

HỌC VIỆN CÔNG NGHỆ BUUTURE CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1



**BÁO CÁO BÀI TẬP LỚN
Kho dữ liệu và khai phá dữ liệu**

Giảng viên: Nguyễn Quỳnh Chi

Nhóm bài tập: 09 - Nhóm lớp học: 02

Đề tài: So sánh 3 thuật toán KNN, Decision Tree, SVM

Phạm Quốc Khánh	B20DCCN378
Lê Trung Kiên	B20DCCN354
Phạm Đăng Quang	B20DCCN032
Nguyễn Việt Lương	B20DCCN413
Trần Quang Ngọc	B20DCCN474
Hoàng Bá Quốc	B20DCCN558

Hà Nội – 2024

I. Giới thiệu bài toán.....	4
1. Data mining.....	4
2. Quy trình của Data mining.....	4
3. Kỹ thuật khai phá dữ liệu phổ biến.....	5
II. KNN.....	7
1. Khái niệm.....	7
2. Các bước hoạt động của thuật toán KNN.....	8
3. Ưu và nhược điểm của kNN.....	10
III. Decision Tree.....	10
1. Giới thiệu thuật toán.....	10
2. Phép đo lựa chọn thuộc tính (Attribute Selection Measures).....	12
3. Cách hoạt động.....	14
4. Ưu điểm và nhược điểm.....	14
5. Các thuật toán xây dựng Decision Tree phổ biến.....	15
IV. SVM.....	17
1. Giới thiệu.....	17
2. Cách Hoạt Động của Thuật Toán.....	19
3. Ưu, Nhược điểm.....	20
4. Ứng dụng.....	21
V. Bộ dữ liệu và tiền xử lý dữ liệu.....	26
1. Mô tả dữ liệu.....	26
2. Tiền xử lý dữ liệu.....	28
3. EDA (trực quan hóa dữ liệu).....	30
4. Trích rút đặc trưng.....	1
VI. Xây dựng mô hình.....	1
1. KNN.....	1
2. Decision tree.....	1
3. SVM.....	1
VII. Chứng minh tính đúng đắn của thuật toán đã xây dựng.....	1
VIII. Kiểm nghiệm và đánh giá mô hình đã xây dựng.....	1
IX. Tài liệu tham khảo.....	1

Phạm Quốc Khánh	B20DCCN378	Tìm hiểu thuật toán Decision Tree, code thuật toán Decision Tree
Lê Trung Kiên	B20DCCN354	Tiền xử lý dữ liệu, trực quan hóa dữ liệu, code thuật toán Decision Tree
Phạm Đăng Quang	B20DCCN032	Nhóm trưởng, code thuật toán SVM, Decision Tree, đánh giá và kiểm nghiệm mô hình
Nguyễn Việt Lương	B20DCCN413	Tiền xử lý dữ liệu, trực quan hóa dữ liệu, code thuật toán KNN
Trần Quang Ngọc	B20DCCN474	Tìm hiểu thuật toán SVM, code thuật toán SVM
Hoàng Bá Quốc	B20DCCN558	Tìm hiểu thuật toán KNN, code thuật toán KNN, làm slide

I. Giới thiệu bài toán

1. Data mining

Data mining hay còn gọi là khai phá dữ liệu. Đây là quá trình tìm kiếm, khám phá và phân tích các mẫu tiềm ẩn, thông tin hữu ích từ trong cơ sở dữ liệu lớn. Nó giúp bạn hiểu rõ hơn về dữ liệu và tạo ra những thông tin giá trị từ những số liệu đó. Data mining sử dụng các phương pháp và thuật toán để phân tích dữ liệu. Mỗi phương pháp có ưu điểm và hạn chế riêng, bạn cần phải chọn phương pháp phù hợp với bài toán cụ thể mà bạn phải giải quyết.

- Lợi ích của data mining
 - + Tối ưu hóa chiến dịch tiếp thị
 - + Phát hiện gian lận và rủi ro
 - + Nâng cao dự đoán và dự báo
 - + Tối ưu hóa quy trình sản xuất và vận hành
- Ứng dụng của data mining trong các lĩnh vực khác nhau
 - + Kinh doanh và tiếp thị
 - + Sức khỏe và y tế
 - + Khoa học và nghiên cứu
 - + Tài chính và ngân hàng

2. Quy trình của Data mining

- Bước 1: Thu thập dữ liệu

Dữ liệu là yếu tố quan trọng trong quy trình data mining. Bước đầu tiên là thu thập dữ liệu từ các nguồn khác nhau như cơ sở dữ liệu, tệp tin, trang web hay nguồn mạng xã hội. Việc thu thập dữ liệu đòi hỏi sự chính xác và đầy đủ để đảm bảo rằng kết quả của quy trình data mining sẽ đáng tin cậy.

- Bước 2: Tiền xử lý dữ liệu

Sau khi thu thập dữ liệu, bước tiếp theo là tiền xử lý dữ liệu. Điều này bao gồm việc làm sạch và chuẩn hóa dữ liệu để loại bỏ các giá trị nhiễu, dữ liệu không hợp lệ hoặc trùng lặp. Tiền xử lý dữ liệu cũng có thể bao gồm việc chuyển đổi dữ liệu sang định dạng phù hợp để phân tích và khai thác.

- Bước 3: Khai phá dữ liệu

Sau khi tiền xử lý dữ liệu, bước tiếp theo là khai phá dữ liệu. Qua quá trình này, bạn sẽ cần áp dụng các kỹ thuật và phương pháp khai phá dữ liệu như phân tích đa biến, phân cụm, phân loại, gom nhóm, dự đoán và liên kết để tìm ra thông tin quan trọng và mô hình dữ liệu.

- Bước 4: Đánh giá và hiệu chỉnh

Sau khi khai phá dữ liệu, bước tiếp theo là đánh giá và hiệu chỉnh kết quả. Bạn cần đánh giá độ chính xác và độ tin cậy của kết quả data mining, đồng thời tìm cách cải thiện và hiệu chỉnh mô hình. Điều này đảm bảo rằng kết quả được đưa ra từ quy trình data mining là chính xác và đáng tin cậy.

- **Bước 5: Triển khai và áp dụng**

Cuối cùng, sau khi đánh giá và hiệu chỉnh, bạn sẽ triển khai và áp dụng kết quả của quy trình data mining. Điều này có thể bao gồm việc sử dụng các mô hình dữ liệu để dự đoán, phân loại hoặc gợi ý trong các ứng dụng thực tế. Triển khai và áp dụng hiệu quả kết quả data mining là mục tiêu cuối cùng để đảm bảo rằng quy trình này mang lại giá trị thực tế và ứng dụng trong thực tế.

3. Kỹ thuật khai phá dữ liệu phổ biến

- + **Kỹ thuật phân tích phân loại (Classification Analysis)**

Một kỹ thuật khai phá dữ liệu đầu tiên là kỹ thuật phân tích phân loại. Đây là một kỹ thuật cho phép phân loại một đối tượng vào một hoặc một số lớp cho trước.

Bạn có thể ứng dụng kỹ thuật này nhằm phân loại khách, hay các mặt hàng... bằng cách mô tả nhiều thuộc tính giúp phân loại các đối tượng vào một lớp cụ thể.

Chúng ta thường ứng dụng kỹ thuật khai thác dữ liệu này để lấy được những thông tin quan trọng từ dữ liệu và siêu dữ liệu. Do đó, trong quá trình phân tích, phân loại, chúng ta cần áp dụng các thuật toán khác nhau sao cho phù hợp với mục tiêu sử dụng.

Ví dụ: Email có thể sử dụng các thuật toán nhất định để mô tả một email thế nào là hợp pháp với một email spam. Hay các doanh nghiệp có thể ứng dụng kỹ thuật này nhằm phân loại khách hàng theo độ tuổi, hoặc đối tượng khác nhau....

- + **Kỹ thuật Association Rule Learning – Kỹ thuật khai phá dữ liệu**

Kỹ thuật Association Rule Learning là một kỹ thuật khai phá dữ liệu được sử dụng nhằm xác định mối quan hệ giữa các biến khác nhau trong cơ sở dữ liệu. Ngoài ra, kỹ thuật này còn được ứng dụng nhằm “giải nén: các mẫu ẩn trong dữ liệu.

