|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” | Mẫu 2 |

BÁO CÁO CHUYÊN ĐỀ SỐ 4.5.2

“Áp dụng mô hình học máy SVM, KNN”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

Hà Nội - 2023

|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” |  |

BÁO CÁO CHUYÊN ĐỀ SỐ 4.5.2

“Áp dụng mô hình học máy SVM, KNN”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

|  |  |
| --- | --- |
| **Người thực hiện chuyên đề** | **Cơ quan chủ trì** |
| *(Họ tên và chữ ký)* | *(Họ tên và chữ ký)* |

Hà Nội - 2023

MỤC LỤC

[MỤC LỤC 1](#_Toc129729421)

[DANH MỤC HÌNH ẢNH 2](#_Toc129729422)

[1. K-nearest neighbors (KNN) 3](#_Toc129729423)

[1.1. Sử dụng thuật toán KNN 3](#_Toc129729424)

[1.2. Cách xác định giá trị *k* tối ưu 6](#_Toc129729425)

[1.3. Ưu và nhược điểm của KNN 7](#_Toc129729426)

[1.4. Về ứng dụng của thuật toán KNN phải kể đến như: 7](#_Toc129729427)

[1.5. Thực nghiệm 8](#_Toc129729428)

[2. Support Vector Machines (SVM) 10](#_Toc129729429)

[2.1. Các vector hỗ trợ 11](#_Toc129729430)

[2.2. Huấn luyện SVM 15](#_Toc129729431)

[2.3 Thực nghiệm 16](#_Toc129729432)

# DANH MỤC HÌNH ẢNH

[Hình 1.1: Hoạt động của thuật toán KNN 4](#_Toc129729406)

[Hình 1.2: Thêm điểm dũ liệu mới 5](#_Toc129729407)

[Hình 1.3: Biểu diễn và tính khoảng cách hai điểm 5](#_Toc129729408)

[Hình 1.4: Mô hình hoá hàng xóm gần nhất của điểm 6](#_Toc129729409)

[Hình 1.5: Cài đặt thư viện scikit-learn 8](#_Toc129729410)

[Hình 1.6: Kết quả độ chính xác của mô hình 9](#_Toc129729411)

[Hình 2.1: Mô phỏng phân lớp dựa trên SVM 11](#_Toc129729412)

[Hình 2.2: Siêu mặt phẳng phân lớp dữ liệu 11](#_Toc129729413)

[Hình 2.3: Vector hỗ trợ siêu mặt phẳng 12](#_Toc129729414)

[Hình 2.4: Biên phân tách 13](#_Toc129729415)

[Hình 2.5: Các giá trị biên trong SVM 14](#_Toc129729416)

[Hình 2.6: Biểu diễn trực quan trên trục đồ thì 14](#_Toc129729417)

[Hình 2.7: Phân tách bằng đồ thị phức tạp 15](#_Toc129729418)

[Hình 2.8: Cài đặt thư viện scikit-learn 17](#_Toc129729419)

[Hình 2.9: Kết quả độ chính xác của mô hình 18](#_Toc129729420)

# 1. K-nearest neighbors (KNN)

K-nearest neighbor là một trong những thuật toán học máy có giám sát đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong Machine Learning. Khi training, thuật toán này không học một điều gì từ dữ liệu training (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. KNN có thể áp dụng được vào cả hai loại của bài toán phân loại (classification) và hồi quy (regression).

Với KNN, trong bài toán Classification, nhãn (label) của một dữ liệu mới được suy ra từ K điểm dữ liệu gần nhất trong tập dữ liệu tập huấn (training set). Nhãn của một tập dữ liệu kiểm thử (test set) có thể được quyết định bằng bầu chọn theo số phiếu (major voting) giữa các điểm gần nhất đó rồi suy ra nhãn.

Trong bài toán Regression, 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.

Nói 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 dữ liệu tập huấn gần nó nhất (k – lân cận), không quan tâm đến việc có một vài điểm dữ liệu trong những điểm gần nhất này là nhiễu.

