong Random Forest được thể hiện qua 2 yếu tố:

1. Ngẫu nhiên (random) tạo tập dữ liệu huấn luyện mới từ tập dữ liệu ban đầu khi xây dựng các Decision Tree. Kỹ thuật này gọi là *bootstrapping*, nghĩa là tập dữ liệu mới sẽ là tập dữ liệu huấn luyện ban đầu nhưng trong đó có một số điểm dữ liệu được sử dụng lặp lại để thay thế một số điểm dữ liệu khác được chọn ngẫu nhiên. Bằng cách này, mỗi cây trong rừng sẽ có tập huấn luyện khác nhau nhưng về cơ bản chỉ sai lệch chút ít so với được huấn luyện từ tập huấn luyện ban đầu.

2. Ngẫu nhiên chọn ra một tập con các thuộc tính để phân chia node trên Decision Tree. Thông thường số thuộc tính chọn ra bằng căn bậc 2 của số thuộc tính ban đầu. Ví dụ, có tổng cộng 16 thuộc tính thì tại mỗi node chỉ dùng ngẫu nhiên 4 node để xem xét cho việc phân chia.

2.2.2 Mã giả

Result: Xây dựng được Random Forest
Chia tập huấn luyện ban đầu thành các tập huấn luyện con một cách ngẫu nhiên. (Mỗi tập con sẽ dùng để xây dựng 1 Decision Tree);
for *mỗi tập huấn luyện con* **do**
    Khởi tạo Decision Tree với nút gốc chứa tập huấn luyện con hiện tại;
    Chọn một tập các thuộc tính một cách ngẫu nhiên;
    Khởi tạo min_GI;
    Khởi tạo biến selected_attribute để chứa thuộc tính dùng để phân chia;
    for *mỗi thuộc tính trong tập thuộc tính được tạo* **do**
        Tính Gini Index GI cho từng thuộc tính.;
        if $GI < min\_GI$ **then**
            min_GI = GI;
            selected_attribute = thuộc tính hiện tại;
        end
    end
    Chia tập huấn luyện con hiện tại bởi selected_attribute;
    Xây dựng các cây con một cách đệ quy;
end

Algorithm 3: Giải thuật Random Forest

3 K-Nearest Neighbours (kNN)

3.1 Bài toán mở đầu

Một người nước ngoài đến Việt Nam và được những người bạn Việt của mình tư vấn về ẩm thực: nếu muốn mua kẹo dừa thì hãy chọn kẹo dừa Bến Tre, muốn ăn phở thì hãy chọn phở Hà Nội, muốn ăn bún bò thì nên chọn bún bò Huế,... Sự tư vấn này dựa trên một kinh nghiệm: Khu vực A có nhiều cửa hàng sản xuất sản phẩm B có chất lượng tốt thì khả năng cao một cửa hàng nhất định sản xuất sản phẩm B cũng có chất lượng tốt nếu nó nằm trong khu vực A (ví dụ: nhiều xưởng ở Bến Tre được xác nhận sản xuất kẹo dừa ngon thì kẹo dừa ở một xưởng bất kỳ tại Bến Tre cũng có rất có khả năng sẽ ngon).

Ngạn ngữ Anh có câu: *"Show me who your friends are and I'll tell you who you are?"* (tạm dịch: *"Hãy cho tôi biết bạn của anh là ai, tôi sẽ chỉ cho anh biết, anh là người như thế nào"*). Như vậy, trong cuộc sống, chúng ta thường phân loại một đối tượng nào đó theo loại của những đối tượng lân cận nó. Trong học máy cũng có một giải thuật có ý tưởng tương tự đó là *k-nearest neighbours* (kNN [Fix85]).

3.2 Khái niệm

kNN là một trong những thuật toán học có giám sát đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong lĩnh vực học máy. Khi huấn luyện, thuật toán này không học một điều gì từ dữ liệu huấn luyện (đây cũng là lý do thuật toán này được xếp vào loại *lazy learning*), mọi tính toán được thực hiện khi nó cần dự đoán kết quả của dữ liệu mới. K-nearest neighbors có thể áp dụng được vào cả hai loại của bài toán học có giám sát là phân loại và hồi quy. k-NN còn được gọi là một thuật toán *Instance-based* hay *Memory-based learning*.