Association Rule Learning rất hữu ích trong quá trình kiểm tra, dự đoán về các hành vi, do đó kỹ thuật này được ứng dụng phổ biến nhất trong ngành bán lẻ.

- + **Kỹ thuật Association Rule Learning**

Cùng với đó, các doanh nghiệp khi sử dụng kỹ thuật này còn có thể xác định được hành vi mua sắm của người tiêu dùng. Hỗ trợ phân tích dữ liệu trong giỏ hàng của khách hàng tiềm năng.

Do đó, với lĩnh vực công nghệ thông tin, các lập trình viên có thể sử dụng kỹ thuật này để xây dựng các chương trình Machine Learning.

+ Kỹ thuật phát hiện bất thường (Anomaly or Outlier Detection)

Về cơ bản, kỹ thuật khai phá dữ liệu phát hiện bất thường này được sử dụng để nhấn mạnh việc quan sát các mục dữ liệu trong bộ dữ liệu. Để từ đó tìm ra các tập dữ liệu không khớp với mẫu dự kiến. Sự bất thường ở đây có thể là độ lệch, sự khác thường, các nhiễu loạn và ngoại lệ khác...

Những sự bất thường được đánh giá là khá quan trọng, bởi nó có thể cung cấp một số thông tin cần thiết. Nó như một dữ liệu khác biệt so với mức trung bình chung trong một tập dữ liệu.

Có thể thấy, một cái gì đó khác thường đã xảy ra, và các nhà phân tích dữ liệu cần chú ý. Kỹ thuật này được ứng dụng trong đa dạng lĩnh vực khác nhau. Chẳng hạn như theo dõi sức khỏe, phát hiện các xâm nhập...

+ Kỹ thuật phân tích theo cụm (Clustering Analysis)

“Cụm” được hiểu với nghĩa là một nhóm các đối tượng dữ liệu. Các đối tượng có sự tương đồng nhau sẽ nằm trong cùng một cụm. Kết quả là các đối tượng tương đồng sẽ cùng trong một nhóm dữ liệu.

Kỹ thuật phân tích theo cụm này thường được ứng dụng để tạo hồ sơ khách hàng. Hay được ứng dụng trong phân chia phân khúc các đối tượng khách hàng trong lĩnh vực Marketing.

+ Kỹ thuật phân tích hồi quy (regression analysis) – data mining

Theo thuật ngữ thống kê, phân tích hồi quy nhằm giúp xác định và phân tích mối quan hệ giữa các biến. Do đó kỹ thuật phân tích hồi quy sẽ giúp bạn hiểu được những giá trị đặc trưng của sự thay đổi ở các biến phụ thuộc.

+ Kỹ thuật dự báo (prediction)

Trong các kỹ thuật khai phá dữ liệu, kỹ thuật dự báo được ứng dụng trong một số các trường hợp đặc biệt. Kỹ thuật này được sử dụng nhằm khám phá các mối quan hệ giữa các biến độc lập và phụ thuộc.

Chẳng hạn, bạn có thể ứng dụng kỹ thuật dự báo trong việc bán hàng, nhằm dự báo lợi nhuận trong tương lai. Nếu bán hàng là một biến độc lập, thì lợi nhuận có thể là một biến phụ thuộc. Khi đó chúng ta có thể vẽ đường cong quy hồi để dự đoán lợi nhuận hiệu quả.

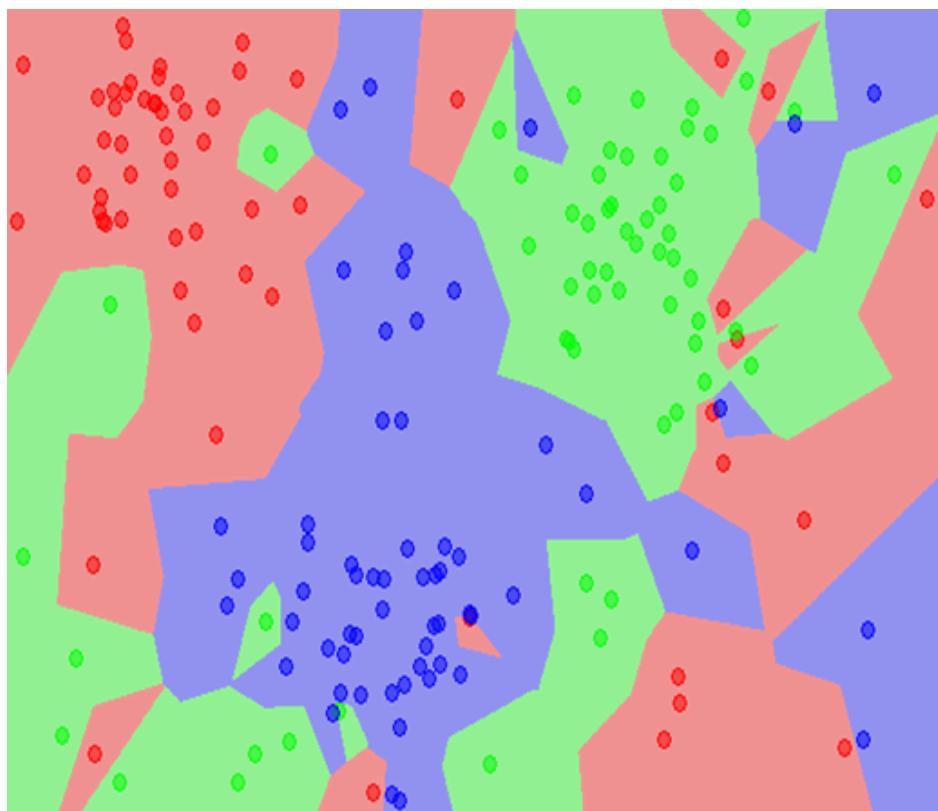
II. KNN

1. Khái niệm

K-nearest neighbor là một trong những thuật toán supervised-learning đơn giản nhất 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. K-nearest neighbor có thể áp dụng được vào cả hai loại của bài toán Supervised learning là Classification và Regression. KNN còn được gọi là một thuật toán Instance-based hay Memory-based learning.

Bản chất của k-NN chính là tính toán khoảng cách từ điểm cần xét đến k điểm lân cận, nên nếu các features có scale khác nhau, hoặc có đơn vị khác nhau, ta có thể thực hiện chuẩn hóa (normalize) dữ liệu để có thể đẩy mạnh hiệu suất của thuật toán. Tuỳ thuộc vào mục đích phân loại hay hồi quy mà đầu ra của k-NN sẽ khác nhau:

- Với bài toán phân loại, output của k-NN là lớp của bản ghi cần phân loại. Một bản ghi được xác định lớp (class) theo nguyên tắc biểu quyết đa số. Theo đó, lớp của bản ghi được xác định theo k bản ghi gần nhất với bản ghi đang được xét theo nguyên tắc số đông. k đảm bảo điều kiện là số nguyên dương, thường đủ nhỏ.



- Với bài toán hồi quy, output của k-NN chính là giá trị dự đoán của của hàm măt măt tại bản ghi đang được xét. Giá trị dự đoán này là trung bình nhãn của k mẫu gần nhất.

