**Trường Đại Học Xây Dựng Hà Nội**

**Khoa Công nghệ Thông tin**

**--o0o--**

Icon

Description automatically generated

**Báo cáo môn học:**

**CHUYÊN ĐỀ TỔNG HỢP**

**Tìm hiểu về thuật toán SVM và KNN**

**Áp dụng, triển khai, đánh giá trên 2 bộ dữ liệu Maintance và Material**

**Lớp: 66MHT1**

**Sinh viên: Vũ Hoàng Giang - 0188166**

**Giáo viên hướng dẫn : Lê Thị Thùy Dương**

**Hà Nội, 11/2024**

Mục lục

[**A.** **Lý Thuyết** 3](#_Toc183389633)

[**A.1.Support Vector Machine** 3](#_Toc183389634)

[**1.** **Giới thiệu** 3](#_Toc183389635)

[**2.** **Hard Margin SVM** 6](#_Toc183389636)

[**3.** **Soft Margin SVM** 9](#_Toc183389637)

[**4.** **Kernel SVM** 13](#_Toc183389638)

[**5.** **Multi-class Support Vector Machine** 18](#_Toc183389639)

[**A.2.K-nearest Neighbor** 29](#_Toc183389640)

[**1.** **Giới thiệu.** 29](#_Toc183389641)

[**2.** **K-Nearest Neighbor** 29](#_Toc183389642)

[**B.** **Bộ dữ liệu Maintance** 34](#_Toc183389643)

[**1.** **Giới thiệu** 34](#_Toc183389644)

[**2.** **Bộ dữ liệu maintance** 34](#_Toc183389645)

[**3.** **Cân bằng dữ liệu SMOTE - Synthetic Minority Over-sampling Technique** 35](#_Toc183389646)

[**4.** **Chuẩn hóa Minmax** 36](#_Toc183389647)

[**5.** **Mối tương quan giữa các đặc trưng** 36](#_Toc183389648)

[**6.** **Triển khai trên bộ dữ liệu** 37](#_Toc183389649)

[**7.** **Phân tích hiệu suất của các thuật toán** 39](#_Toc183389650)

[**i.** **Support Vector Classification (SVC)** 40](#_Toc183389651)

[**ii.** **K-nearest Neighbor (KNN)** 41](#_Toc183389652)

[**8.** **Phân tích độ nhạy của các thuật toán được đề xuất** 42](#_Toc183389653)

[**9.** **Thảo luận** 43](#_Toc183389654)

[**C.** **Bộ dữ liệu Material** 44](#_Toc183389655)

[**1.** **Giới thiệu** 44](#_Toc183389656)

[**2.** **Bộ dữ liệu material** 44](#_Toc183389657)

[**3.** **Tiền xử lý dữ liệu** 44](#_Toc183389658)

[**4.** **Cân bằng dữ liệu SMOTE - Synthetic Minority Over-sampling Technique** 45](#_Toc183389659)

[**5.** **Chuẩn hóa Minmax** 46](#_Toc183389660)

[**6.** **Mối tương quan giữa các đặc trưng** 46](#_Toc183389661)

[**7.** **Triển khai trên tập dữ liệu** 47](#_Toc183389662)

[**8.** **Phân tích hiệu suất của các thuật toán** 50](#_Toc183389663)

[**a.** **Support Vector Classification - SVC** 51](#_Toc183389664)

[**b.** **K-nearest Neighbor - KNN** 52](#_Toc183389665)

[**9.** **Thảo luận** 53](#_Toc183389666)

1. **Lý Thuyết**

## **A.1.Support Vector Machine**

1. **Giới thiệu**

* Máy vectơ hỗ trợ (Support vector machine - SVM) được đề cử bởi V. Vapnik và các đồng nghiệp của ông vào những năm 1970s ở Nga, và sau đó đã trở nên nổi tiếng và phổ biến vào những năm 1990s
* SVM là một phương pháp phân lớp tuyến tính (linear classifier), với mục đích xác định một siêu phẳng (hyperplane) để phân tách hai lớp của dữ liệu – ví dụ: lớp các ví dụ có nhãn dương (positive) và lớp các ví dụ có nhãn âm (negative)
* Các hàm nhân (kernel functions), cũng được gọi là các hàm biến đổi (transformation functions), được dùng cho các trường hợp phân lớp phi tuyến.
* SVM có một nền tảng lý thuyết chặt chẽ – dựa trên nhiều định lý toán học
* SVM là một phương pháp tốt (phù hợp) đối với những bài toán phân lớp có không gian biểu diễn thuộc tính lớn– các đối tượng cần phân lớp được biểu diễn bởi một tập rất lớn các thuộc tính
* SVM đã được biết đến là một trong số các phương pháp phân lớp tốt nhất đối với các bài toán phân lớp văn bản (text/document classification)
* *Ví dụ:*

Biểu diễn tập r các ví dụ huấn luyện (training examples) :

{(

* là một vectơ đầu vào được biểu diễn trong không gian

X ⊆ Rn

* yi là một nhãn lớp (giá trị đầu ra), yi ∈ {1,−1}
  + yi = 1: lớp dương (positive);
  + yi = −1: lớp âm (negative)

Đối với một ví dụ :

SVM xác định một hàm phân tách tuyến tính:

* + w là vectơ trọng số các thuộc tính
  + b là một giá trị số thực
* **Một số khái niệm**
  1. Khoảng cách từ một điểm đến một siêu mặt phẳng

Trong không gian 2 chiều, ta biết rằng khoảng cách từ một điểm có toạ độ tới đường thẳng có phương trình được xác định bởi:

A math equations with black text

Description automatically generated with medium confidence

Trong không gian 3 chiều, ta biết rằng khoảng cách từ một điểm có toạ độ tới đường thẳng có phương trình được xác định bởi:

A black and white math equation

Description automatically generated with medium confidence

Hơn nữa, nếu ta bỏ dấu trị tuyệt đối ở tử số, chúng ta có thể xác định được điểm đó nằm về phía nào của *đường thẳng* hay *mặt phẳng* đang xét. Những điểm làm cho biểu thức trong dấu giá trị tuyệt đối mang dấu dương nằm về cùng 1 phía (tôi tạm gọi đây là *phía dương* của đường thẳng), những điểm làm cho biểu thức trong dấu giá trị tuyệt đối mang dấu âm nằm về phía còn lại (tôi gọi là *phía âm*). Những điểm nằm trên *đường thẳng*/*măt phẳng* sẽ làm cho tử số có giá trị bằng 0, tức khoảng cách bằng 0.  
**Việc này có thể được tổng quát lên không gian nhiều chiều:**

Khoảng cách từ một điểm (vector) có toạ độ tới *siêu mặt phẳng* (*hyperplane*) có phương trình được xác định bởi:

A black text on a white background

Description automatically generated

Với với d là số chiều của không gian.

* 1. Mặt siêu phẳng phân cách
     + Mặt siêu phẳng phân tách các ví dụ huấn luyện lớp dương và các ví dụ huấn luyện lớp âm:

A diagram of a mathematical equation

Description automatically generated

* + - Còn được gọi là ranh giới (bề mặt) quyết định
    - Tồn tại nhiều mặt siêu phẳng phân tách

A diagram of a graph

Description automatically generated with medium confidence

* 1. Mặt siêu phẳng có lề cực đại
* SVM lựa chọn mặt siêu phẳng phân tách có lề (margin) lớn nhất

A diagram of mathematical equations

Description automatically generated

(1)

* *Lý thuyết học máy đã chỉ ra rằng một mặt siêu phẳng phân tách như thế sẽ tối thiểu hóa giới hạn lỗi (phân lớp) mắc phải*

1. **Hard Margin SVM**
   1. SVM – Dữ liệu phân tách được tuyến tính

* Giả sử rằng tập dữ liệu (tập các ví dụ huấn luyện) có thể phân tách được một cách tuyến tính.
* Xét một ví dụ của lớp dương (1) và một ví dụ của lớp âm (,-1) gần nhất đối với siêu phẳng phân tách
* Định nghĩa 2 siêu phẳng lề song song với nhau

* + đi qua và song song với

* + đi qua  và song song với

Sao cho:

* + - nếu
    - nếu
  1. Tính toán mức lề
* Mức lề (margin) là khoảng cách giữa 2 siêu phẳng lề H+ và H-. Trong hình vẽ (1) :
* là khoảng cách từ và
* là khoảng cách từ và
* ( là mức lề
* Theo lý thuyết đại số vectơ, khoảng cách (trực giao) từ một điểm đến mặt siêu phẳng là :

A math equation with a plus and b

Description automatically generated

Trong đó ||w|| là độ dài của w :



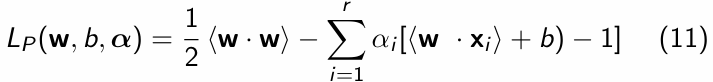
* Tính toán :
  + Áp dụng Eq.3-4: A black and white math equation

    Description automatically generated with medium confidence
* Tính toán :
  + Áp dụng Eq.3-4: 
* Tính toán mức lề (:

A math equation with numbers and symbols

Description automatically generated

* 1. Giải bài toán cực tiểu hóa có ràng buộc
* Biểu thức Lagrange :



Trong đó (là **các hệ số nhân Lagrange**

* Lý thuyết tối ưu chỉ ra rằng một lời giải tối ưu cho (11) phải thỏa mãn các điều kiện nhất định, được gọi là các **điều kiện Karush-Kuhn-Tucker** là các điều kiện cần (nhưng không phải là các điều kiện đủ)
* Tập điều kiện Karush-Kuhn-Tucker :

A math equations with numbers and symbols

Description automatically generated

* + (14) chính là tập các ràng buộc ban đầu
  + Điều kiện bổ sung (16) chỉ ra rằng chỉ những ví dụ (điểm dữ liệu) thuộc các mặt siêu phẳng lề (mới có >0 - bởi vì với những ví đụ đó thì 
* Những ví dụ (điểm dữ liệu) này được gọi là **các vectơ hỗ trợ!**
  + Đối với các ví dụ khác (không phải là các vectơ hỗ trợ) thì =0
* *Trong trường hợp tổng quát, các điều kiện Karush-KuhnTucker là cần đối với một lời giải tối ưu, nhưng chưa đủ*
* *Tuy nhiên, đối với bài toán cực tiểu hóa đang xét có hàm mục tiêu lồi (convex) và các ràng buộc tuyến tính, thì các điều kiện Karush-Kuhn-Tucker là cần và đủ đối với một lời giải tối ưu*
* **Giải quyết bài toán tối ưu này vẫn là một nhiệm vụ khó khăn – do sự tồn tại của các ràng buộc bất đẳng thức!**
* Phương pháp Lagrange giải quyết bài toán tối ưu hàm lồi dẫn đến một biểu thức **đối ngẫu (dual)** của bài toán tối ưu
  1. Biểu thức đối ngẫu
* Để thu được biểu thức đối ngẫu từ biểu thức ban đầu:
* Gán giá trị bằng 0 đối với các đạo hàm bộ phận của biểu thức Lagrange trong (11) đối với **các biến ban đầu** (**w** và b)
* Sau đó, áp dụng các quan hệ thu được đối với biểu thức Lagrange
  + Áp dụng các biểu thức (12-13) vào biểu thức Lagrange ban đầu (11) để loại bỏ **các biến ban đầu** (**w** và b)
* Biểu thức đối ngẫu :

A black and white math symbol

Description automatically generated

* Cả hai biểu thức (11) và (17) đều là các biểu thức Lagrange
  + Dựa trên cùng một hàm mục tiêu– nhưng với các ràng buộc khác nhau
  + Lời giải tìm được, bằng **cách cực tiểu hóa hoặc cực đại hóa**
* **Bài toán đối ngẫu:**

**A number and mathematical symbols

Description automatically generated with medium confidenceCực đại hóa:  (18)**

**Với điều kiện**

* Giải quyết biểu thức (18), ta thu được các hệ số nhân Lagrange (các hệ số này sẽ được dùng để tính w và b)
  1. Tính toán các giá trị  **và**
* Gọi SV là tập các vectơ hỗ trợ
  + SV là tập con của tập r các ví dụ huấn luyện ban đầu
    - với các vecto hỗ trợ
    - với các vecto không phải vecto hỗ trợ
* Sử dụng biểu thức (12), ta có thể tính được giá trị **:**

A black text with black letters

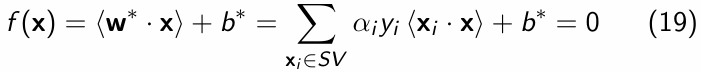
Description automatically generated

* Sử dụng biểu thức (16) và (bất kỳ) một vectơ hỗ trợ, ta có:



Vì với các vecto hỗ trợ

* 
* 
  1. Ranh giới quyết định phân lớp
  + **Ranh giới quyết định phân lớp** được xác định bởi siêu phẳng:



* Đối với một ví dụ cần phân lớp **z**, cần tính giá trị:



* **Nếu biểu thức (20) trả về giá trị 1, thì ví dụ z được phân vào lớp có nhãn dương (positive); ngược lại, được phân vào lớp có nhãn âm (negative)**
* *Việc phân lớp này:* 
  + *Chỉ phụ thuộc vào các vectơ hỗ trợ*
  + *Chỉ cần giá trị tích vô hướng (tích trong) của 2 vectơ (chứ không cần biết giá trị của 2 vectơ đấy)*
  1. Tóm tắt
* Với bài toán binary classification mà 2 classes là linearly separable, có vô số các siêu mặt phẳng giúp phân biệt hai classes, tức mặt phân cách. Với mỗi mặt phân cách, ta có một classifier. Khoảng cách gần nhất từ 1 điểm dữ liệu tới mặt phân cách ấy được gọi là margin của classifier đó.
* Support Vector Machine là bài toán đi tìm mặt phân cách sao cho margin tìm được là lớn nhất, đồng nghĩa với việc các điểm dữ liệu an toàn nhất so với mặt phân cách.
* Bài toán tối ưu trong SVM là một bài toán lồi với hàm mục tiêu là stricly convex, nghiệm của bài toán này là duy nhất. Hơn nữa, bài toán tối ưu đó là một Quadratic Programming (QP).
* Mặc dù có thể trực tiếp giải SVM qua bài toán tối ưu gốc này, thông thường người ta thường giải bài toán đối ngẫu. Bài toán đối ngẫu cũng là một QP nhưng nghiệm là sparse nên có những phương pháp giải hiệu quả hơn.