## 1.1. Sử dụng thuật toán KNN

Giả sử có hai danh mục, tức là Danh mục A và Danh mục B, và chúng ta có một điểm dữ liệu mới x1, vậy điểm dữ liệu này sẽ nằm trong danh mục nào trong số các danh mục này. Để giải quyết dạng bài toán này, chúng ta cần một thuật toán KNN. Với sự trợ giúp của KNN, chúng ta có thể dễ dàng xác định danh mục hoặc lớp của một tập dữ liệu cụ thể. Sơ đồ:



Hình 1.1: Hoạt động của thuật toán KNN

Hoạt động KNN có thể được giải thích dựa trên thuật toán dưới đây:

* Bước 1: Chọn số K = số láng giềng gần nhất
* Bước 2: Tính khoảng cách Euclide của K số láng giềng
* Bước 3: Lấy K láng giềng gần nhất theo khoảng cách Euclide được tính toán
* Bước 4: Trong số K lân cận này, đếm số điểm dữ liệu trong mỗi loại
* Bước 5: Gán các điểm dữ liệu mới cho danh mục đó mà số lượng hàng xóm là tối đa

Giả sử có một điểm dữ liệu mới và cần đặt nó vào danh mục bắt buộc như trong hình dưới đây:



Hình 1.2: Thêm điểm dũ liệu mới

Đầu tiên, ta sẽ chọn số lượng hàng xóm, ví dụ chọn k = 5.

Tiếp theo, ta tính toán khoảng cách Euclide giữa các điểm dữ liệu. Khoảng cách Euclide là khoảng cách giữa hai điểm mà chúng ta đã học trong hình học, được tính bằng công thức:



Hình 1.3: Biểu diễn và tính khoảng cách hai điểm

Bằng cách tính toán khoảng cách Euclide, chúng ta có được những người hàng xóm gần nhất trong loại A và hai người hàng xóm gần nhất trong loại B như hình dưới đây:



Hình 1.4: Mô hình hoá hàng xóm gần nhất của điểm

Ta thấy, 3 láng giềng gần nhất thuộc loại A, do đó điểm dữ liệu mới này phải thuộc loại A.

## 1.2. Cách xác định giá trị *k* tối ưu

Một số điểm cần nhớ khi chọn giá trị của k:

* k = 1: Mô hình quá cụ thể và không khái quát hóa tốt. Nó cũng có xu hướng nhạt cảm với các dữ liệu bất thường (nhiễu) trong tập dữ liệu. Mô hình đạt được độ chính xác cao trên dữ liệu đào tạo nhưng sẽ là một công cụ dự đoán kém đối với các điểm dữ liệu mới chưa được gặp trước đây. Do đó, chúng ta có thể sẽ tạo ra một mô hình quá khớp (overfit).
* K = 100: Ngược lại với k = 1, mô hình quá tổng quát và không phải là một công cụ dự báo tốt trên cả tập dữ liệu đào tạo và tập dữ liệu kiểm tra. Tình trạng này được gọi là mô hình chưa khớp (underfit).

Không có cách cụ thể nào để xác định giá trị tốt nhất cho “k”, vì vậy chúng ta cần thử một số giá trị để tìm ra giá trị tốt nhất trong số đó. [Scikit-learning](https://vncoder.vn/bai-hoc/machine-learning-thu-vien-scikit-learn-382) cung cấp chức năng GridSearchCV cho phép ta dễ dàng kiểm tra nhiều giá trị k khác nhau.