2. Các bước hoạt động của thuật toán KNN

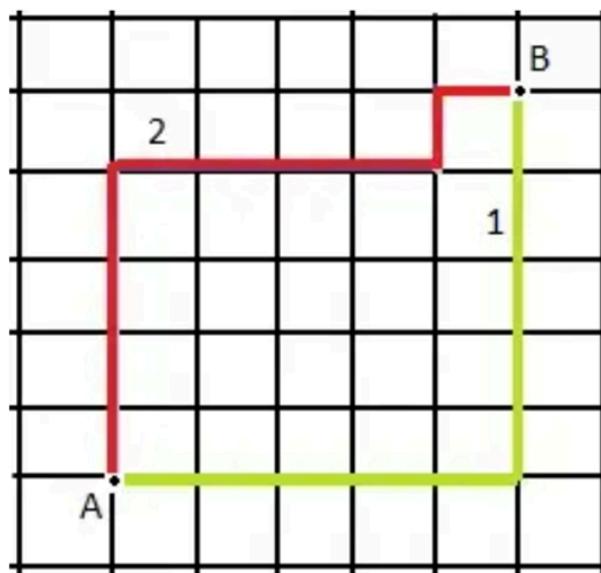
Về cơ bản, toàn bộ các bước xử lý của thuật toán này thực hiện trong quá trình inference. Với một điểm dữ liệu mới x cần dự đoán label l chúng ta sẽ làm như sau

- **Bước 1:** Tính khoảng cách từ x đến tất cả các điểm (x', l) trong toàn tập dữ liệu.
- **Bước 2:** Sắp xếp lại tập dữ liệu theo thứ tự khoảng cách từ nhỏ đến lớn
- **Bước 3:** Lọc ra Top k điểm có khoảng cách nhỏ nhất. Đếm số lần xuất hiện của mỗi class l_i trong top k . Giả sử là l_j có số lần xuất hiện lớn nhất trong top k
- **Bước 4:** Đưa ra kết luận x có nhãn là l_j

* Độ đo khoảng cách:

Trong k-NN, ta có thể sử dụng nhiều loại khoảng cách khác nhau, nhưng 3 loại khoảng cách thường gặp chính là khoảng cách Manhattan, khoảng cách Euclid và khoảng cách Cosine.

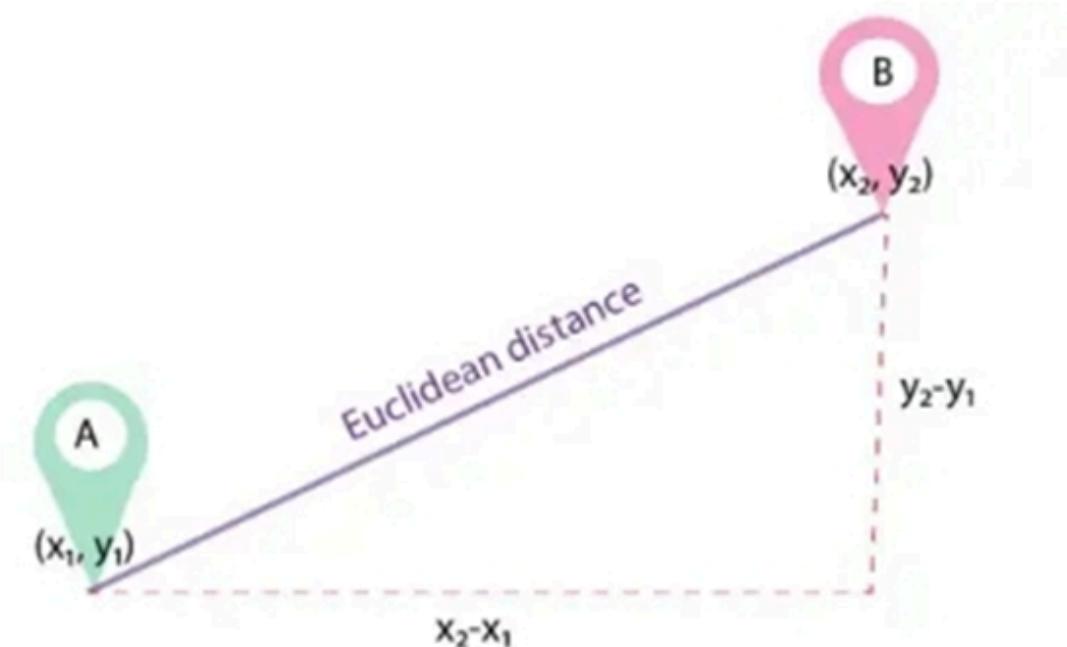
- Khoảng cách Manhattan:



$$d = \sum_{i=1}^n \|x_i - y_i\|$$

Khoảng cách Manhattan là tổng khoảng cách các thành phần tọa độ giữa 2 điểm.

- Khoảng cách Euclid:



$$d = \sqrt{\left(\sum_{i=1}^n (x_i - y_i)^2\right)}$$

Khoảng cách Euclid là khoảng cách đường chim bay giữa 2 điểm.

- Khoảng cách Cosine:

Ta có độ đo tương đồng Cosine được tính theo công thức sau:

$$\cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{||\vec{a}|| ||\vec{b}||}$$

* Chọn số k phù hợp

Giá trị k cho thuật toán k-NN quy định thuật toán phải quan sát k số điểm lân cận xung quanh điểm được xét nhằm phân loại. Nếu $k=1$, ta gắn nhãn điểm được xét trùng với điểm gần nhất trong tập train. Chọn k là một công việc quan trọng, do các giá trị khác nhau của k có thể dẫn tới hiện tượng over hoặc underfit.

Thường, lựa chọn số k là phụ thuộc vào dữ liệu. Thông thường, giá trị k lớn đồng nghĩa với việc ta giảm thiểu được nhiễu khi thực hiện phân lớp, nhưng sẽ khiến ranh giới giữa các lớp trở nên mờ hơn. Ta có thể chọn được số k tốt thông qua các thủ thuật heuristic, cross - validation để có thể lọc k.

Độ chính xác của thuật toán k-NN có thể bị giảm đột ngột nếu có các đặc trưng nhiều hoặc đặc trưng không liên quan, hoặc scale của các features không đồng đều, không tương xứng với mức độ tương quan với nhãn. Khi làm việc với bài toán phân lớp nhị phân, ta nên để k lẻ để tránh trường hợp hoà.

3. Ưu và nhược điểm của kNN

a. Ưu điểm

- Đơn giản và dễ giải thích
- Không dựa trên bất kỳ giả định nào, vì thế nó có thể được sử dụng trong các bài toán phi tuyến tính.
- Hoạt động tốt trong trường hợp phân loại với nhiều lớp
- Sử dụng được trong cả phân loại và hồi quy

b. Nhược điểm

- Trở nên rất chậm khi số lượng điểm dữ liệu tăng lên vì mô hình cần lưu trữ tất cả các điểm dữ liệu.
- Tốn bộ nhớ
- Nhạy cảm với các dữ liệu bất thường (nhiễu)

III. Decision Tree

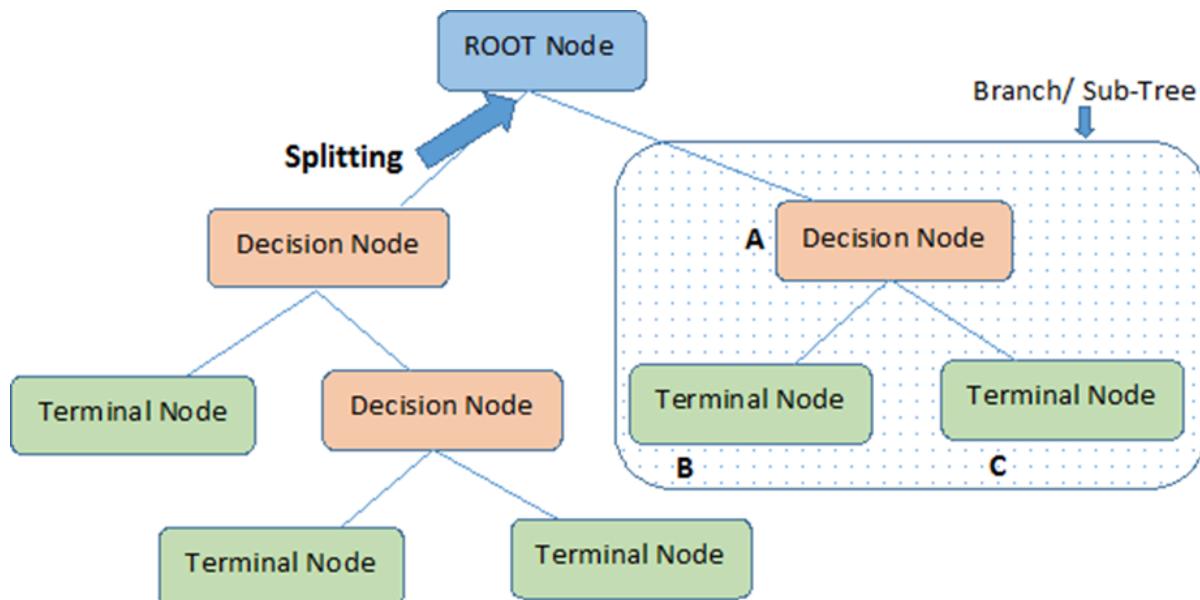
1. Giới thiệu thuật toán

Thuật toán cây quyết định là một phần của họ các thuật toán học có giám sát. Khác với các thuật toán học có giám sát khác, thuật toán cây quyết định có thể giải quyết các vấn đề hồi quy và phân loại. Một cây quyết định là

một biểu đồ giống như cây với các nút đại diện cho nơi chúng ta chọn một thuộc tính và đặt một câu hỏi; các cạnh đại diện cho câu trả lời cho câu hỏi, và các lá đại diện cho đầu ra thực tế hoặc nhãn lớp. Chúng được sử dụng trong quyết định phi tuyến tính với một bề mặt quyết định tuyến tính đơn giản.