Trong bài toán phân loại, nhãn của một điểm dữ liệu mới được suy ra trực tiếp từ k điểm dữ liệu có khoảng cách gần với nó nhất trong tập huấn luyện. Nhãn của một điểm dữ liệu mới có thể được quyết định bằng *major voting* (bầu chọn theo số phiếu) giữa các điểm gần nhất, hoặc nó có thể được suy ra bằng cách đánh trọng số khác nhau cho mỗi điểm gần nhất đó.

Trong bài toán hồi quy, đầu ra của một điểm dữ liệu sẽ bằng chính đầu ra của điểm dữ liệu đã biết gần nhất (trong trường hợp k=1), hoặc là trung bình có trọng số của đầu ra của những điểm gần nhất, hoặc bằng một mối quan hệ dựa trên khoảng cách tới các điểm gần nhất đó.

Một cách ngắn gọn, kNN là thuật toán đi tìm đầu ra của một điểm dữ liệu mới bằng cách chỉ dựa trên thông tin của k điểm dữ liệu trong tập huấn luyện có khoảng cách gần nó nhất, không quan tâm đến việc có một vài điểm dữ liệu trong những điểm gần nhất này là nhiễu.

Khoảng cách dùng trong kNN có thể được tính bằng nhiều cách: *khoảng cách Euclid* ([Smi11]), *khoảng cách Manhattan* ([Gar97]), *khoảng cách cosine* ([Sin+01]),... Xét hai điểm dữ liệu trong không gian n chiều tương ứng với n thuộc tính: x_1 và x_2. Khoảng cách giữa hai điểm này theo các metrics được tính như sau:
Khoảng cách Euclid (norm 2):

$$d(x_1, x_2) = \sqrt{\sum_{i=1}^{n}(x_{1i} - x_{2_i})^2} \tag{48}$$

Trong không gian 2 chiều hay 3 chiều, khoảng cách Euclid chính là khoảng cách đường chim bay giữa hai điểm dữ liệu. Trong thực tế, nếu cần tính khoảng cách nhiều lần (hàng ngàn lần), phép lấy căn bậc hai có thể được bỏ qua để tăng tốc độ tính toán. Kết quả này vẫn sẽ mang lại tỷ lệ tương tự như khi có căn bậc hai và vẫn hiệu quả trong các giải thuật học máy.

Khoảng cách Manhattan (norm 1):

$$d(x_1, x_2) = \sum_{i=1}^{n} |x_{1i} - x_{2i}| \tag{49}$$

Khoảnh cách Manhattan còn gọi là khoảng cách trong thành phố. Về mặt hình học khoảng cách Manhanttan có thể được hiểu là quảng đường cần đi giữa hai điểm trong không gian n chiều nếu ta chỉ được phép đi song song với một trục tọa độ bất kỳ.

Khoảng cách cosine:

$$d(x_1, x_2) = \frac{x_1.x_2}{\sqrt{\sum_{i=1}^{n} x_{1i}^2}\sqrt{\sum_{i=1}^{n} x_{2i}^2}} \tag{50}$$

với $x_1.x_2$ là tích vô hướng giữa hai vector (dot product):

$$x_1.x_2 = \sum_{i=1}^{n} x_{1i}.x_{2i} \tag{51}$$

ngoài các metric tính khoảng cách giới thiệu ở trên còn có các metric khác như: *khoảng cách Minkowski*, *khoảng cách Hamming*, *khoảng cách Chebyshev*,... cũng thường được áp dụng trong học máy.

3.3 Mã giả

Input: x_0, k, T
Result: x_0 được phân loại
Khởi tạo vector chứa các khoảng cách;
Khởi tạo biến c chứa lớp của x_0;
for *mỗi điểm dữ liệu x trong T* **do**
    Tính khoảng cách d giữa x và x_0;
    Thêm d vào vector chứa khoảng cách;
end
Sắp xếp vector chứa khoảng cách theo thứ tự tăng dần;
Chọn k điểm dữ liệu gần nhất (k-nearest neighbours);
Gán lớp xuất hiện nhiều nhất trong k điểm này cho c;
return c;

Algorithm 4: Giải thuật kNN

Trong giải thuật 4, input là điểm dữ liệu cần phân loại x_0, số điểm dữ liệu lân cận cần xét k và tập huấn luyện T. Giải thuật lặp qua từng điểm dữ liệu trong tập huấn luyện để tính khoảng cách với x_0 (dùng một trong các metric được giới thiệu ở trên). Sau đó chọn ra k điểm dữ liệu gần x_0 nhất. Cuối cùng, xem xét loại nào chiếm đa số trong k điểm dữ liệu này và quyết định đó là loại của x_0.