## 1.3. Ưu và nhược điểm của KNN

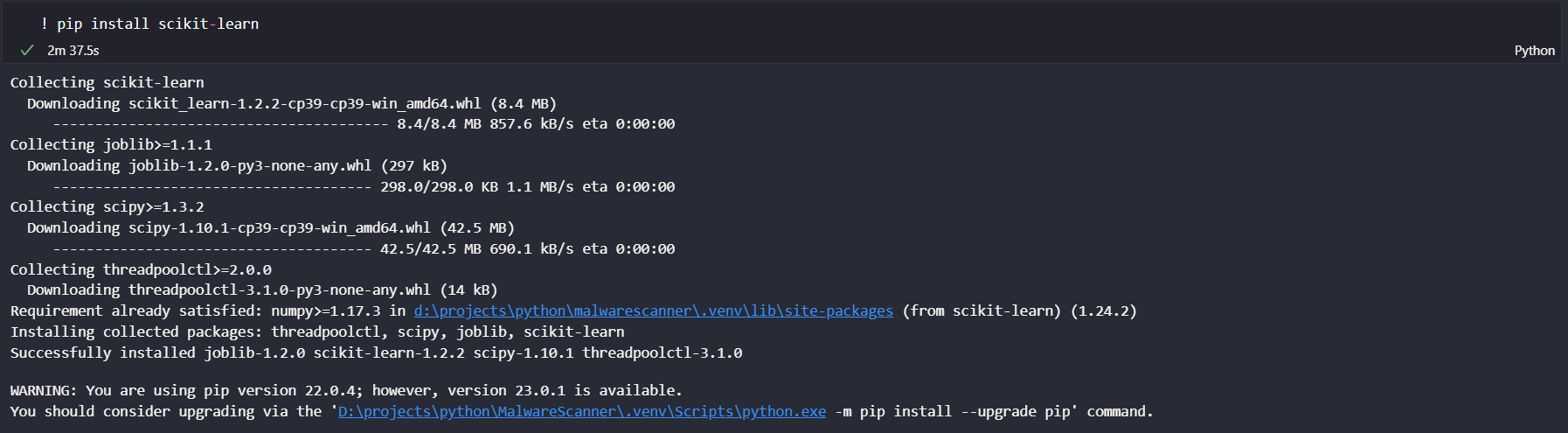
* **Ưu điểm:**
* Đơn giản và dễ giải thích
* Độ phức tạp tính toán của quá trình training bằng 0
* Không dựa trên bất kỳ giả định nào, vì thế nó có thể được sử dụng trong các bài toán phi tuyến tính
* Việc dự đoán kết quả của dữ liệu mới rất đơn giản
* Hoạt động tốt trong trường hợp phân loại với nhiều lớp
* Sử dụng được trong cả phân loại và hồi quy
* **Nhược điểm:**
* Tốn bộ nhớ
* Nhạy cảm với các dữ liệu bất thường (nhiễu khi k nhỏ)
* Mọi tính toán đều nằm ở khâu test. Trong đó việc tính khoảng cách tới từng điểm dữ liệu trong traning set sẽ tốn rất nhiều thời gian, đặc biệt là với các cơ sở dữ liệu có số chiều lớn và có nhiều điểm dữ liệu. Với K càng lớn thì độ phức tạp cũng sẽ tăng lên. Ngoài ra, việc lưu toàn bộ dữ liệu trong bộ nhớ cũng ảnh hưởng tới hiệu năng của KNN.

## ****1.4. Về ứng dụng của thuật toán KNN phải kể đến như:****

* **Trong y tế:** xác định bệnh lý của người bệnh mới dựa trên dữ liệu lịch sử của các bệnh nhân có cùng bệnh lý có cùng các đặc điểm đã được chữa khỏi trước đây, hay xác định loại thuốc phù hợp giống ví dụ chúng tôi trình bày ở trên.
* **Trong lĩnh vực ngân hàng:** xác định khả năng khách hàng chậm trả các khoản vay hoặc rủi ro tín dụng do nợ xấu dựa trên phân tích Credit score; xác định xem liệu các giao dịch có hành vi phạm tội, lừa đảo hay không.
* **Trong giáo dục:** phân loại các học sinh theo hoàn cảnh, học lực để xem xem cần hỗ trợ gì cho những học sinh ví dụ như hoàn cảnh sống khó khăn nhưng học lực lại tốt.
* **Trong thương mại điện tử:** phân loại khách hàng theo sở thích cụ thể để hỗ trợ personalized marketing hay xây dựng hệ thống khuyến nghị, dựa trên dữ liệu từ website, social media.
* **Trong kinh tế nói chung:** giúp dự báo các sự kiện kinh tế trong tương lai, dự báo tình hình thời tiết trong nông nghiệp, xác định xu hướng thị trường chứng khoán để lên kế hoạch đầu tư thích hợp.

1.5. Thực nghiệm

Dựa trên các đặc điểm trên, chúng ta sẽ áp dụng mô hình học máy KNN để chạy thực nghiệm việc phân loại thông qua *sklearn.neighbors.KNeighborsClassifier* của thư viện *scikit-learn*. Để có thể sử dụng được thư viện, ta tiến hành cài đặt thư viện *scikit-learn* thông qua câu lệnh *pip install scikit-learn.*



Hình 1.5: Cài đặt thư viện scikit-learn

Sau khi cài đặt xong, ta tiến hành nạp dữ liệu sử dụng để huấn luyện và kiểm tra từ file có định dạng .csv vào chương trình thông qua thư viện *pandas*.