Mục tiêu của việc sử dụng một cây quyết định là tạo ra một mô hình huấn luyện có thể sử dụng để dự đoán lớp hoặc giá trị của biến mục tiêu bằng cách học các quy tắc quyết định đơn giản suy ra từ dữ liệu trước đó (dữ liệu huấn luyện).

Trong cây quyết định, chúng ta bắt đầu từ gốc của cây để dự đoán một nhãn lớp cho một bản ghi. Chúng ta so sánh các giá trị của thuộc tính gốc với thuộc tính của bản ghi. Dựa trên sự so sánh đó, chúng ta đi theo nhánh tương ứng với giá trị đó và di chuyển đến nút tiếp theo.



Note:- A is parent node of B and C.

Thuật ngữ:

- Root Node: Đại diện cho toàn bộ tập dữ liệu hoặc mẫu và được chia thành hai hoặc nhiều tập con đồng nhất.
- Leaf/ Terminal Node: Node không được chia là node lá hoặc node cuối.
- Decision Node: Khi một node con chia thành các node con tiếp theo, thì nó được gọi là node quyết định.
- Branch/ Sub-Tree: Một phần phụ của cây hoặc cây con được gọi là nhánh hoặc cây con.
- Parent và Child Node: Một node, được chia thành các node con, được gọi là node cha của các node con đó, trong khi các node con là con của node cha.

- Splitting: Đây là quá trình chia một node thành hai hoặc nhiều node con.
- Pruning: Cắt tỉa là khi chọn lọc loại bỏ các nhánh từ cây. Mục tiêu là loại bỏ các nhánh không mong muốn, cải thiện cấu trúc của cây và định hướng sự phát triển mới và lành mạnh.

2. Phép đo lựa chọn thuộc tính (Attribute Selection Measures)

ASM được sử dụng để đánh giá mức độ hữu ích của các thuộc tính khác nhau trong việc phân chia tập dữ liệu. Mục tiêu của ASM là xác định thuộc tính nào sẽ tạo ra các tập con dữ liệu đồng nhất nhất sau khi phân chia, từ đó tối đa hóa information gain thu được.

Quá trình xây dựng cây quyết định thông qua ASM thường bắt đầu với việc chọn thuộc tính tốt nhất để phân chia dữ liệu. Thuộc tính này được chọn dựa trên mức độ giảm entropy hoặc sự tăng thông tin trong các tập con dữ liệu được phân chia. Sau đó, quá trình này được lặp lại đệ quy trên mỗi tập con, cho đến khi mỗi tập con chỉ chứa các mẫu thuộc cùng một lớp hoặc không còn có thêm giá trị nào để dự đoán.

Entropy là một trong những thước đo phổ biến nhất được sử dụng trong việc tính toán ASM. Entropy đo lường mức độ không chắc chắn hoặc ngẫu nhiên trong tập dữ liệu. Cụ thể, entropy cho một tập hợp con của tập dữ liệu gốc với K số lớp có thể được định nghĩa như sau:

$$\text{Entropy}(S) = - \sum_{i=1}^K p(i) \log_2(p(i))$$

Ở đây:

- S là mẫu tập dữ liệu.
- K là lớp cụ thể từ K lớp.
- $p(i)$ là tỷ lệ các điểm dữ liệu thuộc lớp i trên tổng số điểm dữ liệu trong mẫu tập dữ liệu S .

Information Gain (IG):

Information Gain là một khái niệm quan trọng trong việc xây dựng cây quyết định, đo lường mức độ giảm entropy hoặc phương sai do việc chia tách tập dữ liệu dựa trên một thuộc tính cụ thể. Nó được sử dụng để đánh giá tính hữu ích của một thuộc tính trong việc phân loại dữ liệu.

Thông tin thu được từ một thuộc tính càng cao thì tính năng dự đoán biến mục tiêu càng có giá trị. Information Gain của một thuộc tính A đối với tập dữ liệu S được tính bằng cách đo lường mức giảm entropy của tập dữ liệu sau khi phân chia dựa trên thuộc tính A.

Cụ thể, Information Gain được tính bằng công thức sau:

$$IG(A) = |H| - \sum_v \frac{|H_v|}{|S|} \times H_v$$

- $|H|$ là entropy của mẫu dữ liệu S trước khi phân chia.
- $|H_v|$ là entropy của tập con S có giá trị v cho thuộc tính A .
- $|S|$ là số lượng mẫu dữ liệu trong tập dữ liệu S .

Information Gain đo lường mức giảm entropy đạt được bằng cách phân vùng tập dữ liệu trên thuộc tính A . Thuộc tính có Information Gain cao nhất sẽ được chọn làm tiêu chí phân tách để xây dựng cây quyết định.

Information Gain được sử dụng trong cả cây quyết định phân loại và hồi quy. Trong phân loại, entropy được sử dụng làm thước đo độ tinh khiết, trong khi trong hồi quy, phương sai được sử dụng làm thước đo độ tinh khiết. Tuy nhiên, quá trình tính toán Information Gain vẫn giống nhau trong cả hai trường hợp, chỉ có sự thay đổi về thước đo độ tinh khiết được sử dụng trong công thức.

Gini Impurity:

Gini Impurity là một thước đo quan trọng được sử dụng trong việc xây dựng cây quyết định, cũng giống như entropy. Gini Impurity được sử dụng để đánh giá mức độ hỗn loạn hoặc không chắc chắn trong tập dữ liệu khi phân loại.

Gini Impurity đo lường mức độ hỗn loạn trong dữ liệu. Nó có giá trị từ 0 đến 1 :

$$\text{Gini}(D) = 1 - \sum_{i=1}^m p_i^2$$

- Giá trị 0 đại diện cho trường hợp tất cả các quan sát thuộc về cùng một lớp, tức là không có sự hỗn loạn.
- Giá trị 1 đại diện cho trường hợp tất cả các quan sát được phân bố ngẫu nhiên trong các lớp, tức là hỗn loạn tối đa.

Trong quá trình xây dựng cây quyết định, chúng ta muốn chọn thuộc tính và giá trị phân chia để giảm Gini Impurity của các tập con kết quả. Mục tiêu là chọn phân chia làm giảm Gini Impurity một cách tối đa, từ đó tạo ra các tập con đồng nhất hơn và tăng độ chính xác của cây quyết định.

Cũng giống như entropy, Gini Impurity cũng được sử dụng để đánh giá chất lượng của sự phân chia thuộc tính. Thuộc tính và giá trị phân chia tạo ra sự giảm Gini Impurity lớn nhất sẽ được chọn để phân loại dữ liệu.

3. Cách hoạt động

Trong cây quyết định, để dự đoán lớp của tập dữ liệu đã cho, thuật toán bắt đầu từ nút gốc của cây. Thuật toán này so sánh các giá trị của thuộc tính gốc với thuộc tính bản ghi (tập dữ liệu thực) và dựa trên sự so sánh, đi theo nhánh và nhảy đến nút tiếp theo.