1. **Soft Margin SVM**
   1. Linear SVM: Không phân tách được

* Phương pháp SVM trong trường hợp các ví dụ phân lớp tuyến tính nhưng không thể phân tách được:
  + Trường hợp phân lớp tuyến tính và phân tách được là lý tưởng (ít xảy ra)
  + Tập dữ liệu có thể chứa nhiễu, lỗi
* Hai ví dụ nhiều vàđược gán nhãn lớp sai

A diagram of mathematical equations

Description automatically generated

* 1. Nới lỏng các điều kiện
  + Để làm việc với các dữ liệu chứa nhiễu, cần nới lỏng các điều kiện lề (margin constraints) bằng cách sử dụng các biến **slack**  **0** 
    - nếu
    - nếu
  + Với một ví dụ nhiễu/lỗi :
  + là giới hạn trên của lỗi của các ví dụ huấn luyện
  + Các điều kiện mới đối với trường hợp (phân lớp tuyến tính) không thể phân tách được:

A math equations on a white background

Description automatically generated

* **Tích hợp lỗi vào hàm mục tiêu**
  + Bằng cách gán giá trị chi phí (cost) cho các lỗi, và tích hợp chi phí này trong hàm mục tiêu mới:

Cực tiểu hóa : 

Trong đó C(>0) là tham số xác định **mức độ chi phí (penalty degree)** đối với các lỗi

* C càng lớn, mức độ chi phí càng cao với các lỗi
* k=1 thường đc sử dụng
  1. Bài toán tối ưu mới:

Cực tiểu hóa: A black symbols on a white background

Description automatically generated

Với điều kiện: A number and symbol on a white background

Description automatically generated

* Soft-margin SVM
* Biểu thức tối ưu Lagrange :

A black and white math equation

Description automatically generated with medium confidence

Trong đó ((là **các hệ số nhân Lagrange**

* Tập điều kiện Karush-Kuhn-Tucker

A math equations with numbers

Description automatically generated with medium confidence

A math equations with numbers and symbols

Description automatically generated

* 1. Biểu thức đối ngẫu
* Chuyển biểu thức Lagrange từ dạng ban đầu (primal formulation) về dạng đối ngẫu (dual formulation)
  + Gán giá trị bằng 0 cho các đạo hàm bộ phận của biểu thức Lagrange (22) đối với các biến ban đầu (w, b, **)**
  + Thay thế các kết quả thu được vào biểu thức Lagrange ban đầu

→ Sử dụng các kết quả của các biểu thức (23-25) để thay thế vào trong biểu thức Lagrange ban đầu (22)

* Từ biểu thức (25), ta có 

Vì ( nên suy ra điều kiện

* **Biểu thức đối ngẫu**

Cực đại hóa : A black and white text

Description automatically generated

Với điều kiện : A number of numbers and symbols

Description automatically generated with medium confidence

Ta thấy **và các hệ số nhân Lagrange của chúng**  không xuất hiện trong biểu thức đối ngẫu

* Hàm mục tiêu giống hệt như đối với bài toán phân lớp tuyến tính phân tách được

(Linear SVM)

* Khác biệt duy nhất là tập các ràng buộc mới
* **Bài toán đối ngẫu (32) được giải quyết bằng các phương pháp số học, ta được các giá trị (hệ số nhân Lagrange)** 
  1. Tính toán các giá trị  **và**
* **Các giá trị (hệ số nhân Lagrange) αi lời giải được sử dụng để tính toán và** 
  + **Chưa biết !**
* Để tính được
  + Từ (25) và (31), ta suy ra được: =0 nếu
  + Vì vậy, ta có thể sử dụng một ví dụ học thỏa mãn điều kiện (0 < < C) ( là một vecto hỗ trợ!) và (30) với : =0 để tính
* Việc tính toán b\* tương tự như với trường hợp phân lớp tuyến tính phân tách được!
  1. Các đặc điểm quan trọng
* Từ các biểu thức (25-31), ta có thể suy ra các kết luận sau:

A group of math symbols

Description automatically generated

* Biểu thức (33) thể hiện một đặc điểm rất quan trọng của SVM
  + Lời giải được xác định dựa trên rất ít (sparse) các giá trị
    - Rất nhiều ví dụ học nằm ngoài khoảng lề (margin area), và chúng có giá trị bằng 0
    - Các ví dụ **nằm trên lề**  *-* là các vecto hỗ trợ, thì có giá trị khác không (0 < < C)
    - Các ví dụ **nằm trong khoảng lề**  *-* là các ví dụ lỗi/nhiễu thì có giá trị khác không (=C)
  + Nếu không có đặc điểm thưa thớt (sparsity) này, thì phương pháp SVM không thể hiệu quả đối với các tập dữ liệu lớn
  1. Ranh giới quyết định phân lớp
* Ranh giới quyết định phân lớp chính là siêu phẳng:

A black and white math symbol

Description automatically generated

* + Rất nhiều ví dụ học có giá trị = 0 (chính là đặc điểm thưa thớt– sparsity– của phương pháp SVM)
* Đối với một ví dụ cần phân loại **z**, nó được phân loại bởi: 
* Cần xác định giá trị phù hợp của tham số C (trong hàm tối ưu mục tiêu)
  + Thường được xác định bằng cách sử dụng một tập tối ưu (validation set)
  1. Tóm tắt
* SVM thuần (Hard Margin SVM) hoạt động không hiệu quả khi có nhiễu ở gần biên hoặc thậm chí khi dữ liệu giữa hai lớp gần linearly separable. Soft Margin SVM có thể giúp khắc phục điểm này.
* Trong Soft Margin SVM, chúng ta chấp nhận lỗi xảy ra ở một vài điểm dữ liệu. Lỗi này được xác định bằng khoảng cách từ điểm đó tới đường biên tương ứng. Bài toán tối ưu sẽ tối thiểu lỗi này bằng cách sử dụng thêm các biến được gọi là slack varaibles.
* Để giải bài toán tối ưu, có hai cách khác nhau. Mỗi cách có những ưu, nhược điểm riêng, các bạn sẽ thấy trong các bài tới.
* Cách thứ nhất là giải bài toán đối ngẫu. Bài toán đối ngẫu của Soft Margin SVM rất giống với bài toán đối ngẫu của Hard Margin SVM, chỉ khác ở ràng buộc chặn trên của các nhân tử Laggrange. Ràng buộc này còn được gọi là box costraint.
* Cách thứ hai là đưa bài toán về dạng không ràng buộc dựa trên một hàm mới gọi là hinge loss. Với cách này, hàm mất mát thu được là một hàm lồi và có thể giải được khá dễ dàng và hiệu quả bằng các phương pháp Gradient Descent.
* Trong Soft Margin SVM, có một hằng số phải được chọn, đó là**C**. Hướng tiếp cận này còn được gọi là C-SVM. Ngoài ra, còn có một hướng tiếp cận khác cũng hay được sử dụng, gọi là ν-SVM, bạn đọc có thể đọc thêm [tại đây](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.2928&rep=rep1&type=pdf).

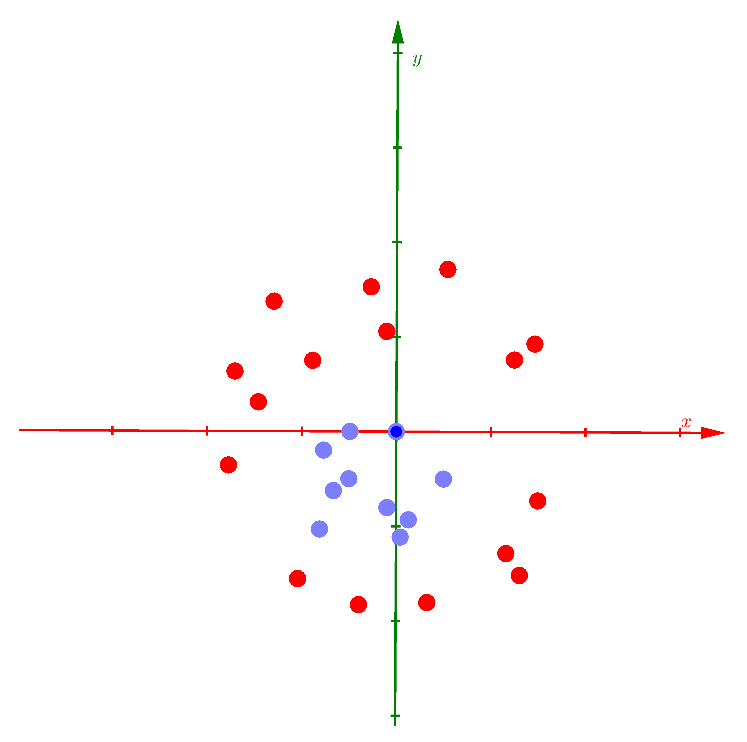
1. **Kernel SVM**
   1. Giới thiệu

Kernel SVM, tức việc áp dụng SVM lên bài toán mà dữ liệu giữa hai classes là hoàn toàn *không linear separable (không phân biệt tuyến tính)*

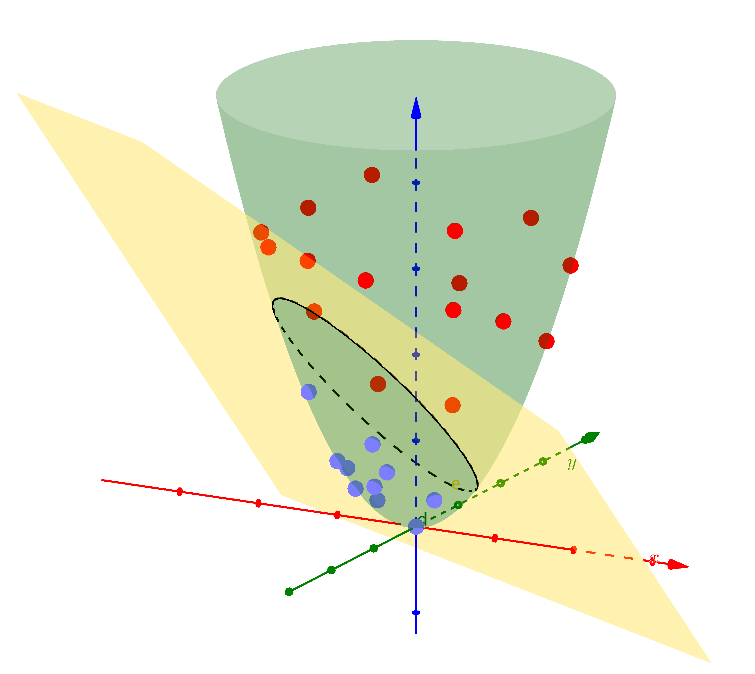
Ý tưởng cơ bản của Kernel SVM và các phương pháp kernel nói chung là tìm một phép biến đổi sao cho dữ liệu ban đầu là *không phân biệt tuyến tính* được biến sang không gian mới. Ở không gian mới này, dữ liệu trở nên *phân biệt tuyến tính*.

Ví dụ:

Dưới đây với việc biến dữ liệu *không phân biệt tuyến tính* trong không gian hai chiều thành *phân biệt tuyến tính* trong không gian ba chiều bằng cách giới thiệu thêm một chiều mới



1. Dữ liệu của hai classes là *không phân biệt tuyến tính* trong không gian hai chiều.



1. Nếu coi thêm chiều thứ ba là một hàm số của hai chiều còn lại  các điểm dữ liệu sẽ được phân bố trên 1 parabolic và đã trở nên *phân biệt tuyến tính*. Mặt phẳng màu vàng là mặt phân chia, có thể tìm được bởi Hard/Soft Margin SVM.

A diagram of a graph

Description automatically generated

1. Giao điểm của mặt phẳng tìm được và mặt parabolic là một đường ellipse, khi chiếu toàn bộ dữ liệu cũng như đường ellipse này xuống không gian hai chiều ban đầu, ta đã tìm được đường phân chia hai classses.

* Kernel SVM là việc đi tìm một hàm số biến đổi dữ liệu**x** từ không gian *feature* ban đầu thành dữ liệu trong một không gian mới bằng hàm số **Φ(x)** *(Hàm số này cần thỏa mãn mục đích của chúng ta: trong không gian mới, dữ liệu giữa hai classes là phân biệt tuyến tính hoặc gần như phần biệt tuyến tính.)*
* Các hàm **Φ()** thường tạo ra dữ liệu mới có số chiều cao hơn số chiều của dữ liệu ban đầu, thậm chí là vô hạn chiều.
  1. Cơ sở toán học
* Bài toán đối ngẫu trong Soft Margin SVM cho dữ liệu *gần phân biệt tuyến tính (có nhiễu/lỗi)*:

A math equations and formulas

Description automatically generated with medium confidence

Trong đó:

A screenshot of a computer

Description automatically generated

Sau khi giải được λ cho bài toán (1), *nhãn* của một điểm dữ liệu mới sẽ được xác định bởi dấu của biểu thức:

A mathematical equation with numbers

Description automatically generated with medium confidence

Trong đó:

A white background with black text

Description automatically generated

*Với dữ liệu thực tế, rất khó có dữ liệu gần như phân biệt tuyến tính, nghiệm của (1) có thể không thực sự tạo ra một bộ phân lớp tốt.*

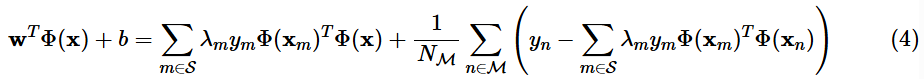
* Giả sử rằng ta có thể tìm được hàm số **Φ()** sao cho sau khi được biến đổi sang không gian mới, mỗi điểm dữ liệu xx trở thành **Φ(x)**, và trong không gian mới này, dữ liệu trở nên *gần phân biệt tuyến tính*.