import pandas as pd

import numpy as np

train\_df=pd.read\_csv(r'dataset/train-0.csv',header=None,skiprows=1)

val\_df=pd.read\_csv(r'dataset/file-0.csv',header=None,skiprows=1)

test\_df=pd.read\_csv(r'dataset/file-1.csv',header=None,skiprows=1)

train\_x = np.array(train\_df.iloc[:, 3:])

train\_y = np.array(train\_df.iloc[:, 2])

val\_x = np.array(val\_df.iloc[:, 3:])

val\_y = np.array(val\_df.iloc[:, 2])

test\_x = np.array(test\_df.iloc[:, 3:])

test\_y = np.array(test\_df.iloc[:, 2])

Ở đây, chúng ta sử dụng bộ dữ liệu được tách ra thành 3 file với 3 mục đích khác nhau, cụ thể là sử dụng để huấn luyện, xác thực và kiểm tra. Ta tiến hành tách các đặc trưng và nhãn từ dữ liệu đã được nạp vào rồi sau đó biến đổi chúng thành đối tượng có dạng dữ liệu kiểu mảng bằng thư viện *numpy.* Tiếp đó, ta nạp thư viện scikit-learn đã cài đặt vào trong chương trình, thực hiện huấn luyện cho mô hình với dữ liệu vừa nạp vào và tiến hành cho mô hình phân loại mã độc.

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

neigh = KNeighborsClassifier(n\_neighbors=228)

neigh.fit(train\_x, train\_y)

y\_pred\_vc = neigh.predict(test\_x)

accuracy = accuracy\_score(test\_y, y\_pred\_vc)

print("KNN:", accuracy)

Số lượng neighbors được sử dụng cho mô hình là 228 được truyền vào dưới dạng tham số thông qua biến *n\_neighbors*. Sau khi tiến hành huấn luyện và cho mô hình phân loại mã độc, ta sử dụng hàm *sklearn.metrics.accuracy\_score* để đánh giá độ chính xác kết quả dự đoán của mô hình với dữ liệu kiểm tra có sẵn. Kết quả thu được từ mô hình có độ chính xác đạt gần 0.81598 (~82%).



Hình 1.6: Kết quả độ chính xác của mô hình

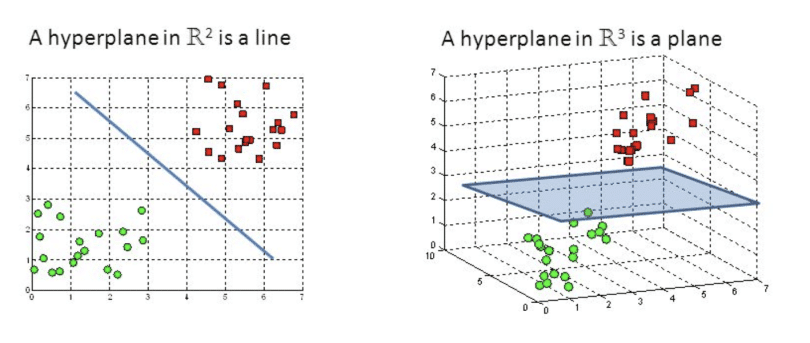
# 

# 2. Support Vector Machines (SVM)

Trong các nghiên cứu thời gian gần đây, phương pháp phân lớp dùng tập phân lớp vector hỗ trợ đang được nghiên cứu và áp dụng tương đối mạnh mẽ trong lĩnh vực phân lớp và nhận dạng. SVM là phương pháp được ra đời từ lý thuyết học thống kê do Vapnik và Chervonenkis nghiên cứu và phát triển. Đây là phương pháp có nhiều tiềm năng phát triển về trong thực tiễn cũng như trong các nghiên cứu lý thuyết. Từ kết quả của những nghiên cứu gần đây cho thấy rằng SVM là phương pháp có khả năng phân lớp khá tốt không chỉ đối với bài toán phân lớp dữ liệu văn bản mà còn trong nhiều ứng dụng như nhận dạng văn bản viết tay, phát hiện mặt người khung hình… Khả năng phân lớp của phương pháp SVM được đánh giá là khá cao so với các phương pháp khác.