Đối với nút tiếp theo, thuật toán lại so sánh giá trị thuộc tính với các nút con khác và di chuyển xa hơn. Nó tiếp tục quá trình cho đến khi nó đạt đến nút lá của cây. Quy trình hoàn chỉnh có thể được hiểu rõ hơn bằng cách sử dụng thuật toán dưới đây:

Bước 1: Bắt đầu cây với nút gốc (Đặt tên: S), nút này chứa tập dữ liệu hoàn chỉnh.

Bước 2: Tìm thuộc tính tốt nhất trong tập dữ liệu bằng cách sử dụng Phép đo lựa chọn thuộc tính (ASM).

Bước 3: Chia S thành các tập con chứa các giá trị có thể có cho các thuộc tính tốt nhất.

Bước 4: Tạo nút cây quyết định chứa thuộc tính tốt nhất.

Bước 5: Tạo một cách đệ quy cây quyết định mới bằng cách sử dụng các tập con của tập dữ liệu đã tạo ở bước -3. Tiếp tục quá trình này cho đến khi đạt đến một giai đoạn mà bạn không thể phân loại thêm các nút và được gọi là nút cuối cùng là nút lá.

4. Ưu điểm và nhược điểm

Ưu điểm:

1. Chi phí xây dựng không đắt: Xây dựng cây quyết định không yêu cầu nhiều tài nguyên tính toán và thời gian. Thuật toán có thể được triển khai một cách tương đối đơn giản và hiệu quả.

- Phân loại nhanh chóng: Sau khi cây quyết định đã được xây dựng, việc phân loại các bản ghi mới (chưa biết) trở nên rất nhanh chóng. Thời gian phân loại thường rất ngắn, đặc biệt đối với các cây quyết định nhỏ và đơn giản.
- Dễ dàng giải thích: Cây quyết định tạo ra một mô hình dễ hiểu, có thể được giải thích một cách trực quan. Người dùng có thể theo dõi các quyết định và luồng logic trong cây để hiểu ý nghĩa của quyết định phân loại.
- Độ chính xác tương đối: Cây quyết định có thể đạt được độ chính xác so sánh tương đương với các kỹ thuật phân loại khác, đặc biệt là trên các tập dữ liệu đơn giản. Với các tập dữ liệu phức tạp hơn, việc sử dụng các biến thể của cây quyết định hoặc kỹ thuật kết hợp có thể cải thiện độ chính xác.

Nhược điểm:

- Dễ bị overfitting: Cây quyết định phức tạp có xu hướng quá mức và không tổng quát hóa tốt cho dữ liệu mới. Kịch bản này có thể tránh được thông qua quá trình cắt tỉa trước hoặc sau cắt tỉa. Việc cắt tỉa trước sẽ tạm dừng sự phát triển của cây khi không có đủ dữ liệu trong khi sau khi cắt tỉa sẽ loại bỏ các cây phụ có dữ liệu không đầy đủ sau khi xây dựng cây.
- Các công cụ ước tính phương sai cao: Các biến thể nhỏ trong dữ liệu có thể tạo ra một cây quyết định rất khác. Tính tổng hợp, hoặc tính trung bình của các ước tính, có thể là một phương pháp giảm phương sai của cây quyết định. Tuy nhiên, cách tiếp cận này bị hạn chế vì nó có thể dẫn đến các yếu tố dự báo có tương quan cao.
- Tốn kém hơn: Do cây quyết định có cách tiếp cận tìm kiếm tham lam trong quá trình xây dựng, chúng có thể tốn kém hơn để đào tạo so với các thuật toán khác.
- Không được hỗ trợ đầy đủ trong scikit-learning: Scikit-learning là một thư viện máy học phổ biến dựa trên Python. Mặc dù thư viện này có mô-đun Cây quyết định, việc triển khai hiện tại không hỗ trợ các biến phân loại.

5. Các thuật toán xây dựng Decision Tree phổ biến

a. Thuật toán ID3

- Thuật toán ID3 sử dụng các chỉ số liên quan đến lý thuyết thông tin, đặc biệt là entropy và information gain, để đưa ra quyết định trong quá trình xây dựng cây.

Information Gain và Attribute Selection

Thuật toán ID3 sử dụng một độ đo về độ không thuần khiết, chẳng hạn như entropy hoặc Gini Impurity, để tính toán information gain của mỗi thuộc tính. Entropy là một đại lượng đo lường mức độ không gian không gian dữ liệu trong tập dữ liệu. Một tập dữ liệu có entropy cao là một tập dữ liệu trong đó các điểm dữ liệu được phân bố đều qua các danh mục khác nhau. Một tập dữ liệu có entropy thấp là một tập dữ liệu trong đó các điểm dữ liệu tập trung trong một hoặc một số ít các danh mục.

$$H(S) = -(P_i * \log_2(P_i))$$

- Nếu entropy thấp, dữ liệu được hiểu rõ; nếu entropy cao, cần thêm thông tin. Tiền xử lý dữ liệu trước khi sử dụng ID3 có thể cải thiện độ chính xác. Tóm lại, ID3 cố gắng giảm sự không chắc chắn và đưa ra quyết định thông thái bằng cách chọn những thuộc tính mang lại những thông tin quan trọng nhất trong tập dữ liệu.
- Information gain đánh giá mức độ thông tin có giá trị mà một thuộc tính có thể cung cấp. Chúng ta chọn thuộc tính có information gain cao nhất, cho thấy khả năng của nó đóng góp nhiều nhất vào việc hiểu dữ liệu. Nếu information gain cao, điều đó ngụ ý rằng thuộc tính đó mang lại những cái nhìn quan trọng. ID3 hoạt động như một thám tử, lựa chọn những thuộc tính để tối đa hóa information gain ở mỗi bước. Phương pháp này nhằm mục tiêu tối thiểu hóa sự không chắc chắn và đưa ra những quyết định thông thái, có thể được cải thiện thêm bằng cách tiền xử lý dữ liệu.

- $|S|$ là tổng số bản ghi trong tập dữ liệu .
- $|S_v|$ là số lượng bản ghi trong tập dữ liệu mà thuộc tính D có giá trị v.
- $H(S)$ là độ không chắc chắn (entropy) của tập dữ liệu.

b. Thuật toán C4.5

Thuật toán C4.5 là thuật toán cải tiến của ID3.

Trong thuật toán ID3, Information Gain được sử dụng làm độ đo. Tuy nhiên, phương pháp này lại ưu tiên những thuộc tính có số lượng lớn các giá trị mà ít xét tới những giá trị nhỏ hơn. Do vậy, để khắc phục nhược điểm trên, ta sử dụng độ đo Gain Ratio (trong thuật toán C4.5) như sau:

Đầu tiên, ta chuẩn hoá information gain với trị thông tin phân tách (split information):

$$Gain\ Ratio = \frac{Information\ Gain}{Split\ Info}$$

Trong đó: Split Info được tính như sau:

$$-\sum_{i=1}^n Di \log_2 Di$$

Giả sử phân chia biến thành n nút cón và Di đại diện cho số lượng bản ghi thuộc nút đó. Do đó, hệ số Gain Ratio sẽ xem xét được xu hướng phân phối khi chia cây.