Trong không gian mới, (1) có dạng:

A black and white math equations

Description automatically generated with medium confidence

Và nhãn của một điểm dữ liệu mới được xác định bởi dấu của biểu thức:



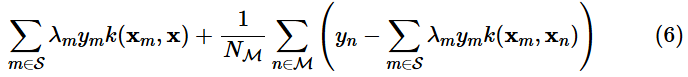
Trong bài toán (3) và biểu thức (4), chúng ta không cần tính trực tiếp **Φ(x)** cho mọi điểm dữ liệu. Chúng ta chỉ cần tính được  dựa trên hai điểm dữ liệu **x, z** bất kì! **(kernel trick)**

* *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, được gọi chung là****kernel method****.*
* Bằng cách định nghĩa hàm kernel , ta có thể viết lại (3), (4) như sau
* (3) có dạng:

A math equations on a white background

Description automatically generated

* (4) có dạng:



* Ví dụ

A math equations on a white background

Description automatically generated

* 1. Hàm số kernel (kernel method)
     1. Tính chất
* Các hàm kerrnel cần có các tính chất:
  + Đối xứng: k(x,z)=k(z,x)
  + *Về lý thuyết*, hàm kerrnel cần thỏa mãn [điều kiện Mercer](https://en.wikipedia.org/wiki/Mercer%27s_theorem#Mercer.27s_condition) (đảm bảo hàm mục tiêu của (5) là hàm lồi):
  + A close up of a sign

    Description automatically generated with medium confidence
  + *Trong thực hành, có một vài hàm số***k()***không thỏa mãn điều kiện Merrcer nhưng vẫn cho kết quả chấp nhận được. Những hàm số này vẫn được gọi là kernel method.*
* Nếu một hàm kerrnel thỏa mãn điều kiện (7), xét , ta có:

A black text with black letters

Description automatically generated

Với **K** là một ma trận đối xứng mà phần tử ở hàng thứ nn cột thứ mm của nó được định nghĩa bởi: 

Từ (8) ta suy ra **K** là một ma trận nửa xác định dương.

* Bài toán tối ưu (5) có ràng buộc là lồi và hàm mục tiêu là một hàm lồi (một quadratic form)
  + 1. Một số hàm kernel thông dụng
       1. Linear

Đây là trường hợp đơn giản với kernel chính tích vô hướng của hai vector:



* + - 1. Polynomial



Với **d** là một số dương để chỉ bậc của đa thức. dd có thể không là số tự nhiên vì mục đích chính của ta không phải là bậc của đa thức mà là cách tính kernel.

* *Polynomial kernel có thể dùng để mô tả hầu hết các đa thức có bậc không vượt quá dd nếu***d***là một số tự nhiên.*
  + - 1. Radial Basic Function

Radial Basic Function (RBF) kernel hay Gaussian kernel được sử dụng nhiều nhất trong thực tế, và là lựa chọn mặc định trong sklearn. Nó được định nghĩa bởi:



* + - 1. Sigmoid

[Sigmoid function](https://machinelearningcoban.com/2017/01/27/logisticregression/#sigmoid-function) cũng được sử dụng làm kernel:



* + - 1. Bảng tóm tắt các kernel thông dụng

A white sheet with red text

Description automatically generated

* + - 1. Tự định nghĩa

**Ngoài các hàm kernel thông dụng như trên, chúng ta cũng có thể tự định nghĩa các kernel của mình.**

* 1. Tóm tắt
* Nếu dữ liệu của hai lớp là không phân biệt tuyến tính, chúng ta có thể tìm cách biến đổi dữ liệu sang một không gian mới sao cho trong không gian mới ấy, dữ liệu của hai lớp là phân biệt tuyến tính hoặc gần phân biệt tuyến tính.
* Việc tính toán trực tiếp hàm **Φ()** đôi khi phức tạp và tốn nhiều bộ nhớ. Thay vào đó, ta có thể sử dụng **kernel trick**. Trong cách tiếp cận này, ta chỉ cần tính tích vô hướng của hai vector bất kỳ trong không gian mới: 
* Thông thường, các hàm **k()** thỏa mãn điều kiện Merrcer, và được gọi là kernel. Cách giải bài toán SVM với kernel hoàn toàn giống với cách giải bài toán Soft Margin SVM.
* Có 4 loại kernel thông dụng: linear, poly, rbf, sigmoid. Trong đó, rbf được sử dụng nhiều nhất và là lựa chọn mặc định trong các thư viện SVM.
* Với dữ liệu gần phân biệt tuyến tính, linear và poly kernels cho kết quả tốt hơn.

1. **Multi-class Support Vector Machine**
   1. **Giới thiệu**

* Các phương pháp Support Vector Machine đã đề cập (Hard Margin, Soft Margin, Kernel) đều được xây dựng nhằm giải quyết bài toán [Binary Classification](https://machinelearningcoban.com/2017/02/11/binaryclassifiers/), tức bài toán phân lớp với chỉ hai classes.
* Một cách tự nhiên để mở rộng các mô hình này áp dụng cho các bài toán multi-class classification, tức có nhiều classes dữ liệu khác nhau, là [sử dụng nhiều binary classifiers và các kỹ thuật như one-vs-one hoặc one-vs-rest](https://machinelearningcoban.com/2017/02/11/binaryclassifiers/#-binary-classifiers-cho-multi-class-classification-problems).
* Mô hình end-to-end
  + Softmax Regression
  + Convolutional Neural Networks
  + Sự hiệu quả của Softmax Regression nói riêng và Convolutional Neural Networks nói chung là cả *bộ trích chọn đặc trưng* (feature extractor) và *bộ phân lớp* (classifier) được *huấn luyện* đồng thời.
* Những mô hình như thế này còn được gọi là *end-to-end*
* Bias-trick

Thông thường, với một ma trận hệ số  một đầu vào  và vector bias  , ta có thể tính được đầu ra của layer này là:



* 1. Cơ sở toán học
     1. Xây dựng hàm mất mát cho Multi-class Support Vector Machine
        1. Softmax Regression

A diagram of a network

Description automatically generated

* Qua ma trận hệ số **W**, dữ liệu ban đầu trở thành 
* Lúc này, ứng với mỗi một trong **C** classes, chúng ta nhận được một giá trị tương ứng  ứng với class thứ **i**. Giá trị  này còn được gọi là *score* của dữ liệu **x** ứng với class thứ **i**.
* Ý tưởng chính trong **Softmax Regression** là đi tìm ma trận hệ số **W**, mỗi cột của ma trận này ứng với một class, sao cho *score vector* **z** đạt giá trị lớn nhất tại phần tử tương ứng với class chính xác của nó. Sau khi mô hình đã được *trained*, *nhãn* của một điểm dữ liệu mới được tính là vị trí của thành phần score có giá trị lớn nhất trong *score vector.*
* Để huấn luyện trên tập các cặp (*dữ liệu*, *nhãn*), Softmax Regression sử dụng hàm softmax để đưa *score vector* về dạng phân phối xác suất có các phần tử là dương và có tổng bằng 1. Sau đó dùng hàm cross entropy để *ép* vector xác suất này gần với vector xác suất *thật sự* của dữ liệu - tức one-hot vector mà chỉ có đúng 1 phần tử bằng 1 tại class tương ứng, các phần tử còn lại bằng 0.
  + - 1. Hinge loss tổng quát cho Multi-class SVM (hàm mất mát)
* Với Multi-class SVM, trong khi test, class của một input cũng được xác định bởi thành phần có giá trị lớn nhất trong score vector.
* Multi-class SVM xây dựng hàm mất mát dựa trên định nghĩa của *biên an toàn*, giống như trong Hard/Soft Margin vậy.
* Multi-class SVM *muốn* thành phần ứng với *correct class* của *score vector* lớn hơn các phần tử khác, không những thế, nó còn lớn hơn một đại lượng **Δ>0** gọi là *biên an toàn*. A diagram of a line with text

  Description automatically generated with medium confidence
* Giả sử rằng các thành phần của score vector được đánh số thứ tự từ 1. Các classes cũng được đánh số thứ tự từ 1. Giả sử rằng điểm dữ liệu **x** đang xét thuộc class **y** và score vector của nó là vector **z**.
* Thế thì score của *correct class* là , scores của các classes khác là các , **i≠y.** Xét ví dụ như trong Hình 6 với hai score  trong vùng an toàn và  trong vùng vi phạm.
  + Với mỗi score   trong vùng an toàn, loss bằng 0.
  + Với mỗi score  vượt quá điểm an toàn (điểm x đỏ), *loss* do nó gây ra được tính bằng lượng vượt quá so với điểm x đỏ, đại lượng này có thể tính được là:



Tóm lại, với một score , **j≠y,**  *loss* do nó gây ra có thể được viết gọn thành:



Trong đó: là *cột* thứ**j** của ma trận hệ số **W.**

Như vậy, với một điểm dữ liệu (n=1, 2…, N), tổng cộng loss do nó gây ra là:

A black and white text

Description automatically generated

Trong đó  là scores tương ứng với điểm dữ liệu ;  là *correct class* của điểm dữ liệu đó. A math equations on a white background

Description automatically generated

* + - 1. Regularization

Nếu nghiệm tìm được **w** là *hoàn hảo* tức không có score nào *vi phạm* và biểu thức (2) đạt giá trị 0: 

Điều này có nghĩa là **kW** cũng là một nghiệm của bài toán với k>1 bất kỳ. -> vô số nghiệm

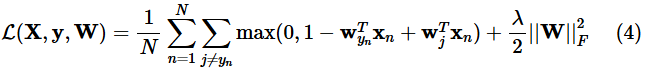
Một phương pháp quen thuộc để tránh hiện tượng này là cộng thêm số hạng [*regularization*](https://machinelearningcoban.com/2017/03/04/overfitting/#-regularization) vào hàm mất mát.

A black text on a white background

Description automatically generated

Trong đó  là [Frobenius norm](https://machinelearningcoban.com/math/#chuan-cua-ma-tran), và **λ** là một giá trị dương giúp cân bằng giữa *data loss* và *regularization loss*, thường được chọn bằng [cross-validation](https://machinelearningcoban.com/2017/03/04/overfitting/#-cross-validation).

* + - 1. Chọn giá trị **Δ**
* Có hai *hyperparameter* trong hàm mất mát (3) là **Δ** và **λ.**
* *Trong thực tế, người ta nhận thấy rằng ΔΔ có thể được chọn bằng 1 mà không ảnh hưởng nhiều tới chất lượng của nghiệm.*
* Hạn chế việc **W**trở nên quá lớn. Việc này đã được điều chỉnh bởi tham số **λ**.
* Cuối cùng, chúng ta sẽ đi tối ưu hàm mất mát sau đây cho Multi-class SVM:



* + - 1. Nhận xét
* ***Soft-margin SVM là một trường hợp đặc biệt của Multi-class SVM***

***A white paper with black text and black text

Description automatically generated***

* + 1. Tính toán hàm mất mát và đạo hàm của nó
* Để tối ưu hàm mất mát, chúng ta sử dụng phương pháp Stochastic Gradient Method.
* Để đảm bảo rằng *loss* và *gradient* được tính một cách chính xác và nhanh, chúng ta sẽ làm từng bước một. Bước thứ nhất là đảm bảo rằng các tính toán là chính xác, dù cách tính có rất chậm. Bước thứ hai là phải đảm bảo có cách tính hiệu quả để thuật toán chạy nhanh hơn. Hai bước này cần được thực hiện trên một lượng dữ liệu nhỏ để đảm bảo chúng được tính chính xác trước khi áp dụng thuật toán vào dữ liệu thật, thường có số điểm dữ liệu lớn và mỗi điểm dữ liệu cũng có số chiều lớn.
  + - 1. Tính hàm mất mát và đạo hàm của nó bằng naïve (đơn giản)
* Dưới đây là cách tính đơn giản *loss* và *gradient* của hàm mất mát trong (4). Chú ý thành phần *regularization*.