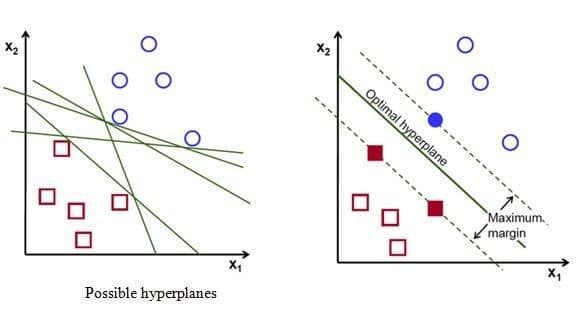
SVM sử dụng thuật toán học với mục tiêu tìm ra một mặt siêu phẳng làm nhỏ nhất có thể độ phân lớp sai cho một đối tượng dữ liệu mới. Độ phân lớp sai của một mặt siêu phẳng được đặc trưng bởi khoảng cách từ điểm gần nhất tới siêu phẳng đấy. Đặc trưng thiết yếu thể hiện khả năng của mô hình phân lớp chính là khả năng phân lớp những dữ liệu mới sau quá trình huấn luyện. Phương pháp huấn luyện sẽ được đánh giá là tốt nếu hiệu suất tổng quát hoá của mô hình phân lớp cao và ngược lại phương pháp huấn luyện sẽ được đánh giá là chưa tốt nếu hiệu suất tổng quát hóa của mô hình là thấp. Hiệu suất tổng quát hoá ở đây phụ thuộc vào hai yếu tố là năng lực của máy học và sai số huấn luyện. Trong các tham số này thì sai số huấn luyện được hiểu là tỷ lệ sai lỗi của quá trình phân lớp trên tập dữ liệu huấn luyện. Còn yếu tố thứ hai năng lực của máy học được xác định bằng kích thước VC (Vapnik-Chervonenkis). Đây được coi là một khái niệm quan trọng đối với một mô hình phân lớp. Kích thước VC được tính bởi số điểm cực đại mô hình phân lớp có thể tách trong không gian đối tượng cần phân loại.

Siêu phẳng tạo ra một biên giới phân chia 2 lớp của dữ liệu.



Hình 2.1: Mô phỏng phân lớp dựa trên SVM

Để phân chia 2 lớp dữ liệu, rõ ràng là có rất nhiều siêu phẳng có thể làm được điều này. Mặc dù vậy, mục tiêu của chúng ta là tìm ra siêu phẳng có lề rộng nhất, tức là có khoảng cách tới các điểm của 2 lớp là lớn nhất, ví dụ như hình dưới đây:

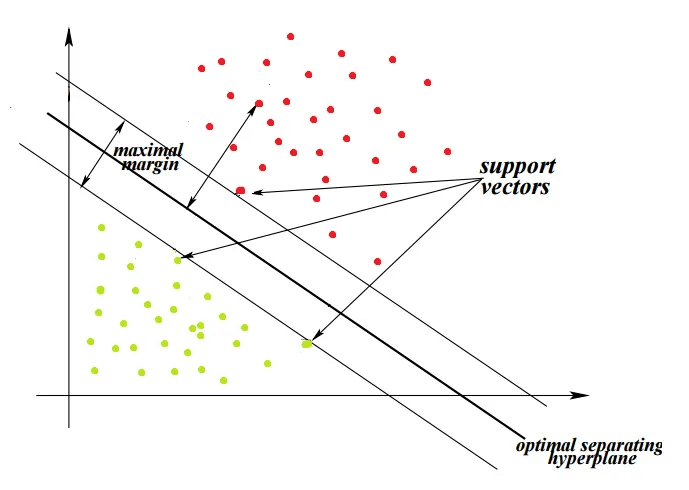


Hình 2.2: Siêu mặt phẳng phân lớp dữ liệu

Lưu ý: Số chiều của siêu phẳng phụ thuộc vào số đặc trưng.

## 2.1. Các vector hỗ trợ

Một điểm trong không gian vector có thể được coi là một vector từ gốc tọa độ tới điểm đó. Các điểm dữ liệu nằm trên hoặc gần nhất với siêu phẳng được gọi là vector hỗ trợ, chúng ảnh hưởng đến vị trí và hướng của siêu phẳng. Các vector này được sử dụng để tối ưu hóa lề và nếu xóa các điểm này, vị trí của siêu phẳng sẽ thay đổi. Một điểm lưu ý nữa đó là các vector hỗ trợ phải cách đều siêu phẳng.