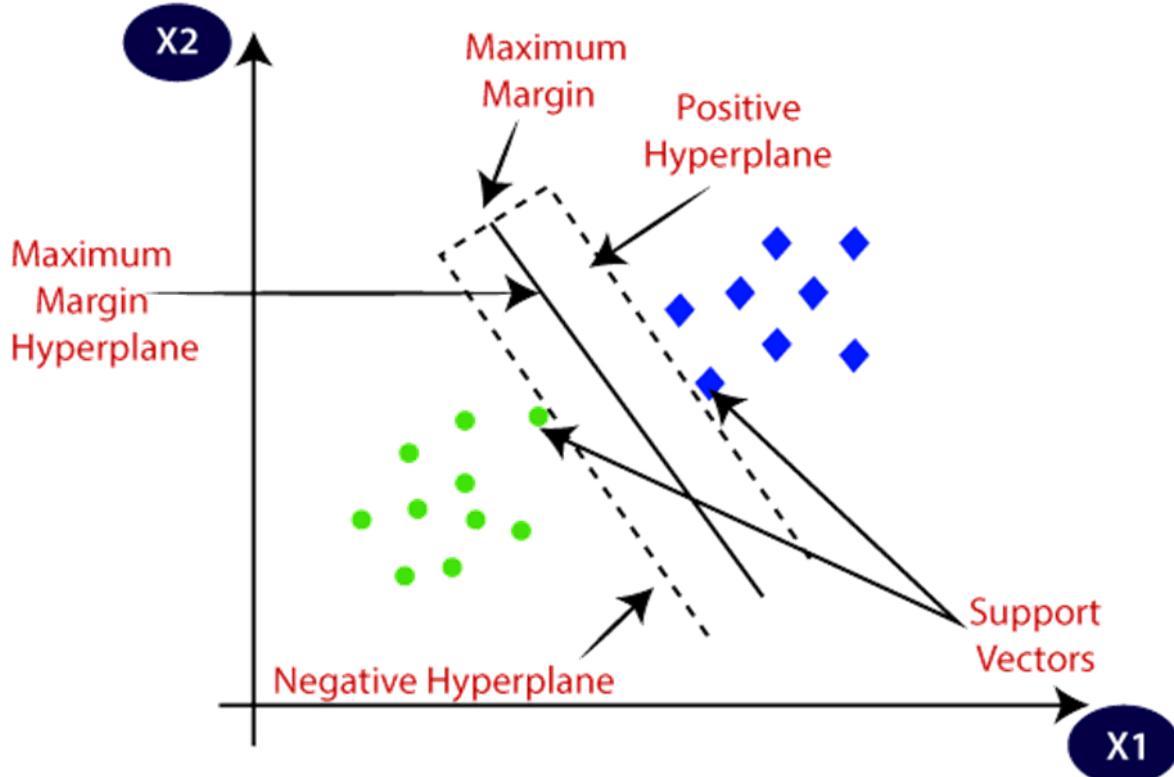
IV. SVM

1. Giới thiệu

Support Vector Machine là một trong những thuật toán Học có giám sát phổ biến nhất, được sử dụng cho cả các vấn đề Phân loại và Hồi quy. Tuy nhiên, chủ yếu, nó được sử dụng cho các vấn đề Phân loại trong Machine Learning.

Mục tiêu của thuật toán SVM là tạo ra đường chia hoặc ranh giới quyết định tốt nhất có thể phân loại không gian n-chiều thành các lớp để chúng ta có thể dễ dàng đưa điểm dữ liệu mới vào loại chính xác trong tương lai. Ranh giới quyết định tốt nhất này được gọi là siêu phẳng.

SVM chọn các điểm/vector cực đại giúp tạo ra siêu phẳng. Các trường hợp cực đại này được gọi là các vector hỗ trợ, và do đó thuật toán được gọi là Máy Vector Hỗ Trợ. Hãy xem biểu đồ dưới đây trong đó có hai danh mục khác nhau được phân loại bằng đường chia quyết định hoặc siêu phẳng:



-Support Vectors: Đây là những điểm gần nhất với siêu phẳng. Một đường chia sẽ được định nghĩa thông qua những điểm dữ liệu này.

-Margin: đây là khoảng cách giữa siêu phẳng và các quan sát gần nhất với siêu phẳng (Support vector). Trong SVM, biên lớn được coi là một biên tốt. Có hai loại biên: biên cứng và biên mềm. Tôi sẽ nói thêm về hai loại này trong phần sau.

Ví dụ:

SVM có thể được hiểu thông qua ví dụ mà chúng ta đã sử dụng trong bộ phân loại KNN.

Giả sử chúng ta nhìn thấy một con mèo lạ mà cũng có một số đặc điểm của chó, vì vậy nếu chúng ta muốn một mô hình có thể xác định chính xác nó là mèo hay chó, một mô hình như vậy có thể được tạo ra bằng cách sử dụng thuật toán SVM.

Đầu tiên, chúng ta sẽ huấn luyện mô hình của mình với rất nhiều hình ảnh của mèo và chó để nó có thể học về các đặc điểm khác nhau của mèo và chó, sau đó chúng ta thử nghiệm nó với sinh vật lạ này. Do đó, khi các

Support vector tạo ra ranh giới quyết định giữa hai dữ liệu này (mèo và chó) và chọn các trường hợp cực đại (Support vector), nó sẽ nhìn thấy trường hợp cực đại của mèo và chó. Dựa trên các vector hỗ trợ, nó sẽ phân loại nó là mèo.

2. Cách Hoạt Động của Thuật Toán

Mục tiêu của SVM (Support Vector Machine) là tìm một siêu phẳng (hyperplane) để phân chia dữ liệu thành hai loại cho bài toán phân loại nhị phân.

Các Điểm Chính:

- Siêu Phẳng (Hyperplane)

Siêu phẳng là một không gian con có số chiều ít hơn một so với không gian chứa nó. Chẳng hạn, trong một không gian n chiều, siêu phẳng có (n-1) chiều.

Đối với SVM, mục tiêu là tối đa hóa khoảng cách (margin) giữa hai lớp. Phương trình của siêu phẳng có thể được biểu diễn như sau:

$$wx - b = 0$$

trong đó:

w là vector trọng số,

x là vector đặc trưng, và

b là hệ số bias.

Để phân loại các điểm dữ liệu, ta sử dụng các điều kiện sau:

- Đối với một điểm dữ liệu có nhãn $y=1$, ta muốn $wx - b \geq 1$
- Đối với một điểm dữ liệu có nhãn $y=-1$ ta muốn $wx - b \leq -1$

Nói chung, ta hướng tới: $y(wx - b) \geq 1$ đối với tất cả các điểm dữ liệu.

- Khoảng Cách (Margin):

Khoảng cách được định nghĩa là khoảng cách giữa siêu phẳng và các điểm dữ liệu gần nhất từ cả hai lớp, được gọi là các vector hỗ trợ (support vectors). Khoảng cách càng lớn, khả năng tổng quát hóa của bộ phân loại càng tốt.

c. Phương Trình Gradient

Các gradient của hàm chi phí đối với trọng số W và hệ số bias b được sử dụng để cập nhật các tham số này trong quá trình tối ưu hóa. Hàm chi phí bao gồm một điều khoản để tối đa hóa khoảng cách (phụ thuộc vào W) và một điều khoản điều chỉnh để ngăn ngừa overfitting.

Đối với một điểm dữ liệu x_i có nhãn y_i thỏa mãn $y_i(w \cdot x_i - b) \geq 1$ (được phân loại đúng và nằm ngoài khoảng cách), các gradient là:

$$\begin{aligned}\frac{\partial J}{\partial w} &= 2\lambda w \\ \frac{\partial J}{\partial b} &= 0\end{aligned}$$

Đối với một điểm dữ liệu x_i có nhãn y_i không thỏa mãn $y_i(w \cdot x_i - b) \geq 1$ (được phân loại sai hoặc nằm ngoài khoảng cách), các gradient là:

$$\begin{aligned}\frac{\partial J}{\partial w} &= 2\lambda w - y_i x_i \\ \frac{\partial J}{\partial b} &= -y_i\end{aligned}$$

Ở đây, J là hàm chi phí của SVM, λ là tham số điều chỉnh (điều khiển sự đánh đổi giữa việc tăng kích thước biên và đảm bảo rằng x_i nằm ở phía đúng của biên), W là vector trọng số, b là thuật ngữ bias, x_i là điểm dữ liệu thứ i , và y_i là nhãn tương ứng. Thiết lập này đảm bảo rằng bộ phân loại không chỉ tìm ra một siêu phẳng phân chia (nếu có) mà còn kiểm tra siêu phẳng đó làm tăng độ rộng của biên giữa các lớp, điều quan trọng trong chiến lược phân loại của SVM.

3. Ưu, Nhược điểm

a. Ưu điểm:

- Khả năng điều chuẩn: SVM có tính năng Điều Chuẩn L2. Vì vậy, nó có khả năng tổng quát tốt giúp ngăn chặn hiện tượng quá mức phù hợp (overfitting).