*import numpy as np*

*from random import shuffle*

*# naive way to calculate loss and grad*

*def svm\_loss\_naive(W, X, y, reg):*

*d, C = W.shape*

*\_, N = X.shape*

*## naive loss and grad*

*loss = 0*

*dW = np.zeros\_like(W)*

*for n in xrange(N):*

*xn = X[:, n]*

*score = W.T.dot(xn)*

*for j in xrange(C):*

*if j == y[n]:*

*continue*

*margin = 1 - score[y[n]] + score[j]*

*if margin > 0:*

*loss += margin*

*dW[:, j] += xn*

*dW[:, y[n]] -= xn*

*loss /= N*

*loss += 0.5\*reg\*np.sum(W \* W) # regularization*

*dW /= N*

*dW += reg\*W # gradient off regularization*

*return loss, dW*

*# random, small data*

*N, C, d = 10, 3, 5*

*reg = .1*

*W = np.random.randn(d, C)*

*X = np.random.randn(d, N)*

*y = np.random.randint(C, size = N)*

*# sanity check*

*print 'loss without regularization:', svm\_loss\_naive(W, X, y, 0)[0]*

*print 'loss with regularization:'svm\_loss\_naive(W, X, y, .1)[0]*

*loss without regularization: 4.68441457903*

*loss with regularization: 6.25136675351*

* *Cách tính với hai vòng for lồng nhau như trên mô tả lại chính xác loss trong (4) nên sai sót, nếu có, ở đây có thể được kiểm tra và sửa lại dễ dàng.*
* *Việc kiểm tra ở cuối cho cái nhìn ban đầu về hàm mất mát: dương và không có regularization sẽ có loss tổng cộng nhỏ hơn.*

Về cách tính *gradient* cho phần *data loss*, mặc dù [hàm max là *convex*](https://machinelearningcoban.com/2017/03/12/convexity/#-cac-tinh-chat-co-ban) nhưng nó không có đạo hàm tại mọi nơi. Cụ thể:

A group of numbers and symbols

Description automatically generated with medium confidence

Rõ ràng là các đạo hàm này không xác định tại các điểm mà . Tuy nhiên, khi thực hành, ta có thể giả sử rằng tại 0, các đạo hàm này cũng bằng 0.

Để kiểm tra lại cách tính đạo hàm như trên dựa vào (5) và (6) có chính xác không, chúng ta sẽ làm một bước quen thuộc là so sánh nó với [*numerical gradient*](https://machinelearningcoban.com/2017/01/12/gradientdescent/#kiem-tra-dao-ham). Nếu sự sai khác là nhỏ, nhỏ hơn 1e-7 thì ta có thể coi là *gradient* tính được là chính xác:

*f = lambda W: svm\_loss\_naive(W, X, y, .1)[0]*

*# for checking if calculated grad is correct*

*def numerical\_grad\_general(W, f):*

*eps = 1e-6*

*g = np.zeros\_like(W)*

*# flatening variable -> 1d. Then we need*

*# only one for loop*

*W\_flattened = W.flatten()*

*g\_flattened = np.zeros\_like(W\_flattened)*

*for i in xrange(W.size):*

*W\_p = W\_flattened.copy()*

*W\_n = W\_flattened.copy()*

*W\_p[i] += eps*

*W\_n[i] -= eps*

*# back to shape of W*

*W\_p = W\_p.reshape(W.shape)*

*W\_n = W\_n.reshape(W.shape)*

*g\_flattened[i] = (f(W\_p) - f(W\_n))/(2\*eps)*

*# convert back to original shape*

*return g\_flattened.reshape(W.shape)*

*# compare two ways of computing gradient*

*g1 = svm\_loss\_naive(W, X, y, .1)[1]*

*g2 = numerical\_grad\_general(W, f)*

*print 'gradient difference: %f' %np.linalg.norm(g1 - g2)*

*# this should be very small*

*gradient difference: 0.000000*

* **Sự sai khác là xấp xỉ 0, vậy chúng ta có thể yên tâm khi nói rằng cách tính *gradient* đã thỏa mãn sự *chính xác*, chúng ta cần tính nó một cách *hiệu quả* nữa.**
  + - 1. Tính hàm mất mát và đạo hàm của nó bằng vectorized

A diagram of a mathematical equation

Description automatically generated

* Ở đây, chúng ta tạm quên phần *regularization loss* đi vì cả *loss* và *gradient* của phần này đều có cách tính đơn giản. Với phần *data loss*, chúng ta cũng bỏ qua hệ số  đi cho dễ hình dung.
* Giả sử rằng có 4 classes và mini-batch gồm có 3 điểm dữ liệu ,. 3 điểm này lần lượt thuộc vào các class 1, 3, 2. Các ô có nền màu đỏ nhạt ở mỗi cột tương ứng với *correct class* của điểm dữ liệu của cột đó. Các bước tính *loss* và *gradient* có thể được hình dung như sau:
  + Bước 1: Tính score matrix
  + Bước 2: Với mỗi ô, tính Chú ý rằng ta không cần tính các ô có nền màu đỏ nhạt vì có thể coi chúng bằng 0 do trong biểu thức data loss. Sau khi tính được giá trị của từng ô, ta chỉ quan tâm tới các ô có giá trị lớn hơn 0 - là các ô được tô nền màu xanh lục. Lấy tổng của tất cả các phần tử ở các ô xanh lục, ta sẽ được data loss.
  + Bước 3: Theo công thức (6), với ô màu xanh lục ở hàng 2, cột 1, thì đạo hàm theo vector hệ số ,sẽ được cộng thêm một lượng , và đạo hàm theo vector hệ số , sẽ được trừ đi một lượng ,. Tương tự với các ô màu xanh lục còn lại. Với các ô màu đỏ ở hàng 1 cột 1, chúng ta chú ý rằng trong cùng cột 1, có bao nhiêu ô màu xanh lục thì có bấy nhiêu lần đạo hàm của  bị trừ đi một lượng , . Điều này được suy ra từ (5). Từ đó suy ra trong khối ô vuông thứ 3, giá trị của ô màu đỏ sẽ bằng đối số của tổng số lượng các ô màu xanh lục. Vậy nên ô màu đỏ ở hàng 1 cột 1 phải bằng -2.
  + Bước 4: Bây giờ cộng theo các hàng, ta sẽ được đạo hàm theo hệ số của class tương ứng.
* Bây giờ cộng theo các hàng, ta sẽ được đạo hàm theo hệ số của class tương ứng.

*# more efficient way to compute loss and grad*

***def******svm\_loss\_vectorized****(W, X, y, reg):*

*d, C* ***=*** *W.shape*

*\_, N* ***=*** *X.shape*

*loss* ***=*** *0*

*dW* ***=*** *np.zeros\_like(W)*

*Z* ***=*** *W.T.dot(X)*

*correct\_class\_score* ***=*** *np.choose(y, Z).reshape(N,1).T*

*margins* ***=*** *np.maximum(0, Z* ***-*** *correct\_class\_score* ***+*** *1)*

*margins[y, np.arange(margins.shape[1])]* ***=*** *0*

*loss* ***=*** *np.sum(margins, axis* ***=*** *(0, 1))*

*loss* ***/=*** *N*

*loss* ***+=*** *0.5* ***\**** *reg* ***\**** *np.sum(W* ***\**** *W)*

*F* ***=*** *(margins* ***>*** *0).astype(int)*

*F[y, np.arange(F.shape[1])]* ***=*** *np.sum(****-****F, axis* ***=*** *0)*

*dW* ***=*** *X.dot(F.T)****/****N* ***+*** *reg****\*****W*

***return*** *loss, dW*

* Sau khi đã viết đoạn code mà chúng ta *cho rằng* đã hiệu quả (không còn vòng for nào) này, chúng ta cần phải kiểm chứng hai điều:
  + - Quy trình 4 bước tôi nêu ở trên có chính xác không. Việc này được kiểm chứng bằng cách so sánh đạo hàm này với đạo hàm nhận được bằng cách tính *naive*.
    - Cách tính thứ hai này liệu có thực sự *hiệu quả*, tức có nhanh hơn cách *naive* nhiều không.

*N, C, d = 49000, 10, 3073*

*reg = .1*

*W = np.random.randn(d, C)*

*X = np.random.randn(d, N)*

*y = np.random.randint(C, size = N)*

*import time*

*t1 = time.time()*

*l1, dW1 = svm\_loss\_naive(W, X, y, reg)*

*t2 = time.time()*

*print 'Naive : run time:', t2 - t1, '(s)'*

*t1 = time.time()*

*l2, dW2 = svm\_loss\_vectorized(W, X, y, reg)*

*t2 = time.time()*

*print 'Vectorized: run time:', t2 - t1, '(s)'*

*print 'loss difference:', np.linalg.norm(l1 - l2)*

*print 'gradient difference:', np.linalg.norm(dW1 - dW2)*

***Naive : run time: 34.326472044 (s)***

***Vectorized: run time: 0.267823934555 (s)***

***loss difference: 3.63797880709e-12***

***gradient difference: 2.70855454684e-14***

Kết quả nhận được cho chúng ta thấy rằng cách tính bằng *vectorized* nhanh hơn rất nhiều (khoảng 120 lần) so với cách tính *naive*. Hơn nữa, sự chênh lệch giữa kết quả của hai cách tính là rất nhỏ, đều nhỏ hơn 1e-10 . Vậy thì chúng ta có thể *yên tâm* sử dụng cách *vectorized* này để cập nhật nghiệm.

* + - 1. Gradient Descent cho Multi-class SVM

*# Mini-batch gradient descent*

*def multiclass\_svm\_GD(X, y, Winit, reg, lr=.1, \*

*batch\_size = 100, num\_iters = 1000, print\_every = 100):*

*W = Winit*

*loss\_history = np.zeros((num\_iters))*

*for it in xrange(num\_iters):*

*# randomly pick a batch of X*

*idx = np.random.choice(X.shape[1], batch\_size)*

*X\_batch = X[:, idx]*

*y\_batch = y[idx]*

*loss\_history[it], dW = \*

*svm\_loss\_vectorized(W, X\_batch, y\_batch, reg)*

*W -= lr\*dW*

*if it % print\_every == 1:*

*print 'it %d/%d, loss = %f' \*

*%(it, num\_iters, loss\_history[it])*

*return W, loss\_history*

*N, C, d = 49000, 10, 3073*

*reg = .1*

*W = np.random.randn(d, C)*

*X = np.random.randn(d, N)*

*y = np.random.randint(C, size = N)*

*W, loss\_history = multiclass\_svm\_GD(X, y, W, reg)*

***it 1/1000, loss = 1802.750975***

***it 101/1000, loss = 251.495825***

***it 201/1000, loss = 62.021015***

***it 301/1000, loss = 45.626031***

***it 401/1000, loss = 38.334262***

***it 501/1000, loss = 43.666638***

***it 601/1000, loss = 45.649841***

***it 701/1000, loss = 35.401936***

***it 801/1000, loss = 36.211475***

***it 901/1000, loss = 41.676211***

* Chúng ta thử visisualize giá trị của loss sau mỗi vòng lặp:

*import matplotlib.pyplot as plt*

*# plot loss as a function of iteration*

*plt.plot(loss\_history)*

*plt.show()*

***A white rectangular object with black text

Description automatically generated***

* Từ *lịch sử loss* này ta thấy rằng giá trị của *loss* sau mỗi vòng lặp có xu hướng giảm và hội tụ, đây chính là điều mà chúng ta mong muốn.
  1. Tóm tắt
* [Giống như Softmax Regression](https://machinelearningcoban.com/2017/02/17/softmax/#-boundary-tao-boi-softmax-regression-la-linear), Multi-class SVM vẫn được coi là một bộ phân lớp tuyến tính vì đường phân chia giữa các class là các đường tuyến tính.
* Kernel SVM cũng hoạt động khá tốt, nhưng việc tính toán ma trận kernel có thể tốn nhiều thời gian và bộ nhớ. Hơn nữa, việc mở rộng nó ra cho bài toán multi-class classification thường không hiệu quả bằng Multi-class SVM. Một ưu điểm nữa của Multi-class SVM là nó có thể được tối ưu bằng (Stochastic) Gradient Descnet, tức là nó phù hợp với các bài toán large-scale. Việc boundary giữa các class là tuyến tính có thể được giải quyết bằng cách kết hợp nó với Deep Neurel Networks.
* Có một cách nữa mở rộng hinge loss cho bài toán multi-class classification là dùng loss:  Đây chính là vi phạm lớn nhất, so với tổng vi pham mà chúng ta sử dụng trong bài này.
* Trên thực tế, [**Multi-class SVM và Softmax Regression có hiệu quả tương đương nhau**](http://cs231n.github.io/linear-classify/#svmvssoftmax). Có thể trong một bài toán cụ thể, phương pháp này tốt hơn phương pháp kia, nhưng điều ngược lại xảy ra trong các bài toán khác.

## **A.2.K-nearest Neighbor**

1. **Giới thiệu.**

* K-nearest neighbor là một trong những thuật toán supervised-learning đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong Machine Learning.
* Một số tên gọi khác của phương pháp học dựa trên các láng giềng gần nhất (Nearest neighbor learning)
  + Instance-based learning
  + Lazy learning
  + Memory-based 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](https://en.wikipedia.org/wiki/Lazy_learning)), mọi tính toán được thực hiện khi nó cần dự đoán kết quả của dữ liệu mới. K-nearest neighbor có thể áp dụng được vào cả hai loại của bài toán Supervised learning là [Classification](https://machinelearningcoban.com/2016/12/27/categories/#classification-phan-loai) và [Regression](https://machinelearningcoban.com/2016/12/27/categories/#regression-hoi-quy).
  + Với KNN, trong bài toán Classification, label của một điểm dữ liệu mới được suy ra trực tiếp từ K điểm dữ liệu gần nhất trong training set. Label của một test data 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 trong các điểm gần nhất đó rồi suy ra label.
  + Trong bài toán Regresssion, 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 training set 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*.
* **Một số khái niệm**
  1. Chuẩn hóa dữ liệu
* Khi có một thuộc tính trong dữ liệu (hay phần tử trong vector) lớn hơn các thuộc tính khác rất nhiều (ví dụ thay vì đo bằng cm thì một kết quả lại tính bằng mm), khoảng cách giữa các điểm sẽ phụ thuộc vào thuộc tính này rất nhiều. Để có được kết quả chính xác hơn, một kỹ thuật thường được dùng là *Data Normalization* (chuẩn hóa dữ liệu) để đưa các thuộc tính có đơn vị đo khác nhau về cùng một khoảng giá trị, thường là từ 0 đến 1, trước khi thực hiện KNN.