Mã Python cho giải thuật trên được hiện thực trong file "kNN.py" đính kèm.

3.4 Ví dụ

Dùng lại bài toán xác định chất lượng gà ở phần 2.1.7 nhưng với tập huấn luyện cho bởi Bảng 4: Mục tiêu là dự đoán chất lượng của con gà khối lượng 1.7 (kg) và giá là 1 trăm nghìn đồng.

Khối lượng (kg)	Giá (trăm nghìn đồng)	Chất lượng
1.5	1.0	tốt
1.3	1.2	tốt
2.0	1.0	không tốt
1.6	0.8	không tốt
1.4	1.1	tốt
1.8	1.5	tốt
1.0	0.9	không tốt
1.9	1.2	không tốt
2.0	1.5	tốt
1.6	1.3	không tốt

Bảng 4: Bảng dữ liệu huấn luyện cho Ví dụ 3.4

Giải:

Chọn $k = 3$ và dùng khoảng cách Euclid. Tính khoảng cách của điểm dữ liệu cần phân loại (1.7, 1.0) với các điểm dữ liệu trong tập huấn luyện ta được Bảng 5: Từ Bảng 5, ta thu được 3 điểm

Khối lượng (kg)	Giá (trăm nghìn đồng)	Khoảng cách	Chất lượng
1.5	1.0	0.2	tốt
1.3	1.2	0.45	tốt
2.0	1.0	0.3	không tốt
1.6	0.8	0.22	không tốt
1.4	1.1	0.32	tốt
1.8	1.5	0.51	tốt
1.0	0.9	0.71	không tốt
1.9	1.2	0.28	không tốt
2.0	1.5	0.58	tốt
1.6	1.3	0.32	không tốt

Bảng 5: Khoảng cách từ điểm đang phân loại với các điểm trong tập huấn luyện

có khoảng cách gần nhất là (1.0, 0.9), (2.0, 1.5), (1.8, 1.5) với nhãn tương ứng là "không tốt", "tốt", "tốt". Vì 2 trong 3 điểm lân cận có nhãn tốt nên ta dự đoán điểm dữ liệu cần phân loại cũng có nhãn tốt.

3.5 Tăng cường hiệu quả cho kNN

3.5.1 Chọn k phù hợp

Việc lựa chọn giá trị k cũng ảnh hưởng rất lớn đến độ chính xác của giải thuật:

- Nếu k quá nhỏ thì thì những dữ liệu lân cận được xét có thể là nhiễu, không thể hiện được sự phân bố chính xác của các lớp trong khu vực đang xét (cũng như việc không thể chỉ nhìn vào chất lượng của 1,2 cửa hàng trong khu vực mà đánh giá chất lượng của toàn bộ những của hàng khác trong khu vực này).
- Nếu k quá lớn thì các lớp có số lượng phần tử nhỏ sẽ bị các lớp lớn hơn lấn át, và do đó nhãn của các lớp nhỏ sẽ ít được gán.

Vì vậy việc lựa chọn giá trị k phù hợp rất quan trọng khi sử dụng kNN. Một trong những phương pháp ta có thể sử dụng là thử các giá trị k khác nhau và kiểm tra độ chính xác của thuật toán ứng với từng giá trị k ấy.

3.5.2 Đánh trọng số cho các điểm lân cận

Việc chọn lớp xuất hiện nhiều nhất trong k điểm lân cận làm lớp của điểm cần phân loại xem mức độ phản ánh của k điểm lân cận này đến điểm đang xét là như nhau. Tuy nhiên, dựa vào kinh nghiệm: các điểm gần, xa sẽ có mức độ phản ánh khác nhau, ta có thể thêm trọng số (thể hiện mức độ phản ánh) cho k điểm lân cận này. Việc đánh trọng số phải thỏa mãn điều kiện: điểm càng gần thì trọng số càng cao (phản ánh nhiều), điểm càng xa thì trọng số càng ít (phản ánh ít). Một cách đánh trọng số đơn giản là lấy nghịch đảo khoảng cách, khi khoảng cách bằng 0 (điểm cần phân loại trùng với điểm trong tập huấn luyện) thì sẽ xếp điểm cần phân loại vào lớp của điểm dữ liệu đã biết. Ngoài ra, còn có một cách đánh trọng số thường dùng khác là:

$$w_i = e^{\frac{-||x - x_i||_2^2}{\sigma^2}} \tag{52}$$

trong đó x là điểm cần phân loại, x_i là một điểm trong k lân cận của x_i,w_i là trọng số của điểm x_i, σ là một số dương. Nhận thấy rằng hàm số này cũng thỏa mãn điều kiện: điểm càng gần x thì trọng số càng cao, càng xa thì trọng số càng nhỏ.