- Xử lý dữ liệu phi tuyến: SVM có thể hiệu quả xử lý dữ liệu phi tuyến bằng cách sử dụng Kỹ thuật Kernel.
- Giải quyết cả các vấn đề Phân loại và Hồi quy: SVM có thể được sử dụng để giải quyết cả các vấn đề phân loại và hồi quy. SVM được sử dụng cho các vấn đề phân loại trong khi SVR (Hồi quy Vector Hỗ Trợ) được sử dụng cho các vấn đề hồi quy.
- Ôn định: Một thay đổi nhỏ đối với dữ liệu không ảnh hưởng lớn đến siêu phẳng và do đó SVM. Vì vậy mô hình SVM ổn định.

b. Nhược điểm:

- Lựa chọn hàm Kernel phù hợp là khó khăn: Lựa chọn hàm Kernel phù hợp (để xử lý dữ liệu phi tuyến) không phải là một nhiệm vụ dễ dàng. Điều này có thể là một công việc phức tạp và rắc rối. Trong trường hợp sử dụng một hàm Kernel có số chiều cao, bạn có thể tạo ra quá nhiều vector hỗ trợ gây giảm tốc độ huấn luyện đáng kể.
- Yêu cầu về bộ nhớ lớn: Độ phức tạp của thuật toán và yêu cầu về bộ nhớ của SVM rất cao. Bạn cần nhiều bộ nhớ vì bạn phải lưu trữ tất cả các vector hỗ trợ trong bộ nhớ và số lượng này tăng đột ngột với kích thước tập dữ liệu huấn luyện.
- Yêu cầu Scale Đặc trưng: Người ta phải thực hiện việc Scale Đặc trưng của biến trước khi áp dụng SVM.
- Thời gian huấn luyện dài: SVM mất thời gian huấn luyện lâu trên các tập dữ liệu lớn.
- Khó hiểu: Mô hình SVM khó hiểu và khó giải thích đối với con người, không giống như Cây Quyết định.

4. Ứng dụng

a. Inverse Geosounding Problem(Vấn đề nghịch đảo trong địa vật lý)

-Vấn đề địa lý là một trong những ứng dụng quan trọng của SVM giúp xác định cấu trúc lớp của hành tinh. Điều này đòi hỏi các giải pháp cho các vấn đề nghịch đảo đa dạng. Trong trường hợp dữ liệu điện từ, chúng ta sử dụng một hàm tuyến tính cho vấn đề và chúng ta sử dụng thuật toán học SVM cho các mô hình.

-Các mô hình được phát triển ở đây bằng các Kỹ thuật Lập trình tuyến tính. Nhưng kích thước của không gian trong đó mô hình được phát triển có thể nhỏ ở đây, vì kích thước của vấn đề cũng nhỏ.

b. Seismic Liquefaction Potential

-Ở đây, SVM được coi là một thuật toán Học Máy tuyệt vời vì độ chính xác của kết quả là rất cao. Chúng ta có hai vấn đề là SPT (Thử Nghiệm Thấm Tháp Tiêu Chuẩn) và CPT (Thử Nghiệm Thấm Côn). Trong cả hai thử nghiệm này, chúng ta

kiểm tra sự xuất hiện và không xuất hiện của hiện tượng lỏng đất. Ở đây, chúng ta sử dụng các bộ dữ liệu thực địa.

-Độ chính xác đạt được với các bộ dữ liệu sử dụng SVM lần lượt là 96% và 97%. Ở đây, SVM giúp xây dựng một mô hình cho các kết hợp phức tạp như các thông số đất và tiềm năng lỏng đất.

c. Protein Fold and Remote Homology Detection

-Phát hiện đồng dạng từ xa tận dụng rất tốt SVM. Trong trường hợp này, điều này phụ thuộc vào cách các chuỗi protein được mô hình hóa. Chúng ta sử dụng các phương pháp khác nhau để giải quyết các hàm nhân đang được sử dụng.

-Các hàm nhân giúp tìm sự tương đồng giữa các chuỗi protein khác nhau. Sự thật là, nhiều lần các hàm nhân này tạo ra kết quả tốt hơn nhiều so với nhiều kỹ thuật dựa trên SVM. Hàm nhân thực sự là một phần trong SVM chính.

-Theo cách này, SVM đang ảnh hưởng đến các lĩnh vực như sinh học tính toán

d. Phân loại dữ liệu sử dụng SSVM (Data Classification using SSVM)

Ở đây, SVM giúp giải quyết nhiều vấn đề dựa trên toán học. Chúng ta sử dụng các phương pháp làm trơn khác nhau ở đây để giải quyết các vấn đề toán học khác nhau. Có một sự thay đổi nhỏ, chúng ta sẽ không sử dụng phương pháp SVM thông thường ở đây. Chúng ta sẽ sử dụng SSVM hoặc Smooth SVM.

SSVM làm cho việc phân loại trở nên dễ dàng hơn vì nó có tính chất lồi mạnh (một tính chất toán học). Ngoài ra, việc sử dụng hàm nhân ở đây là tùy ý. Đối với vấn đề tối ưu hóa không ràng buộc của SSVM, chúng ta sử dụng thuật toán Newton-Armijo ở đây.

SSVM hoạt động nhanh hơn trên các tập dữ liệu lớn hơn so với SVM thông thường. Nó cũng có thể tạo ra bề mặt chia không gian phi tuyến theo hình ô vuông

e. Facial Expression Classification

Facial Expression Classification using SVM



Happy

Sad

Surprised

Angry

Có nhiều cách để sử dụng phân loại này. Chúng ta có thể sử dụng nó trong các hệ thống chăm sóc sức khỏe, chúng ta cũng có thể sử dụng nó trong việc phân loại biểu hiện vui vẻ hoặc buồn. Chúng ta có thể sử dụng nó trong các bộ lọc. Nếu chúng ta tạo ra các biểu hiện khuôn mặt nhất định, nó sẽ thêm bộ lọc cụ thể tùy thuộc vào biểu hiện đó. Phạm vi của các biểu hiện nằm giữa vui vẻ và buồn. SVM có thể rất hữu ích ở đây

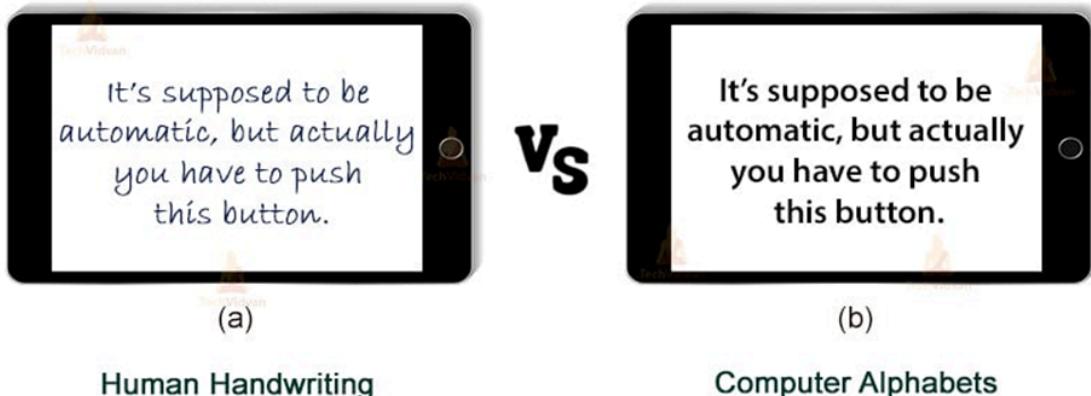
f. Texture Classification using SVM

Trong ứng dụng SVM này, chúng ta sử dụng các hình ảnh của các cấu trúc bề mặt cụ thể và sử dụng dữ liệu đó để phân loại liệu bề mặt có mịn không. Ứng dụng SVM này sẽ rất hữu ích. Nếu chúng ta sử dụng một máy ảnh nhạy cảm để chụp hình và sử dụng dữ liệu đó trong mô hình của chúng ta, chúng ta có thể huấn luyện một mô hình thực sự mạnh mẽ.