* Chuẩn hóa min-max:

A math equation with white text

Description automatically generated

1. **K-Nearest Neighbor**
   1. Ý tưởng

* Với một tập các ví dụ học
  + Lưu lại các ví dụ học
  + Chưa xây dựng một mô hình (mô tả) rõ ràng và tổng quát của hàm mục tiêu cần học
* Đối với một ví dụ cần phân loại/hồi quy
  + Xét quan hệ giữa ví dụ đó với các ví dụ học để gán giá trị của hàm mục tiêu (một nhãn lớp, hoặc một giá trị thực)
  1. Bài toán
* Biểu diễn đầu vào của bài toán
  + Mỗi ví dụ x được biểu diễn là một vectơ n chiều trong không gian các vecto X∈
  + x = () , trong đó ∈ R là một số thực
* Có thể áp dụng với 2 bài toán Phân loại và Hồi quy (Classification/ Regression):
  + Bài toán phân lớp (classification)
    - Hàm mục tiêu có giá trị rời rạc (a discrete-valued target function)
    - Đầu ra của hệ thống là một trong số các giá trị rời rạc đã xác định trước (một trong các nhãn lớp)
  + Bài toán hồi quy (regression)
    - Hàm mục tiêu có giá trị liên tục (a continuous-valued target function)
    - Đầu ra của hệ thống là một giá trị số thực
* Ví dụ

A diagram of a diagram of a red circle with a blue dot

Description automatically generated with medium confidence

* Xét 1 láng giềng gần nhất

Gán z vào lớp c2

* Xét 3 láng giềng gần nhất
  + - Gán z vào lớp c1
* Xét 5 láng giềng gần nhất
  + - Gán z vào lớp c1
  + Giải thuật Phân lớp k-NN
* Mỗi ví dụ học x được biểu diễn bởi 2 thành phần:
  + x = () , trong đó ∈ **R** là một số thực
  + Nhãn lớp: c ∈ **C**, với **C** là tập các nhãn lớp được xác định trước
* Giai đoạn học
  + Đơn giản là lưu lại các ví dụ học trong tập học: **D** = { x }
* Giai đoạn phân lớp: Để phân lớp cho một ví dụ (mới) z
  + Với mỗi ví dụ học x ∈ **D**, tính khoảng cách giữa x và z
  + Xác định tập **NB**(z) - các láng giềng gần nhất của z
    - Gồm k ví dụ học trong **D** gần nhất với z tính theo một hàm khoảng cách d
  + Phân z vào lớp chiếm số đông (the majority class) trong số các lớp của các ví dụ học trong **NB**(z)
  + Giải thuật hồi quy k-NN
* Mỗi ví dụ học x được biểu diễn bởi 2 thành phần:
  + x = () , trong đó ∈ **R** là một số thực
  + Giá trị đầu ra mong muốn:∈ **R** là một số thực
* Giai đoạn học
  + Đơn giản là lưu lại các ví dụ học trong tập học: **D** = { x }
* Giai đoạn phân lớp: Để phân lớp cho một ví dụ (mới) z
  + Với mỗi ví dụ học x ∈ **D**, tính khoảng cách giữa x và z
  + Xác định tập **NB**(z) - các láng giềng gần nhất của z
    - Gồm k ví dụ học trong **D** gần nhất với z tính theo một hàm khoảng cách d
  + Dự đoán giá trị đầu ra đối với z :
  + Một hay nhiều **nearest neighbor**?
* Việc phân lớp (hay dự đoán) chỉ dựa trên duy nhất một láng giềng gần nhất (là ví dụ học gần nhất với ví dụ cần phân lớp/dự đoán) thường không chính xác
* Nếu ví dụ học này là một ví dụ bất thường, không điển hình (an outlier) rất khác so với các ví dụ khác
* Nếu ví dụ học này có nhãn lớp (giá trị đầu ra) sai– do lỗi trong quá trình thu thập (xây dựng) tập dữ liệu
* Thường xét k > 1 các ví dụ học (các láng giềng) gần nhất với ví dụ cần phân lớp/dự đoán
* Đối với bài toán phân lớp có 2 lớp, k thường được chọn là một số lẻ, để tránh cân bằng về tỷ lệ các ví dụ giữa 2 lớp
  + Ví dụ: k = 3,5,7,...
  1. Hàm tính khoảng cách d
* Hàm tính khoảng cách d
  + Đóng vai trò rất quan trọng trong phương pháp học dựa trên các láng giềng gần nhất
  + Thường được xác định trước, và không thay đổi trong suốt quá trình học và phân loại/dự đoán
* Lựa chọn hàm khoảng cách d
  + Các hàm khoảng cách hình học: Dành cho các bài toán có các thuộc tính đầu vào là kiểu số thực ∈ **R**
  + Hàm khoảng cách Hamming: Dành cho các bài toán có các thuộc tính đầu vào là kiểu nhị phân ∈ {0,1}
  + Hàm tính độ tương tự Cosine: Dành cho các bài toán phân lớp văn bản ( là giá trị trọng số TF/IDF của từ khóa thứ i)

1. Các hàm tính khoảng cách hình học (Geometry distance functions)

A math equations with black text

Description automatically generated with medium confidence

1. Hàm khoảng cách
   * Hamming

A mathematical equation with numbers and symbols

Description automatically generated

* + - Đối với các thuộc tính đầu vào là kiểu nhị phân {0,1}
    - Ví dụ: x = (0,1,0,1,1)

1. Hàm tính độ tương tự
   * Cosine

A math equations with numbers

Description automatically generated with medium confidence

* + Đối với đầu vào là một vectơ các giá trị trọng số (TF/IDF) của các từ khóa
  1. Trọng số các thuộc tính
  + Làm sao để xác định các giá trị trọng số của các thuộc tính?
    - Dựa trên các tri thức cụ thể của bài toán (vd: được chỉ định bởi các chuyên gia trong lĩnh vực cra bài toán đang xét)
    - Bằng một quá trình tối ưu hóa các giá trị trọng số (vd: sử dụng một tập học để học một bộ các giá trị trọng số tối ưu)
* Xét tập **NB**(z) - gồm k ví dụ học gần nhất với ví dụ cần phân lớp/dự đoán z
  + Mỗi ví dụ (láng giềng gần nhất) này có khoảng cách khác nhau đến z
  + Các láng giềng này có cảnh hưởng khác nhau đối với việc phân lớp/dự đoán cho z
* Gọi v là hàm xác định trọng số theo khoảng cách
  + Đối với một giá trị d(x,z) - khoảng cách giữa x và z
  + v(x,z) tỷ lệ nghịch với d(x,z)
* Đối với bài toán phân lớp:

A math equations and formulas

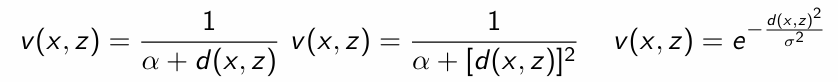
Description automatically generated with medium confidence

* Đối với bài toán dự đoán (hồi quy):

A black and white math symbols

Description automatically generated with medium confidence

* Lựa chọn một hàm xác định trọng số theo khoảng cách:



* 1. Ưu nhược điểm
* Ưu điểm
  + Độ phức tạp tính toán của quá trình training là bằng 0.
  + Việc dự đoán kết quả của dữ liệu mới rất đơn giản.
  + Không cần giả sử gì về phân phối của các class.
* Nhược điểm của KNN
  + KNN rất nhạy cảm với nhiễu khi K nhỏ.
  + Như đã nói, KNN là một thuật toán mà mọi tính toán đều nằm ở khâu test. Trong đó việc tính khoảng cách tới *từng* điểm dữ liệu trong training 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. Sử dụng khi nào?
  + Các ví dụ được biểu diễn trong không gian vectơ
  + Số lượng các thuộc tính (để biểu diễn ví dụ) là không nhiều
  + Một tập học có kích thước lớn
  1. Tăng tốc cho KNN
* Ngoài việc tính toán khoảng cách từ một điểm test data đến tất cả các điểm trong traing set (Brute Force), có một số thuật toán khác giúp tăng tốc việc tìm kiếm này. Bạn đọc có thẻ tìm kiếm thêm với hai từ khóa: [K-D Tree](http://pointclouds.org/documentation/tutorials/kdtree_search.php) và [Ball Tree](https://en.wikipedia.org/wiki/Ball_tree).

1. **Bộ dữ liệu Maintance**
2. **Giới thiệu**

Các quy trình sản xuất và công nghiệp đóng vai trò cốt lõi trong việc tạo ra hàng hóa và dịch vụ cần thiết cho xã hội. Chúng bao gồm các chuỗi hoạt động có tổ chức nhằm biến nguyên vật liệu thành các sản phẩm hoàn chỉnh hoặc bán thành phẩm, được tối ưu hóa nhằm đạt hiệu suất cao, chất lượng ổn định và chi phí thấp nhất có thể.

AI4I, hay Artificial Intelligence for Industry, là một nền tảng và chương trình của Siemens được thiết kế nhằm thúc đẩy sự phát triển và ứng dụng AI trong các quy trình sản xuất và công nghiệp. AI4I cung cấp các công cụ, dữ liệu và nền tảng cần thiết để các công ty có thể triển khai các ứng dụng AI vào quy trình sản xuất của họ, giúp cải thiện hiệu suất, giảm thiểu lãng phí và tối ưu hóa chi phí.

AI4I tập trung vào các giải pháp AI như bảo trì dự đoán (predictive maintenance), phát hiện lỗi trong chuỗi sản xuất, tối ưu hóa quy trình và phân tích dữ liệu sản xuất theo thời gian thực. Những giải pháp này không chỉ giúp doanh nghiệp nâng cao chất lượng sản phẩm mà còn giúp cải thiện hiệu suất sử dụng thiết bị và tài nguyên.

Hiện nay, các ngành công nghiệp ngày càng ứng dụng công nghệ như AI, robot, IoT, và phân tích dữ liệu lớn để tối ưu hóa quy trình. Những công nghệ này không chỉ giúp tăng cường hiệu suất, mà còn cải thiện độ chính xác, giảm thiểu lãng phí và giúp quy trình trở nên linh hoạt hơn.

1. **Bộ dữ liệu maintance**

Bộ dữ liệu AI4I (AI for Industry 4.0) là một tập dữ liệu được xây dựng với mục tiêu hỗ trợ việc nghiên cứu và phát triển các giải pháp trí tuệ nhân tạo (AI) trong lĩnh vực công nghiệp, đặc biệt là công nghiệp sản xuất thông minh. Bộ dữ liệu này chứa các thông tin liên quan đến các quá trình sản xuất và hoạt động của máy móc trong môi trường sản xuất công nghiệp, bao gồm các chỉ số về trạng thái hoạt động, hiệu suất, và các yếu tố gây lỗi của thiết bị.

Bộ dữ liệu AI4I có các cột thông tin như sau:

* **UDI** (Unique Device Identifier): Mã định danh duy nhất của thiết bị, giúp nhận diện từng mẫu dữ liệu một cách riêng biệt.
* **Product ID**: Mã sản phẩm được sản xuất, có thể liên quan đến loại sản phẩm hoặc lô sản phẩm cụ thể.
* **Type**: Loại sản phẩm hoặc danh mục của sản phẩm, phân biệt các dòng sản phẩm khác nhau trong quá trình sản xuất.
* **Air Temperature [K]**: Nhiệt độ không khí xung quanh máy móc trong quá trình sản xuất, tính bằng độ Kelvin (K).
* **Process Temperature [K]**: Nhiệt độ tại điểm hoạt động của quá trình sản xuất, tính bằng độ Kelvin (K). Thường cao hơn nhiệt độ không khí do phát sinh nhiệt từ hoạt động máy móc.
* **Rotational Speed [rpm]**: Tốc độ quay của động cơ, tính bằng vòng/phút (rpm), thể hiện tốc độ hoạt động của máy móc.
* **Torque [Nm]**: Mô-men xoắn (torque), tính bằng Newton-mét (Nm), thể hiện lực quay của máy móc trong quá trình vận hành.
* **Tool Wear [min]**: Độ mòn của dụng cụ sản xuất, tính bằng phút (min), biểu thị thời gian công cụ đã được sử dụng trong sản xuất. Đây là thông số quan trọng để đánh giá tình trạng của công cụ và khả năng cần bảo trì hoặc thay thế.
* **Target**: Nhãn mục tiêu (target), xác định xem một sản phẩm có đạt tiêu chuẩn hay không. Đây có thể là biến nhị phân, cho biết sản phẩm đã qua kiểm tra chất lượng (đạt hoặc không đạt).
* **Failure Type**: Loại lỗi gặp phải (nếu có) trong quá trình sản xuất. Các giá trị trong cột này có thể bao gồm:
  + Không lỗi (No Failure)
  + Lỗi quá tải công cụ (Tool Wear Failure)
  + Lỗi nguồn nhiệt (Heat Dissipation Failure)
  + Lỗi công cụ (Power Failure)
  + Lỗi bôi trơn (Overstrain Failure)
  + Lỗi môi trường (Random Failures)

Bộ dữ liệu này có thể được dùng để xây dựng các mô hình AI và phân tích dữ liệu, nhằm dự đoán lỗi hoặc tối ưu hóa hiệu suất của máy móc. Các cột **Rotational Speed**, **Torque**, và **Tool Wear** có thể đóng vai trò quan trọng trong việc phân tích sự hao mòn của máy móc và dự đoán thời điểm bảo trì cần thiết. Các cột **Air Temperature** và **Process Temperature** giúp đánh giá sự ảnh hưởng của nhiệt độ đến hiệu suất hoạt động và tuổi thọ của thiết bị.