3.5.3 Chuẩn hóa dữ liệu

Khi có một thuôc tính (hay phần tử trong vector) có giá trị lớn hơn các thuộc tính khác rất nhiều thì khoảng cách giữa các điểm sẽ phụ thuộc vào thuộc tính này rất nhiều. Điều này lại thường xuất hiện trong thực tế gây ảnh hưởng đến độ chính xác của giải thuật. Trong Ví dụ ở phần 3.4, nếu đơn vị của giá được thay bằng nghìn thì chất lượng gà theo kNN dường như chỉ phụ thuộc giá. Ngược lại nếu thay đơn vị của khối lượng là gam thì chất lượng dường như chỉ phụ vào khối lượng. Để có được kết quả chính xác hơn, một kỹ thuật thường được dùng là *data normalization* (*chuẩn hóa dữ liệu*). Phương pháp này đưa các thuộc tính có khoảng giá trị gốc khác nhau về cùng một khoảng giá trị, thường là từ 0 đến 1, trước khi thực hiện kNN. Có nhiều kỹ thuật chuẩn hóa khác nhau được áp dụng không chỉ với kNN mà còn với hầu hết các thuật toán học máy khác [PJ17].

4 Ứng dụng thực tế

4.1 Mô tả

Phần này sẽ mô tả việc áp dụng các giải thuật Soft Margin SVM, kNN và Decision Tree vào bài toán thực tế. Bài toán cần giải quyết trong phần này là bài toán phân loại email (ham/spam). Dataset được sử dụng là Spam Mails Dataset ([Gar19]) gồm 5171 mẫu dữ liệu với 87% loại ham, 13% loại spam. Toàn bộ các giải thuật đều được hiện thực trong file "SVM-kNN-DecisionTree.ipynb" đính kèm (hoặc có thể truy cập tại link github này hay link google drive này). File "SVM-kNN-DecisionTree.ipynb" gồm 7 mục, trong đó mục I, II, III trình bày việc load dữ liệu từ dataset, load google pretrained word2vec model và các bước tiền xử lý để số hóa dữ liệu dạng văn bản chuẩn bị đưa vào các giải thuật phân lớp. Mục IV, V, VI lần lượt hiện thực các giải thuật Soft Margin SVM, kNN và Decistion Tree và dùng các giải thuật này để giải quyết bài toán phân loại email nêu trên. Mục VII áp dụng các giải thuật trên cho dataset quy mô nhỏ hơn là Banknote Authentication Dataset ([DG17]).

4.2 Kết quả

Bảng 6 và Bảng 7 lần lượt mô tả *độ chính xác phân loại* (*accurracy*) trên tập kiểm tra và thời gian huấn luyện của các giải thuật trên Spam Mails Dataset và Banknote Authentication Dataset. Riêng giải thuật kNN thì thời gian được tính là thời gian phân loại. Nhìn chung, trên bộ dữ liệu nhỏ như Banknote Authentication Dataset, độ chính xác phân loại của các giải thuật tương đối cao và xấp xỉ nhau. Nhưng đối với bộ dữ liệu lớn hơn như Spam Mails Dataset thì độ chính xác của kNN kém hơn đáng kể so với 2 giải thuật còn lại.

Giải thuật	**Độ chính xác** (%)	**Thời gian** (s)
Soft Margin SVM	92	149
kNN	71	31
Decision Tree	82	1205

Bảng 6: Độ chính xác và thời gian chạy các giải thuật trên Spam Mails Dataset

Giải thuật	**Độ chính xác** (%)	**Thời gian** (s)
Soft Margin SVM	98	1.12
kNN	99	2.033
Decision Tree	98	2.35

Bảng 7: Độ chính xác và thời gian chạy các giải thuật trên Banknote Authentication Dataset

References

[Sha48] Claude E Shannon. "A mathematical theory of communication". In: *The Bell system technical journal* 27.3 (1948), pp. 379–423.