Hình 2.3: Vector hỗ trợ siêu mặt phẳng

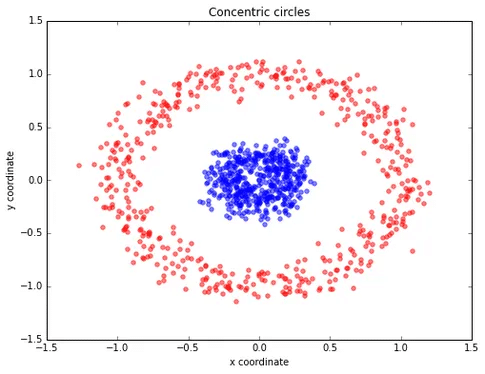
SVM chỉ có thể hoạt độgn trên dữ liệu có thể phân chia tuyến tính.

Nếu dữ liệu không thể phân chia tuyến tính thì sao?

**Ví dụ:** Nhìn vào hình ảnh bên dưới, dữ liệu được phân tách phi tuyến tính, rõ ràng ta không thể vẽ một đường thẳng để phân loại các điểm dữ liệu đỏ và xanh. Để giải quyết vấn đề này, có 2 giải pháp:

1. Lề mềm (Soft margin)

2. Thủ thuật Kernel (Kernel tricks)



Hình 2.4: Biên phân tách

Thuật toán này cho phép SVM mắc một số lỗi nhất định và giữ cho lề càng rộng càng tốt để các điểm khác vẫn có thể được phân loại chính xác. Nói một cách khác, nó cân bằng giữa việc phân loại sai và tối đa hóa lề.

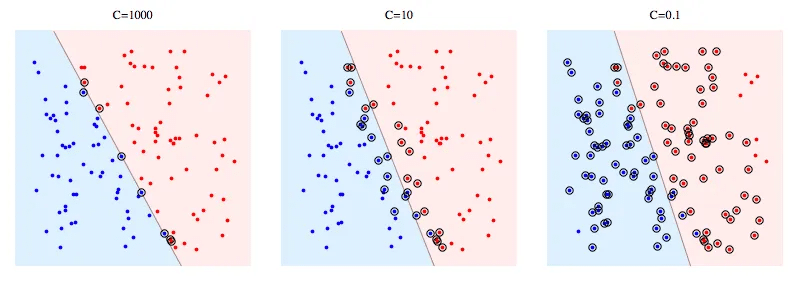
Có 2 kiểu phân loại sai có thể xảy ra:

1. Dữ liệu nằm ở đúng bên nhưng phạm vào lề

2. Dữ liệu nằm ở sai bên

Mức độ chấp nhận lỗi

Mức độ chấp nhận lỗi là một siêu tham số quan trọng trong SVM. Khi lập trình với sklearn, mức độ chấp nhận lỗi được coi như một tham số phạt (C). Hình dưới thể hiện SVM với [các giá trị C khác nhau](https://machinelearningcoban.com/2017/04/13/softmarginsmv/#-anh-huong-cua-\c\-len-nghiem).



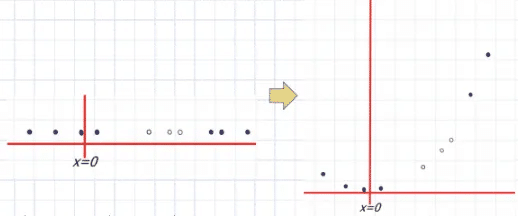
Hình 2.5: Các giá trị biên trong SVM

C càng lớn có nghĩa là SVM càng bị phạt nặng khi thực hiện phân loại sai. Do đó, lề càng hẹp và càng ít vector hỗ trợ được sử dụng.

[Thủ thuật Kernel](https://machinelearningcoban.com/2017/04/22/kernelsmv/)

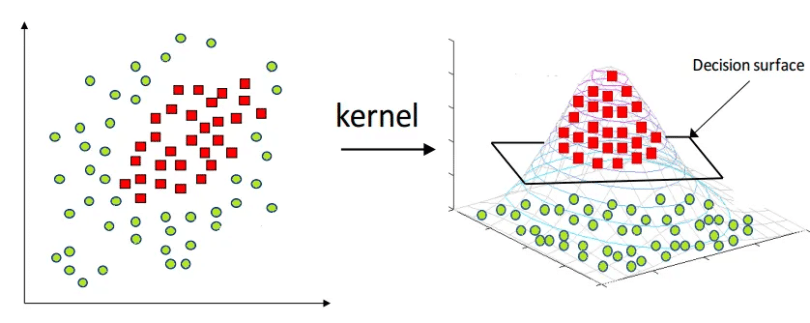
Một kernel là một hàm ánh xạ dữ liệu từ không gian ít nhiều hơn sang không gian nhiều chiều hơn, tự đó ta tìm được siêu phẳng phân tách dữ liệu. Một cách trực quan, kỹ thuật này giống như việc bạn gập tờ giấy lại để có thể dùng kéo cắt một lỗ tròn trên nó.