1. **Cân bằng dữ liệu SMOTE - Synthetic Minority Over-sampling Technique**

SMOTE (Synthetic Minority Over-sampling Technique) là một phương pháp cân bằng dữ liệu trong các bài toán phân loại mất cân bằng. Phương pháp này tạo thêm các mẫu tổng hợp từ lớp thiểu số (minority class) thay vì chỉ nhân bản ngẫu nhiên, giúp cải thiện hiệu suất của các mô hình học máy khi có sự mất cân bằng giữa các lớp.

**Cách hoạt động của SMOTE**

* **Chọn các điểm dữ liệu từ lớp thiểu số**: SMOTE bắt đầu bằng cách chọn ngẫu nhiên một điểm dữ liệu thuộc lớp thiểu số.
* **Xác định hàng xóm gần nhất**: Sau khi chọn một điểm từ lớp thiểu số, SMOTE tìm k hàng xóm gần nhất của điểm này trong cùng lớp thiểu số. Số lượng hàng xóm k thường được chọn là 5, nhưng có thể điều chỉnh tùy vào tình huống.
* **Tạo các điểm tổng hợp**: Một điểm ngẫu nhiên từ các hàng xóm gần nhất được chọn và SMOTE tạo một điểm tổng hợp mới bằng cách nội suy giữa điểm ban đầu và điểm hàng xóm. Quá trình nội suy được thực hiện bằng cách:

A group of symbols on a white background

Description automatically generated

trong đó  là một số ngẫu nhiên trong khoảng [0,1].

Kết quả là một điểm mới nằm trên đoạn thẳng nối giữa hai điểm, thuộc không gian của lớp thiểu số.

* **Lặp lại**: Quy trình trên được lặp lại cho đến khi đạt được số lượng mẫu mong muốn để cân bằng với lớp đa số.
* **Borderline-SMOTE**

Borderline-SMOTE (Borderline Synthetic Minority Over-sampling Technique) là một biến thể của thuật toán SMOTE (Synthetic Minority Over-sampling Technique), được sử dụng để cân bằng dữ liệu mất cân bằng trong các bài toán phân loại. Trong các bài toán này, số lượng mẫu của lớp thiểu số (minority class) thường nhỏ hơn rất nhiều so với lớp đa số (majority class), làm giảm độ chính xác và độ tin cậy của mô hình. Borderline-SMOTE tập trung vào việc tạo mẫu tổng hợp cho các điểm dữ liệu nằm gần biên giới giữa các lớp, nơi mà khả năng nhầm lẫn giữa các lớp thường cao hơn.

**Cách hoạt động của Borderline-SMOTE**

1. **Xác định các mẫu khó phân loại**: Đầu tiên, Borderline-SMOTE xác định các mẫu của lớp thiểu số nằm gần biên giới phân chia giữa lớp thiểu số và lớp đa số, cụ thể là các mẫu có nguy cơ bị nhầm lẫn cao hơn. Các mẫu này thường nằm gần các điểm của lớp đa số.
2. **Phân loại các điểm biên**: Những điểm dữ liệu thuộc lớp thiểu số gần kề với lớp đa số được chọn làm các điểm biên giới (borderline). Borderline-SMOTE xác định những điểm này bằng cách sử dụng một số tiêu chí như số lượng hàng xóm gần nhất thuộc lớp đa số. Những mẫu này có nguy cơ cao bị phân loại sai trong quá trình học của mô hình.
3. **Tạo các mẫu tổng hợp**: Sau khi xác định các điểm biên giới, Borderline-SMOTE tạo ra các điểm tổng hợp mới bằng cách nội suy giữa các điểm biên này với các hàng xóm gần nhất của chúng (thuộc lớp thiểu số). Điều này giúp tăng số lượng mẫu ở các khu vực quan trọng nhất (biên giới giữa các lớp) thay vì tăng số lượng mẫu trên toàn bộ không gian lớp thiểu số như SMOTE gốc.
4. **Chuẩn hóa Minmax**

Chuẩn hóa Min-Max (Min-Max Normalization) là một phương pháp được sử dụng trong tiền xử lý dữ liệu nhằm đưa các giá trị của thuộc tính về một khoảng nhất định, thường là từ 0 đến 1.

Giả sử có một tập dữ liệu với một thuộc tính x, và giá trị của thuộc tính này nằm trong khoảng từ ​ đến ​. Chuẩn hóa Min-Max đưa các giá trị của x về khoảng [0,1] bằng công thức:

A white background with black text

Description automatically generated

Chuẩn hóa Minimax đặc biệt hữu ích trong ML, nơi các đặc điểm có quy mô hoặc phạm vi lớn có thể chi phối quá trình phân tích và tạo ra kết quả không chính xác. Bằng cách áp dụng chuẩn hóa Minimax cho từng đặc điểm trong tập dữ liệu, chúng ta có thể đảm bảo rằng tất cả các đặc điểm đều có đóng góp như nhau cho quá trình phân tích, bất kể quy mô hoặc phạm vi ban đầu của chúng.

1. **Mối tương quan giữa các đặc trưng**

Ma trận tương quan là một công cụ thống kê được sử dụng để đo lường và thể hiện mức độ tương quan giữa các biến trong một tập dữ liệu.

Trong ma trận này:

* Màu đỏ đậm đại diện cho mối tương quan dương cao (gần +1), có nghĩa là khi một biến tăng, biến kia cũng có xu hướng tăng.
* Màu xanh đậm đại diện cho mối tương quan âm cao (gần -1), có nghĩa là khi một biến tăng, biến kia có xu hướng giảm.
* Màu xám hoặc màu nhạt gần như không có mối tương quan (giá trị gần 0).

Dưới đây là ma trận tương quan thể hiện mối tương quan giữa các đặc trưng của bộ dữ liệu:

A screenshot of a graph

Description automatically generated

Dựa trên ma trận tương qua, ta thấy:

* Biến **air** và **process** có mối tương quan dương cao, cho thấy nhiệt độ không khí và nhiệt độ quá trình có quan hệ chặt chẽ với nhau.
* Biến **rotational speed** và **torque** có mối tương quan âm, tức là khi tốc độ quay tăng, mô-men xoắn có xu hướng giảm.
* Biến **tool wear** có mối tương quan thấp với các biến còn lại, nên có thể độc lập hơn và không ảnh hưởng nhiều đến các biến kia.

1. **Triển khai trên bộ dữ liệu**

****

* Loại bỏ các đăc trưng tên gọi

A screenshot of a computer error

Description automatically generated

A white background with red text

Description automatically generated

* Loại bỏ đặc trưng Failure Type vì chênh lệch dữ liệu

A computer code with black text

Description automatically generated

* Chuyển các đặc trưng phân loại về dạng số

A close-up of a text

Description automatically generated

A screen shot of a computer code

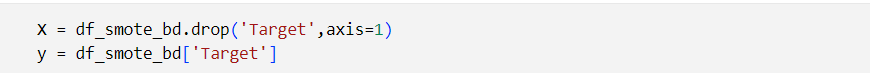
Description automatically generated

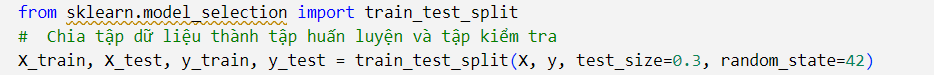
* Áp dụng BorderlineSMOTE để xử lí sự mất cân bằng dữ liệu

A screenshot of a computer code

Description automatically generated

* Áp dụng Chuẩn hóa Min-Max với những đặc trưng không phải đặc trưng phân loại





A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

* Khởi chạy các mô hình

1. **Phân tích hiệu suất của các thuật toán**

Báo cáo sử dụng 2 thuật toán dựa trên ML, cụ thể là SVC và KNN để dự đoán lỗi của máy móc. Để kiểm tra hiệu quả của các thuật toán, báo cáo sử dụng bộ dữ liệu gồm 10000 mẫu máy móc gồm máy móc lỗi và máy móc bình thường. Các tập dữ liệu này đã cung cấp các điểm chuẩn quan trọng để đánh giá các thuật toán dựa trên ML. Tuy nhiên, chất lượng và tính đa dạng của dữ liệu đào tạo cũng như các tính năng và tham số của thuật toán có thể làm giảm khả năng xác định các máy móc lỗi. Các mô hình dựa trên ML được đề xuất đã chứng minh là một phương pháp hiệu quả để phát hiện các máy móc lỗi

* 1. **Support Vector Classification (SVC)**

A yellow and purple squares with numbers

Description automatically generatedA graph of a line

Description automatically generated

A graph of a line

Description automatically generated

* + 1. Ma trận nhầm lẫn
    2. Biểu đồ ROC – AUC
    3. Biểu đồ Precison-Recall

Hiệu suất của mô hình SVC

| **Chỉ số** | **Precision** | **Recall** | **F1-score** | **Accuracy** |
| --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Giá trị** | 0.869 | 0.881 | 0.875 | 0.874 |

1. **Ma trận nhầm lẫn**:
   * Mô hình SVC đã phân loại chính xác phần lớn các mẫu trong tập dữ liệu, với 2517 và 2502 là số lượng dự đoán đúng cho mỗi lớp.
   * Tuy nhiên, vẫn còn có một số dự đoán sai, với 364 mẫu của lớp 0 bị dự đoán thành lớp 1 và 316 mẫu của lớp 1 bị dự đoán thành lớp 0.
2. **Biểu đồ ROC – AUC**:
   * Đường cong ROC gần sát với góc trên bên trái của biểu đồ, cho thấy mô hình có khả năng phân biệt tốt giữa các lớp.
   * Giá trị AUC đạt 0.95, là một con số cao, chứng tỏ mô hình có độ nhạy cao trong việc phân biệt các lớp.
3. **Biểu đồ Precision-Recall**:
   * Đường cong Precision-Recall cho thấy mô hình đạt độ chính xác và độ nhạy cao khi tỷ lệ Recall ở mức cao.
   * Diện tích dưới đường cong (AP = 0.94) cũng cho thấy khả năng của mô hình trong việc xử lý lớp thiểu số hoặc mất cân bằng trong dữ liệu.
4. **Hiệu suất của mô hình**:
   * Các chỉ số đánh giá của mô hình như Precision (0.869), Recall (0.881), F1-score (0.875), và Accuracy (0.874) đều cho thấy mô hình có hiệu suất tốt và ổn định.
   * Precision và Recall có giá trị khá gần nhau, điều này cho thấy mô hình có sự cân bằng giữa việc nhận diện đúng các mẫu và giảm thiểu dự đoán sai.

**Tổng kết:**

Mô hình SVC này đạt hiệu suất khá tốt trên tập dữ liệu hiện tại. Tuy nhiên, vẫn có thể tối ưu thêm để giảm số lượng mẫu bị phân loại sai, chẳng hạn bằng cách điều chỉnh các tham số của mô hình hoặc sử dụng các kỹ thuật tăng cường dữ liệu nếu có sự mất cân bằng giữa các lớp.

* 1. **K-nearest Neighbor (KNN)**

A graph of a line

Description automatically generatedA yellow and purple squares with numbers

Description automatically generated A graph with a line

Description automatically generated

* + 1. Ma trận nhầm lẫn
    2. Biểu đồ ROC – AUC
    3. Biểu đồ Precison-Recall

**Hiệu suất của mô hình KNN**

| **Chỉ số** | **Precision** | **Recall** | **F1-score** | **Accuracy** |
| --- | --- | --- | --- | --- |
| **Giá trị** | 0.943 | 0.987 | 0.964 | 0.963 |

1. **Ma trận nhầm lẫn**:
   * Mô hình KNN phân loại đúng phần lớn các mẫu trong tập dữ liệu, với 2727 và 2857 là số lượng dự đoán đúng cho mỗi lớp.
   * Tuy nhiên, vẫn có một số lượng nhỏ mẫu bị phân loại sai, với 174 mẫu của lớp 0 bị dự đoán thành lớp 1 và 39 mẫu của lớp 1 bị dự đoán thành lớp 0, cho thấy mô hình có hiệu suất tốt hơn ở lớp 1.
2. **Biểu đồ ROC – AUC**:
   * Đường cong ROC sát với góc trên bên trái của biểu đồ, cho thấy mô hình có khả năng phân biệt giữa các lớp khá tốt.
   * Giá trị AUC đạt 0.99, cho thấy mô hình KNN có độ chính xác cao trong việc phân loại và khả năng phân biệt hai lớp mạnh mẽ.
3. **Biểu đồ Precision-Recall**:
   * Đường cong Precision-Recall cho thấy mô hình duy trì độ chính xác và độ nhạy cao trong phần lớn các giá trị Recall.
   * Diện tích dưới đường cong Precision-Recall (AP = 0.98) cho thấy mô hình hoạt động hiệu quả trong việc phân loại các mẫu của lớp thiểu số.
4. **Hiệu suất của mô hình**:
   * Các chỉ số Precision (0.943), Recall (0.987), F1-score (0.964), và Accuracy (0.963) đều cho thấy mô hình KNN đạt hiệu suất rất cao và ổn định.
   * Giá trị Recall cao (0.987) cho thấy mô hình ít bỏ sót các mẫu dương tính, trong khi Precision cũng cao, cho thấy mô hình ít dự đoán sai các mẫu âm tính.

**Tổng kết:**

Mô hình KNN cho thấy hiệu suất rất tốt trên tập dữ liệu hiện tại, với độ chính xác và khả năng phân loại rất cao. Các chỉ số đều ở mức tốt, đặc biệt là Recall và F1-score, cho thấy mô hình có thể cân bằng giữa độ chính xác và độ nhạy.