[Fix85] Evelyn Fix. *Discriminatory analysis: nonparametric discrimination, consistency properties*. Vol. 1. USAF school of Aviation Medicine, 1985.

[Qui86] J. Ross Quinlan. "Induction of decision trees". In: *Machine learning* 1.1 (1986), pp. 81–106.

[CV95] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[Ho95] Tin Kam Ho. "Random decision forests". In: *Proceedings of 3rd international conference on document analysis and recognition*. Vol. 1. IEEE. 1995, pp. 278–282.

[Gar97] Martin Gardner. "Taxicab geometry". In: *The Last Recreations*. Springer, 1997, pp. 159–175.

[Mit+97] Tom M Mitchell et al. "Machine learning". In: (1997).

[Sin+01] Amit Singhal et al. "Modern information retrieval: A brief overview". In: *IEEE Data Eng. Bull.* 24.4 (2001), pp. 35–43.

[RN02] Stuart Russell and Peter Norvig. "Artificial intelligence: a modern approach". In: (2002).

[BNO03] Dimitri P Bertsekas, Angelia Nedic, and Asuman E Ozdaglar. "Convex analysis and optimization athena scientific". In: *Journal of Mathematical Analysis & Applications* 129.2 (2003), pp. 420–432.

[BBV04] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[Pic04] Gualtiero Piccinini. "The First computational theory of mind and brain: a close look at mcculloch and pitts's "logical calculus of ideas immanent in nervous activity"". In: *Synthese* 141.2 (2004), pp. 175–215.

[RS04] Laura Elena Raileanu and Kilian Stoffel. "Theoretical comparison between the gini index and information gain criteria". In: *Annals of Mathematics and Artificial Intelligence* 41.1 (2004), pp. 77–93.

[Ros+04] Lorenzo Rosasco et al. "Are loss functions all the same?" In: *Neural computation* 16.5 (2004), pp. 1063–1076.

[KEK05] Anthony W Knapp, Charles L Epstein, and Steven G Krantz. *Advanced real analysis*. Springer, 2005.

[HBR09] Laurence Hoffmann, GL Bradley, and KH Rosen. *Applied Calculus for Business, Economics, and the Social and Life Sciences, Expanded Edition*. McGraw-Hill Publishing, 2009.

[BL10] Jonathan Borwein and Adrian S Lewis. *Convex analysis and nonlinear optimization: theory and examples*. Springer Science & Business Media, 2010.

[Smi11] Karl J Smith. *Precalculus: A Functional Approach to Graphing and Problem Solving*. Jones & Bartlett Publishers, 2011.

[HJ12] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[Lem12] Claude Lemaréchal. "Cauchy and the gradient method". In: *Doc Math Extra* 251.254 (2012), p. 10.

[BMS13] Vladimir Boltyanski, Horst Martini, and Valeriu Soltan. *Geometric methods and optimization problems*. Vol. 4. Springer Science & Business Media, 2013.

[SG14] Sonia Singh and Priyanka Gupta. "Comparative study ID3, cart and C4. 5 decision tree algorithm: a survey". In: *International Journal of Advanced Information Science and Technology (IJAIST)* 27.27 (2014), pp. 97–103.

[Sla14] Morton Slater. "Lagrange multipliers revisited". In: *Traces and emergence of nonlinear programming*. Springer, 2014, pp. 293–306.

[Hon15] Ong Xuan Hong. *Chia sẻ kiến thức và thông tin về Machine learning*. https://ongxuanhong.wordpress.com/2015/09/19/support-vector-machine-svm-hoi-gi-dap-nay/. 2015.

[GKS16] Koos Grobler, Rien Kaashoek, and Anton Schep. "Adriaan Cornelis Zaanen". In: *Ordered Structures and Applications: Positivity VII (Zaanen Centennial Conference), 22-26 July 2013, Leiden, the Netherlands*. Springer. 2016.

[DG17] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[PJ17] Amit Pandey and Achin Jain. "Comparative analysis of KNN algorithm using various normalization techniques". In: *International Journal of Computer Network and Information Security* 11.11 (2017), p. 36.

[KJS18] Bogumił Kamiński, Michał Jakubczyk, and Przemysław Szufel. "A framework for sensitivity analysis of decision trees". In: *Central European journal of operations research* 26.1 (2018), pp. 135–159.

[Gar19] Venkatesh Garnepudi. *Spam Mails Dataset*. https://www.kaggle.com/venky73/spam-mails-dataset. 2019.