Biểu diễn trực quan của thủ thuật kernel:



Hình 2.6: Biểu diễn trực quan trên trục đồ thì

Ví dụ



Hình 2.7: Phân tách bằng đồ thị phức tạp

Các kiểu Kernel:

1. Tuyến tính

2. Đa thức

3. RBF

4. Sigmoid

## 2.2. Huấn luyện SVM

Để huấn luyện máy hỗ trợ vector thực chất chúng ta sẽ tiến hành giải bài toán quy hoạch toàn phương Support Vector Machine. Để giải bài toán này chúng ta có thể dùng các phương pháp số. Cụ thể các phương pháp này cần một ma trận kích thước bằng bình phương của số lượng mẫu dùng trong việc training. Điều này trong thực tế đôi khi không khả thi vì kích thước của tập dữ liệu dùng để training thường rất lớn (thậm chí có thể đến hàng chục nghìn mẫu huấn luyện). Nhằm giải quyết vấn đề nêu trên người ta đã phát triển nhiều thuật toán khác nhau dựa trên việc phân rã tập training thành những nhóm dữ liệu. Lúc này bài toán quy hoạch toàn phương sẽ được giải với kích thước nhỏ hơn. Sau đó, những thuật toán này sẽ kiểm tra các điều kiện Karush KuhnTucker để tìm ra được phương án tối ưu nhất. Một vài phương pháp huấn luyện dựa vào tính chất: Nếu trong tập training của bài toán quy hoạch toàn phương Support Vector Machine con (bài toán nhỏ) có ít nhất một mẫu vi phạm vào điều kiện Karush KuhnTucker, thì bài toán này sau khi được giải, hàm mục tiêu sẽ tăng. Một loạt các bài toán quy hoạch toàn phương Support Vector Machine con với ít một mẫu nào đó vi phạm các điều kiện Karush KuhnTucker được đảm bảo sẽ sự hội tụ đến phương án tối ưu.

**Ưu điểm của SVM đó là:**

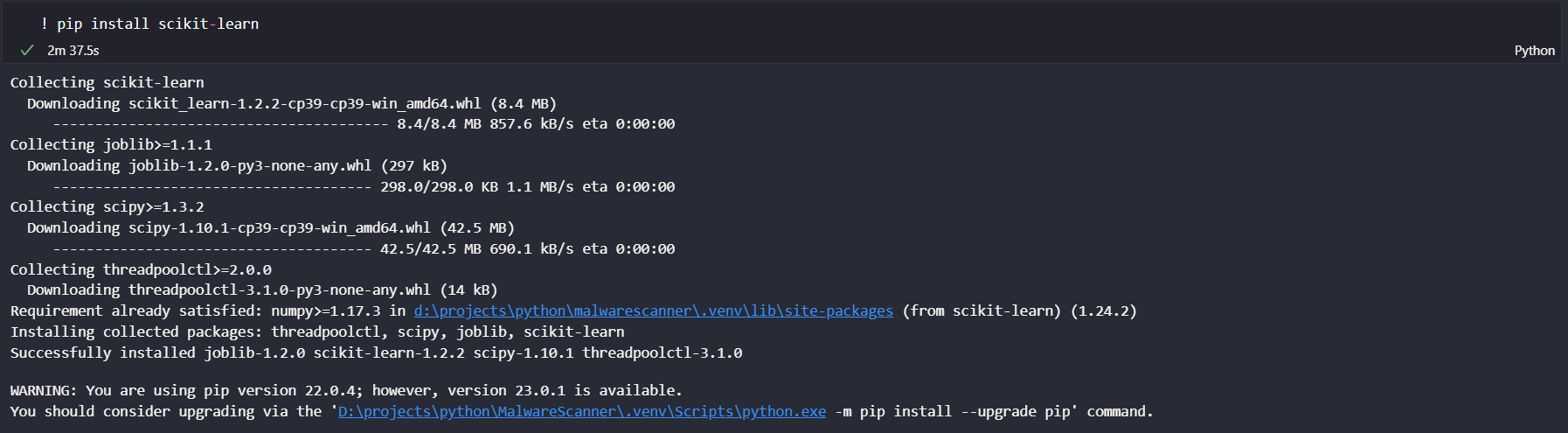
* Đây là thuật toán hoạt động hiệu quả với không gian cao chiều (high dimensional spaces).
* Thuật toán tiêu tốn ít bộ nhớ vì chỉ sử dụng các điểm trong tập hỗ trợ để dự báo trong hàm quyết định.
* Chúng ta có thể tạo ra nhiều hàm quyết định từ những hàm kernel khác nhau. Thậm chí sử dụng đúng kernel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