1. **Phân tích độ nhạy của các thuật toán được đề xuất**

Để đánh giá độ chính xác dự đoán của các thuật toán dựa trên ML, báo cáo này đã sử dụng một số số liệu thống kê, bao gồm lỗi tuyệt đối trung bình (MAE), lỗi bình phương trung bình (MSE) và R bình phương (R2). Các lỗi dự đoán thu được giữa lớp mục tiêu và các giá trị dự đoán được trình bày trong bảng dưới đây. Phân tích được trình bày cung cấp những hiểu biết có giá trị về hiệu suất của các mô hình dựa trên ML, cho phép chúng tôi xác định và giải quyết mọi lỗi dự đoán.

| **Chỉ số** | **Mô hình SVC** | **Mô hình KNN** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| **MAE** | **0.1256** | **0.0367** |

|  |  |  |
| --- | --- | --- |
| **MSE** | **0.1256** | **0.0367** |

|  |  |  |
| --- | --- | --- |
| **R-squared (R²)** | **0.4977** | **0.8530** |

* **MAE và MSE:**
  + Mô hình KNN có MAE và MSE thấp hơn SVC, nghĩa là sai số trung bình của KNN nhỏ hơn, dữ đoán chính xác hơn trong việc dự đoán giá trị thực tế.
* **R²-score:**
  + Hệ số R² của KNN cao hơn nhiều, cho thấy KNN giải thích được phần lớn biến thiên của dữ liệu tốt hơn.

**Tổng kết:**

* Mô hình **KNN** vượt trội hơn so với SVC trên cả ba chỉ số: MAE, MSE và R², thể hiện khả năng dự đoán chính xác và ổn định hơn trên tập dữ liệu hiện tại.
* Với R² cao (0.8530), KNN cho thấy mô hình này phù hợp hơn và có khả năng giải thích biến thiên của dữ liệu cao hơn nhiều so với SVC (0.4977).

1. **Thảo luận**

Trong việc dự đoán lỗi máy móc, Machine Learning (ML) đóng vai trò quan trọng giúp doanh nghiệp giảm thiểu rủi ro và chi phí liên quan đến sự cố thiết bị. Các thuật toán ML phân tích dữ liệu từ cảm biến như nhiệt độ, độ rung, áp suất và âm thanh để phát hiện các dấu hiệu bất thường báo trước sự cố. Theo nghiên cứu, việc áp dụng ML để dự đoán lỗi có thể giúp các công ty giảm tới 30-50% chi phí bảo trì và giảm tới 70% thời gian ngừng máy đột xuất. Điều này đặc biệt có giá trị trong các ngành công nghiệp như sản xuất, dầu khí, và năng lượng, nơi thời gian ngừng hoạt động đột ngột có thể gây thiệt hại hàng triệu đô la mỗi giờ. Bằng cách phát hiện sớm các vấn đề tiềm ẩn, ML giúp các doanh nghiệp chuyển từ bảo trì theo lịch trình sang bảo trì dự đoán (predictive maintenance), tối ưu hóa hiệu suất thiết bị và nâng cao tuổi thọ máy móc.

| **Chỉ số** | **Mô hình SVC** | **Mô hình KNN** | **Nhận xét** |
| --- | --- | --- | --- |
| **Precision** | 0.869 | 0.943 | KNN có độ chính xác cao hơn SVC, ít dự đoán sai hơn cho các mẫu dương tính. |
| **Recall** | 0.881 | 0.987 | KNN có Recall cao hơn, cho thấy khả năng nhận diện đúng các mẫu dương tính tốt hơn. |
| **F1-score** | 0.875 | 0.964 | KNN đạt F1-score cao hơn, cho thấy cân bằng tốt giữa Precision và Recall. |
| **Accuracy** | 0.874 | 0.963 | Độ chính xác của KNN cao hơn SVC, cho thấy hiệu suất tổng thể tốt hơn trên tập dữ liệu. |
| **MAE** | 0.1256 | 0.0367 | MAE của KNN thấp hơn, nghĩa là sai số trung bình của KNN nhỏ hơn, dự đoán chính xác hơn. |
| **MSE** | 0.1256 | 0.0367 | MSE của KNN cũng thấp hơn, cho thấy sai số bình phương nhỏ hơn so với SVC. |
| **R-squared (R²)** | 0.4977 | 0.8530 | KNN có R² cao hơn nhiều, nghĩa là giải thích tốt hơn phần lớn biến thiên của dữ liệu. |

**Tổng kết:**

* Mô hình **KNN** thể hiện hiệu suất vượt trội hơn so với **SVC** trên tất cả các chỉ số đánh giá, đặc biệt là ở Precision, Recall, F1-score, và R².
* Với độ chính xác cao hơn, sai số thấp hơn, và khả năng giải thích biến thiên dữ liệu tốt hơn, **KNN** là lựa chọn tối ưu cho bài toán

1. **Bộ dữ liệu Material**
2. **Giới thiệu**

Lựa chọn vật liệu trong ngành ô tô là một yếu tố quan trọng, ảnh hưởng trực tiếp đến hiệu suất, độ an toàn, chi phí và tác động môi trường của phương tiện. Các nhà sản xuất ô tô luôn tìm cách sử dụng vật liệu tối ưu để đáp ứng yêu cầu về độ bền, trọng lượng nhẹ, khả năng chống ăn mòn và chi phí thấp. Với sự phát triển công nghệ, ngành ô tô đang dần chuyển sang các vật liệu nhẹ, bền và thân thiện môi trường, nhằm tạo ra những mẫu xe không chỉ an toàn và tiết kiệm nhiên liệu mà còn có khả năng tái chế cao, giảm thiểu tác động đến môi trường.

Ứng dụng học máy (Machine Learning - ML) vào việc lựa chọn vật liệu trong ngành ô tô là một bước tiến lớn giúp tối ưu hóa quy trình, giảm thời gian phát triển sản phẩm và nâng cao hiệu suất. ML giúp phân tích và dự đoán các đặc tính của vật liệu, từ đó đưa ra lựa chọn phù hợp với yêu cầu kỹ thuật, chi phí và tính bền vững. Bằng cách tận dụng ML để dự đoán tính chất, tối ưu hóa chi phí và tìm ra vật liệu mới thân thiện môi trường, ngành ô tô có thể cải thiện chất lượng, độ an toàn và tính bền vững của sản phẩm.

Trong tương lai, các công nghệ ML kết hợp với Digital Twin và các phương pháp mô phỏng sẽ mở ra nhiều cơ hội lớn trong việc phát triển các loại vật liệu tiên tiến cho ô tô.

1. **Bộ dữ liệu material**

Bộ dữ liệu gồm 2 bảng: material.csv và Data.csv, gồm 15 đặc trưng mô tả các đặc điểm kĩ thuật của vật liệu và 1 nhãn.

Dưới đây là tổng quan về các đặc trưng:

* **Std**: Tiêu chuẩn vật liệu hoặc mã định danh tiêu chuẩn (Standard).
* **ID**: Mã định danh duy nhất của vật liệu.
* **Material**: Loại vật liệu.
* **Heat treatment**: Phương pháp xử lý nhiệt của vật liệu.
* **Su**: Độ bền kéo (Ultimate tensile strength).
* **Sy**: Giới hạn chảy (Yield strength).
* **A5**: Độ giãn dài đến khi đứt (percentage elongation).
* **Bhn**: Độ cứng Brinell của vật liệu.
* **E**: Mô đun đàn hồi (Elastic modulus).
* **G**: Mô đun cắt (Shear modulus).
* **mu**: Hệ số ma sát (Friction coefficient).
* **Ro**: Khối lượng riêng (Density).
* **pH**: Độ pH, có thể liên quan đến tính chất hóa học bề mặt hoặc môi trường thử nghiệm.
* **Desc**: Mô tả vật liệu hoặc chi tiết về mẫu thử.
* **HV**: Độ cứng Vickers.
* **Use**: Vật liệu có được sử dụng hay không

1. **Tiền xử lý dữ liệu**

Tiền xử lý dữ liệu là một bước rất quan trọng trong việc giải quyết bất kỳ vấn đề nào trong lĩnh vực Học Máy. Hầu hết các bộ dữ liệu được sử dụng trong các vấn đề liên quan đến Học Máy cần được xử lý, làm sạch và biến đổi trước khi một thuật toán Học Máy có thể được huấn luyện trên những bộ dữ liệu này.

Việc xử lý dữ liệu trên tập dữ liệu tập trung vào việc rời rạc hóa, các giá trị bị thiếu được quy kết và dữ liệu không cân bằng, loại bỏ các biến chất lượng thấp và sử dụng các quy trình làm sạch dữ liệu để xác định và sửa chữa sự không nhất quán trong tập dữ liệu. Việc xác định và loại bỏ các mẫu trùng lặp khỏi tập dữ liệu là một bước quan trọng trong giai đoạn chuẩn bị dữ liệu, bằng cách sử dụng phương pháp sau. Giả sử P và H lần lượt là tên gói và giá trị băm SHA256 của mỗi mẫu. Tập hợp các mẫu duy nhất có thể thu được bằng cách chọn mẫu s thuộc S sao cho s có sự kết hợp duy nhất của P và H, như minh họa dưới đây:



Xử lý các giá trị đặc trưng bị thiếu bằng cách sử dụng phép quy ước trung bình hoặc trung vị dựa trên phân phối của các giá trị đặc trưng.

A close up of a number

Description automatically generated with medium confidence

Hoặc

A close up of a text

Description automatically generated

1. **Cân bằng dữ liệu SMOTE - Synthetic Minority Over-sampling Technique**

SMOTE (Synthetic Minority Over-sampling Technique) là một phương pháp cân bằng dữ liệu trong các bài toán phân loại mất cân bằng. Phương pháp này tạo thêm các mẫu tổng hợp từ lớp thiểu số (minority class) thay vì chỉ nhân bản ngẫu nhiên, giúp cải thiện hiệu suất của các mô hình học máy khi có sự mất cân bằng giữa các lớp.

**Cách hoạt động của SMOTE**

* **Chọn các điểm dữ liệu từ lớp thiểu số**: SMOTE bắt đầu bằng cách chọn ngẫu nhiên một điểm dữ liệu thuộc lớp thiểu số.
* **Xác định hàng xóm gần nhất**: Sau khi chọn một điểm từ lớp thiểu số, SMOTE tìm k hàng xóm gần nhất của điểm này trong cùng lớp thiểu số. Số lượng hàng xóm k thường được chọn là 5, nhưng có thể điều chỉnh tùy vào tình huống.
* **Tạo các điểm tổng hợp**: Một điểm ngẫu nhiên từ các hàng xóm gần nhất được chọn và SMOTE tạo một điểm tổng hợp mới bằng cách nội suy giữa điểm ban đầu và điểm hàng xóm. Quá trình nội suy được thực hiện bằng cách:

A group of symbols on a white background

Description automatically generated

trong đó  là một số ngẫu nhiên trong khoảng [0,1].

Kết quả là một điểm mới nằm trên đoạn thẳng nối giữa hai điểm, thuộc không gian của lớp thiểu số.

* **Lặp lại**: Quy trình trên được lặp lại cho đến khi đạt được số lượng mẫu mong muốn để cân bằng với lớp đa số.
* **Borderline-SMOTE**

Borderline-SMOTE (Borderline Synthetic Minority Over-sampling Technique) là một biến thể của thuật toán SMOTE (Synthetic Minority Over-sampling Technique), được sử dụng để cân bằng dữ liệu mất cân bằng trong các bài toán phân loại. Trong các bài toán này, số lượng mẫu của lớp thiểu số (minority class) thường nhỏ hơn rất nhiều so với lớp đa số (majority class), làm giảm độ chính xác và độ tin cậy của mô hình. Borderline-SMOTE tập trung vào việc tạo mẫu tổng hợp cho các điểm dữ liệu nằm gần biên giới giữa các lớp, nơi mà khả năng nhầm lẫn giữa các lớp thường cao hơn.

**Cách hoạt động của Borderline-SMOTE**

1. **Xác định các mẫu khó phân loại**: Đầu tiên, Borderline-SMOTE xác định các mẫu của lớp thiểu số nằm gần biên giới phân chia giữa lớp thiểu số và lớp đa số, cụ thể là các mẫu có nguy cơ bị nhầm lẫn cao hơn. Các mẫu này thường nằm gần các điểm của lớp đa số.
2. **Phân loại các điểm biên**: Những điểm dữ liệu thuộc lớp thiểu số gần kề với lớp đa số được chọn làm các điểm biên giới (borderline). Borderline-SMOTE xác định những điểm này bằng cách sử dụng một số tiêu chí như số lượng hàng xóm gần nhất thuộc lớp đa số. Những mẫu này có nguy cơ cao bị phân loại sai trong quá trình học của mô hình.
3. **Tạo các mẫu tổng hợp**: Sau khi xác định các điểm biên giới, Borderline-SMOTE tạo ra các điểm tổng hợp mới bằng cách nội suy giữa các điểm biên này với các hàng xóm gần nhất của chúng (thuộc lớp thiểu số). Điều này giúp tăng số lượng mẫu ở các khu vực quan trọng nhất (biên giới giữa các lớp) thay vì tăng số lượng mẫu trên toàn bộ không gian lớp thiểu số như SMOTE gốc.
4. **Chuẩn hóa Minmax**

Chuẩn hóa Min-Max (Min-Max Normalization) là một phương pháp được sử dụng trong tiền xử lý dữ liệu nhằm đưa các giá trị của thuộc tính về một khoảng nhất định, thường là từ 0 đến 1.

Giả sử có một tập dữ liệu với một thuộc tính x, và giá trị của thuộc tính này nằm trong khoảng từ ​ đến ​. Chuẩn hóa Min-Max đưa các giá trị của x về khoảng [0,1] bằng công thức:

A white background with black text

Description automatically generated

Chuẩn hóa Minimax đặc biệt hữu ích trong ML, nơi các đặc điểm có quy mô hoặc phạm vi lớn có thể chi phối quá trình phân tích và tạo ra kết quả không chính xác. Bằng cách áp dụng chuẩn hóa Minimax cho từng đặc điểm trong tập dữ liệu, chúng ta có thể đảm bảo rằng tất cả các đặc điểm đều có đóng góp như nhau cho quá trình phân tích, bất kể quy mô hoặc phạm vi ban đầu của chúng.

1. **Mối tương quan giữa các đặc trưng**

Ma trận tương quan là một công cụ thống kê được sử dụng để đo lường và thể hiện mức độ tương quan giữa các biến trong một tập dữ liệu.

Trong ma trận này:

* Màu đỏ đậm đại diện cho mối tương quan dương cao (gần +1), có nghĩa là khi một biến tăng, biến kia cũng có xu hướng tăng.
* Màu xanh đậm đại diện cho mối tương quan âm cao (gần -1), có nghĩa là khi một biến tăng, biến kia có xu hướng giảm.
* Màu xám hoặc màu nhạt gần như không có mối tương quan (giá trị gần 0).

Dưới đây là ma trận tương quan thể hiện mối tương quan giữa các đặc trưng của bộ dữ liệu:

A screenshot of a graph

Description automatically generated

* Các cặp như (Su, Sy) hoặc (A5, E) có vẻ có độ tương quan cao, thể hiện qua màu sắc đậm hơn. Điều này có thể cho thấy rằng khi một biến tăng, biến kia có xu hướng tăng (hoặc giảm) tương ứng.
* Các biến như category và material\_type có tương quan thấp với hầu hết các đặc tính vật liệu khác, điều này có thể ngụ ý rằng các thuộc tính vật lý và cơ học của vật liệu không bị ảnh hưởng nhiều bởi danh mục hoặc loại vật liệu được chỉ định.

1. **Triển khai trên tập dữ liệu**

**A black and yellow text

Description automatically generated**

* Loại bỏ các đặc trưng tên gọi

A close up of text

Description automatically generated

A screenshot of a computer code

Description automatically generated

* Phân loại đặc trưng “Material” từng mẫu thành các loại vật liệu như
  + Thép
  + Đồng
  + Bạc
  + Sắt
  + Nhôm
  + Magie

A screen shot of a computer program

Description automatically generated

* Phân loại đặc trưng Heat treatment thành các loại phương pháp xử lý nhiệt như:
  + Gia Công Nhiệt
  + Gia Công Lạnh
  + Đúc
  + Tôi bề mặt
  + Sản phẩm Hoàn Chỉnh
  + Phương pháp làm mát

A math equation with numbers and symbols

Description automatically generated with medium confidence

* Loại bỏ các biến có chất lượng thấp: Có nhiều hơn 50% giá trị Nah

**A close up of a text

Description automatically generated**

* Loại bỏ các đặc trưng thừa

**A close-up of a computer screen

Description automatically generated**

* Thêm các giá trị thiếu của đặc trưng A5 bằng giá trị trung vị của đặc trưng

**A black text on a white background

Description automatically generated**

* Chuyển các đặc trưng phân loại về dạng số

**A screen shot of a computer code

Description automatically generated**

* Loại bỏ các mẫu trùng lặp

A close-up of a computer

Description automatically generated

A screen shot of a computer code

Description automatically generated

* Áp dụng BorderlineSMOTE để xử lí sự mất cân bằng dữ liệu

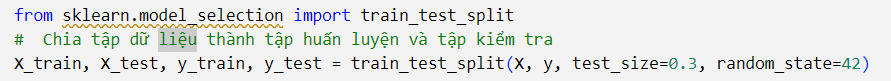
**A computer code with text

Description automatically generated with medium confidence**

* Áp dụng Chuẩn hóa Min-Max với những đặc trưng không phải đặc trưng phân loại

A close-up of a computer code

Description automatically generated

****

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

* Khởi chạy các mô hình

1. **Phân tích hiệu suất của các thuật toán**

Báo cáo sử dụng 2 thuật toán dựa trên ML, cụ thể là SVC và KNN để dự đoán vật liệu được sử dụng. Để kiểm tra hiệu quả của các thuật toán, báo cáo sử dụng bộ dữ liệu gồm 1553 mẫu vật liệu gồm vật liệu được sử dụng và vật liệu không được sử dụng. Các tập dữ liệu này đã cung cấp các điểm chuẩn quan trọng để đánh giá các thuật toán dựa trên ML. Tuy nhiên, chất lượng và tính đa dạng của dữ liệu đào tạo cũng như các tính năng và tham số của thuật toán có thể làm giảm khả năng xác định vật liệu. Các mô hình dựa trên ML được đề xuất đã chứng minh là một phương pháp hiệu quả để lựa chọn vật liệu được sử dụng.

* 1. **Support Vector Classification - SVC**

A chart of a confused matrix

Description automatically generated with medium confidence A graph with a line

Description automatically generated

A graph of a line graph

Description automatically generated with medium confidence

* + 1. Ma trận nhầm lẫn
    2. Biểu đồ ROC – AUC
    3. Biểu đồ Precison-Recall

Hiệu suất của mô hình SVC

| **Chỉ số** | **Precision** | **Recall** | **F1-score** | **Accuracy** |
| --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Giá trị** | 0.829 | 1.000 | 0.906 | 0.899 |

1. **Ma trận nhầm lẫn**:
   * Mô hình có tổng cộng 319 dự đoán chính xác cho nhãn 0 và 378 dự đoán chính xác cho nhãn 1.
   * Có 78 trường hợp nhãn 0 được dự đoán sai thành nhãn 1.
   * Không có trường hợp nào mà nhãn 1 bị dự đoán sai thành nhãn 0, cho thấy recall cho nhãn 1 là hoàn hảo.
2. **Biểu đồ ROC-AUC**:
   * Đường ROC có giá trị AUC là 0.933, cho thấy mô hình phân biệt tốt giữa hai lớp.
   * Đường cong gần như sát trục y, chứng tỏ mô hình có khả năng nhận diện tốt các mẫu dương tính thật so với mẫu âm tính.
3. **Biểu đồ Precision-Recall**:
   * Đường Precision-Recall cho thấy mô hình duy trì độ chính xác cao ngay cả khi recall tăng lên.
   * Với AP (Average Precision) là 0.82, biểu đồ này cho thấy mô hình hoạt động tốt cho cả hai chỉ số precision và recall.
4. **Hiệu suất tổng thể của mô hình**:
   * Precision: 0.829, cho thấy khoảng 82.9% các dự đoán dương tính là chính xác.
   * Recall: 1.000, tức là mô hình phát hiện hoàn toàn các trường hợp dương tính thật.
   * F1-score: 0.906, là một chỉ số cân bằng giữa precision và recall.
   * Accuracy: 0.899, cho thấy độ chính xác tổng thể của mô hình đạt 89.9%.

**Nhận xét tổng quát**

Mô hình SVC này có độ chính xác và recall rất cao, đặc biệt là không bỏ sót bất kỳ trường hợp dương tính nào, thể hiện qua recall đạt 1.000. Tuy nhiên, vẫn có một số dự đoán sai đối với nhãn 0, khiến cho precision thấp hơn. Điều này có thể được cải thiện thêm nếu tìm cách tối ưu hóa precision mà không làm giảm recall.

* 1. **K-nearest Neighbor - KNN**

A yellow and purple squares with numbers

Description automatically generated A graph with a line and a blue line

Description automatically generated

A graph of a line

Description automatically generated

* + 1. Ma trận nhầm lẫn
    2. Biểu đồ ROC – AUC
    3. Biểu đồ Precison-Recall

Hiệu suất của mô hình KNN

| **Chỉ số** | **Precision** | **Recall** | **F1-score** | **Accuracy** |
| --- | --- | --- | --- | --- |
| **Giá trị** | 0.932 | 0.974 | 0.952 | 0.952 |

1. **Ma trận nhầm lẫn**:
   * Mô hình dự đoán đúng 370 trường hợp nhãn 0 và 368 trường hợp nhãn 1.
   * Có 27 trường hợp nhãn 0 được dự đoán sai thành nhãn 1, và 10 trường hợp nhãn 1 bị dự đoán sai thành nhãn 0.
   * Điều này cho thấy mô hình có khả năng phân biệt hai lớp khá tốt, nhưng vẫn có một số dự đoán sai ở cả hai nhãn.
2. **Biểu đồ ROC-AUC**:
   * Đường ROC có giá trị AUC là 0.98, cho thấy khả năng phân biệt tốt giữa hai lớp.
   * Đường cong ROC gần sát trục y và đạt giá trị cao, minh chứng rằng mô hình có hiệu quả cao trong việc nhận diện các mẫu dương tính và âm tính.
3. **Biểu đồ Precision-Recall**:
   * Biểu đồ này cho thấy mô hình duy trì precision cao ngay cả khi recall tăng lên, và đạt AP (Average Precision) là 0.97.
   * Đường Precision-Recall tương đối cao và ổn định, cho thấy mô hình duy trì sự cân bằng giữa precision và recall tốt.
4. **Hiệu suất tổng thể của mô hình**:
   * Precision: 0.932, cho thấy 93.2% các dự đoán dương tính là chính xác.
   * Recall: 0.974, nghĩa là mô hình phát hiện được 97.4% các trường hợp dương tính.
   * F1-score: 0.952, là một chỉ số cân bằng giữa precision và recall.
   * Accuracy: 0.952, cho thấy độ chính xác tổng thể của mô hình đạt 95.2%.

**Nhận xét tổng quát**

Mô hình KNN có hiệu suất cao với độ chính xác, precision, recall, và F1-score đều đạt giá trị cao. Giá trị AUC cũng cao, cho thấy mô hình phân biệt tốt giữa hai lớp. Tuy nhiên, mô hình vẫn có một số nhầm lẫn, đặc biệt là đối với nhãn 0 khi bị nhầm thành nhãn 1. Điều này có thể được cải thiện thêm bằng cách điều chỉnh các tham số của KNN hoặc thử các kỹ thuật giảm lỗi dương tính và âm tính giả.

1. **Thảo luận**

Ứng dụng machine learning trong lựa chọn vật liệu sản xuất đã chứng minh khả năng tăng hiệu quả và tiết kiệm thời gian đáng kể. Theo một nghiên cứu gần đây, việc sử dụng ML giúp giảm đến 70% chi phí thử nghiệm và rút ngắn thời gian phát triển sản phẩm từ vài tháng xuống còn vài tuần. ML có thể phân tích nhanh chóng hàng triệu dữ liệu về đặc tính vật liệu như độ bền, độ dẻo, khả năng chịu nhiệt và chi phí, giúp chọn được vật liệu phù hợp một cách tối ưu mà không cần thử nghiệm thủ công trên từng loại. Chẳng hạn, các mô hình ML trong nghiên cứu vật liệu pin đã giảm thiểu thời gian tìm kiếm vật liệu hiệu quả từ 5-10 năm xuống còn khoảng 1-2 năm, đẩy nhanh tốc độ phát triển công nghệ năng lượng tái tạo. Nhờ đó, ML không chỉ giúp các công ty giảm chi phí mà còn nâng cao khả năng cạnh tranh, đồng thời mở ra tiềm năng khám phá các vật liệu mới phục vụ cho ngành sản xuất và công nghệ.

Dưới đây là bảng so sánh hiệu suất của hai mô hình SVC và KNN:

| **Chỉ số** | **SVC** | **KNN** |
| --- | --- | --- |
| **Precision** | 0.829 | 0.932 |
| **Recall** | 1.000 | 0.974 |
| **F1-score** | 0.906 | 0.952 |
| **Accuracy** | 0.899 | 0.952 |
| **AUC (ROC)** | 0.933 | 0.980 |
| **AP (Precision-Recall)** | 0.82 | 0.97 |

**Nhận xét chi tiết**

1. **Precision**: KNN có precision cao hơn (0.932 so với 0.829 của SVC). Điều này có nghĩa là mô hình KNN ít dự đoán nhầm các trường hợp dương tính giả hơn SVC.
2. **Recall**: SVC có recall cao hơn, đạt giá trị hoàn hảo 1.0, trong khi KNN đạt 0.974. Điều này cho thấy SVC phát hiện tất cả các trường hợp dương tính, nhưng KNN bỏ sót một số ít trường hợp dương tính.
3. **F1-score**: KNN có F1-score cao hơn (0.952 so với 0.906), chứng tỏ KNN duy trì sự cân bằng tốt hơn giữa precision và recall.
4. **Accuracy**: KNN có độ chính xác tổng thể cao hơn (0.952 so với 0.899 của SVC), cho thấy mô hình KNN thực hiện dự đoán chính xác hơn trên toàn bộ tập dữ liệu.
5. **AUC (ROC):** KNN có AUC cao hơn (0.980 so với 0.933 của SVC), điều này chỉ ra rằng KNN có khả năng phân biệt tốt hơn giữa hai lớp.
6. **AP (Precision-Recall):** KNN có AP cao hơn (0.97 so với 0.82 của SVC), cho thấy KNN duy trì độ chính xác cao ngay cả khi recall tăng lên.

**Kết luận**

Mô hình KNN có hiệu suất tổng thể tốt hơn so với SVC ở hầu hết các chỉ số, đặc biệt là về độ chính xác, precision, F1-score, và AUC. Tuy nhiên, SVC đạt recall cao hơn, nghĩa là nó phát hiện tất cả các mẫu dương tính mà không bỏ sót, điều này có thể phù hợp hơn trong các ứng dụng nhạy cảm khi cần hạn chế âm tính giả.