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.

2.3 Thực nghiệm

Dựa trên các đặc điểm trên, chúng ta sẽ áp dụng mô hình học máy SVM để chạy thực nghiệm việc phân loại mã độc thông qua hàm *sklearn.svm.SVC* của thư viện *scikit-learn*. Để có thể sử dụng được thư viện, ta tiến hành cài đặt thư viện *scikit-learn* thông qua câu lệnh *pip install scikit-learn.*



Hình 2.8: Cài đặt thư viện scikit-learn

Sau khi cài đặt xong, ta tiến hành nạp dữ liệu sử dụng để huấn luyện và kiểm tra từ file có định dạng .csv vào chương trình thông qua thư viện *pandas*.

import pandas as pd

import numpy as np

train\_df=pd.read\_csv(r'dataset/train-0.csv',header=None,skiprows=1)

val\_df=pd.read\_csv(r'dataset/file-0.csv',header=None,skiprows=1)

test\_df=pd.read\_csv(r'dataset/file-1.csv',header=None,skiprows=1)

train\_x = np.array(train\_df.iloc[:, 3:])

train\_y = np.array(train\_df.iloc[:, 2])

val\_x = np.array(val\_df.iloc[:, 3:])

val\_y = np.array(val\_df.iloc[:, 2])

test\_x = np.array(test\_df.iloc[:, 3:])

test\_y = np.array(test\_df.iloc[:, 2])

Ở đây, chúng ta sử dụng bộ dữ liệu được tách ra thành 3 file với 3 mục đích khác nhau, cụ thể là sử dụng để huấn luyện, xác thực và kiểm tra. Ta tiến hành tách các đặc trưng và nhãn từ dữ liệu đã được nạp vào rồi sau đó biến đổi chúng thành đối tượng có dạng dữ liệu kiểu mảng bằng thư viện *numpy.* Tiếp đó, ta nạp thư viện scikit-learn đã cài đặt vào trong chương trình, thực hiện huấn luyện cho mô hình với dữ liệu vừa nạp vào và tiến hành cho mô hình phân loại mã độc.

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

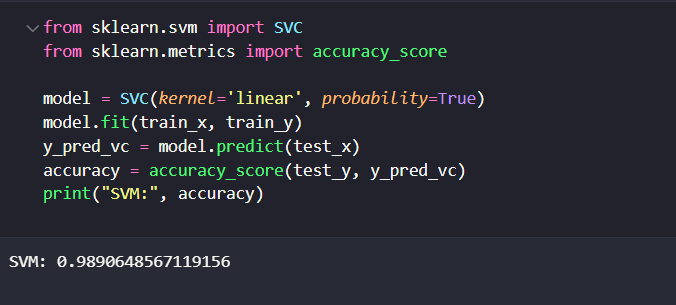
model = SVC(kernel='linear', probability=True)

model.fit(train\_x, train\_y)

y\_predict = model.predict(test\_x)

accuracy = accuracy\_score(test\_y, y\_pred\_vc)

Thuật toán Linear được sử dụng cho mô hình được truyền vào thông qua tham số *kernel*. Sau khi tiến hành huấn luyện và cho mô hình phân loại mã độc, ta sử dụng hàm sklearn.metrics.accuracy\_score để đánh giá độ chính xác kết quả dự đoán của mô hình với dữ liệu kiểm tra có sẵn. Kết quả thu được từ mô hình có độ chính xác đạt gần 0.98906 (~99%).



Hình 2.9: Kết quả độ chính xác của mô hình