Ngoài ra, nếu chúng ta chụp hình các bề mặt, chúng ta có thể xác định bề mặt là mịn hoặc gồ ghề. SVM ở đây phân loại bề mặt là mịn hoặc gồ ghề.

g. Text Classification

Text Classification using SVM



Phân loại văn bản là quá trình tự động phân loại văn bản vào các danh mục đã được xác định trước. Chúng ta sẽ phân loại Email thành thư rác hoặc không phải thư rác, các bài báo tin tức vào các danh mục khác nhau như Chính trị, Thị trường chứng khoán, Thể thao, v.v. Chúng ta có thể sử dụng SVM để phân biệt giữa viết tay của hai người khác nhau.

Tập dữ liệu sẽ chứa hình ảnh của các chữ cái và câu được viết bằng viết tay. Bộ phân loại SVM sẽ được huấn luyện bằng dữ liệu mẫu. Chúng ta cũng có thể sử dụng SVM để phân biệt giữa viết tay của con người và các chữ cái của máy tính.

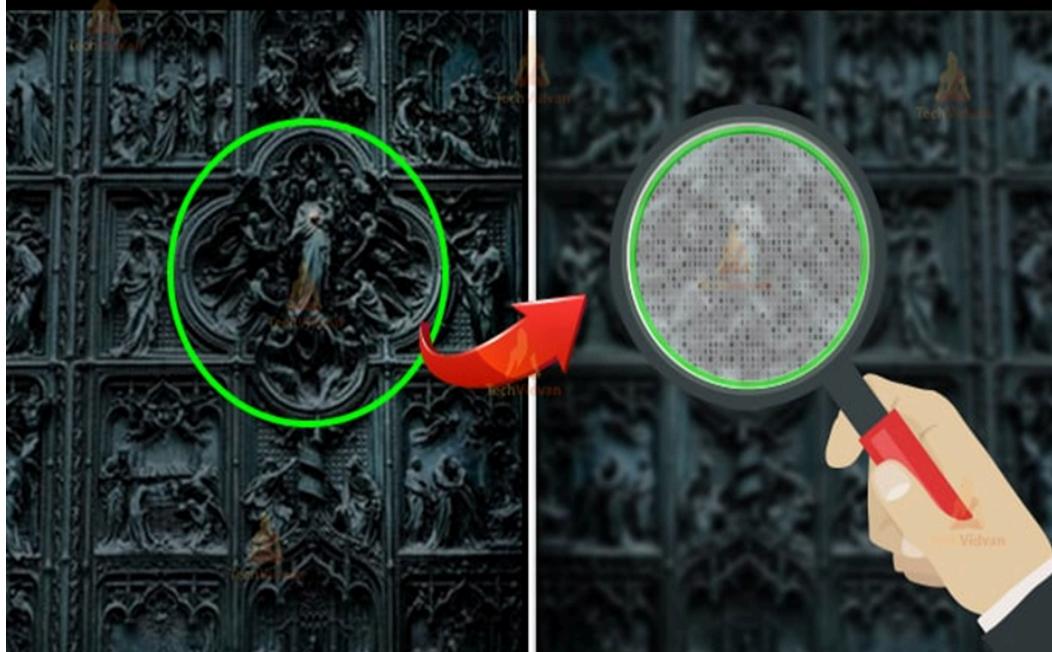
h. Speech Recognition

-Nhận dạng giọng nói được sử dụng để phân loại các từ riêng lẻ từ các bài diễn thuyết liên tục. Các đặc điểm cho mỗi từ cách ly được trích xuất và các mô hình được huấn luyện thành công. Bằng cách sử dụng nhận dạng giọng nói, chúng ta có thể tạo ra một giao diện để giao tiếp với người khiếm thính. Ở đây, âm thanh sẽ là dữ liệu.

-Có nhiều hàm như LPC, LPCC và MFCC thu thập dữ liệu âm thanh. SVM sử dụng dữ liệu âm thanh để huấn luyện các mô hình của nó. Chúng ta sử dụng dữ liệu để huấn luyện nhiều mô hình và sử dụng chúng trong hệ thống. Các kết quả thu được bằng cách sử dụng SVM thường là chính xác.

i. Stenography Detection in Digital Images

Stenography Detection in Digital Images



Sử dụng SVM, chúng ta có thể xác định xem một hình ảnh có là nguyên chất hay bị pha trộn. Điều này có thể được sử dụng trong các tổ chức liên quan đến an ninh để khám phá các tin nhắn bí mật. Đúng, chúng ta có thể mã hóa các tin nhắn trong các hình ảnh có độ phân giải cao.

Trong các hình ảnh có độ phân giải cao, có nhiều pixel hơn, do đó, thông điệp trở nên khó tìm kiếm hơn. Chúng ta có thể phân loại các pixel và lưu trữ dữ liệu trong các tập dữ liệu khác nhau. Chúng ta có thể phân tích các tập dữ liệu đó bằng cách sử dụng SVM.

j. Cancer Diagnosis and Prognosis

Phát hiện ung thư là một trong những lĩnh vực nghiên cứu hàng đầu trên thế giới hiện nay. Có nhiều loại phương pháp Học Máy được sử dụng để tiến hành các nghiên cứu. Google đang sử dụng các phương pháp phân loại hình ảnh để tìm kiếm ung thư.

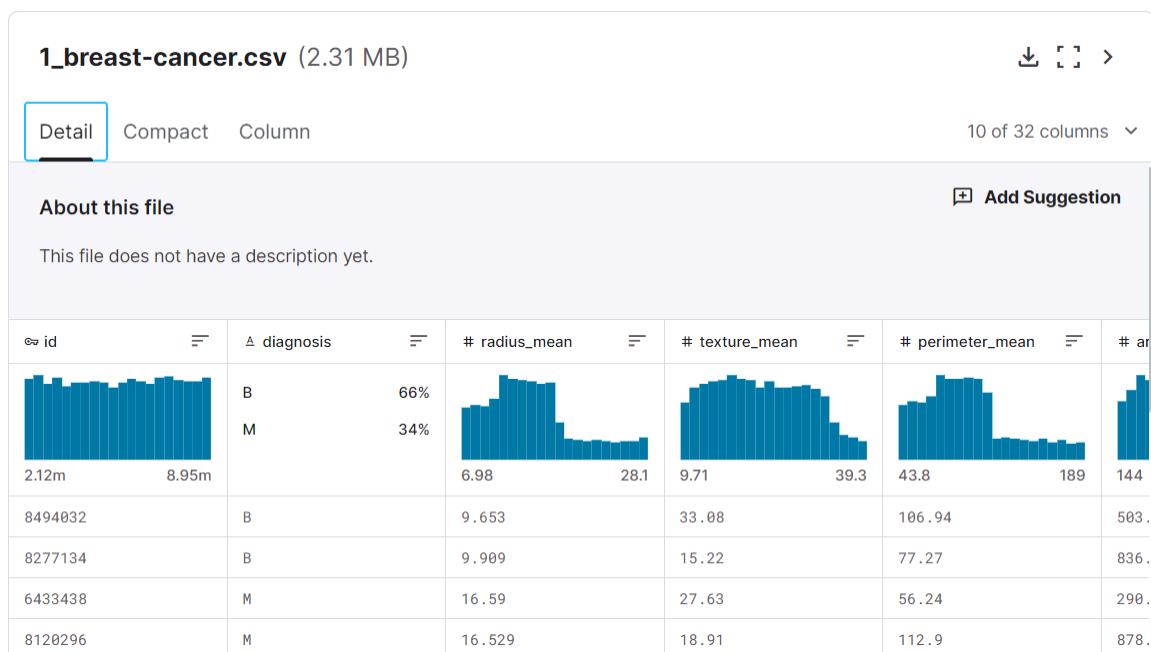
Có nhiều ví dụ như vậy. SVM là một trong những thuật toán được sử dụng cho điều này. SVM giúp trong chẩn đoán và dự đoán bằng cách chạy nhiều mô hình. Có rất nhiều dữ liệu dưới dạng hình ảnh được phân tích. Chúng ta có hàng nghìn bộ dữ liệu.

Chúng ta huấn luyện hàng trăm mô hình sử dụng SVM để phân loại ung thư là ác tính hay lành tính.

V. Bộ dữ liệu và tiền xử lý dữ liệu

1. Mô tả dữ liệu

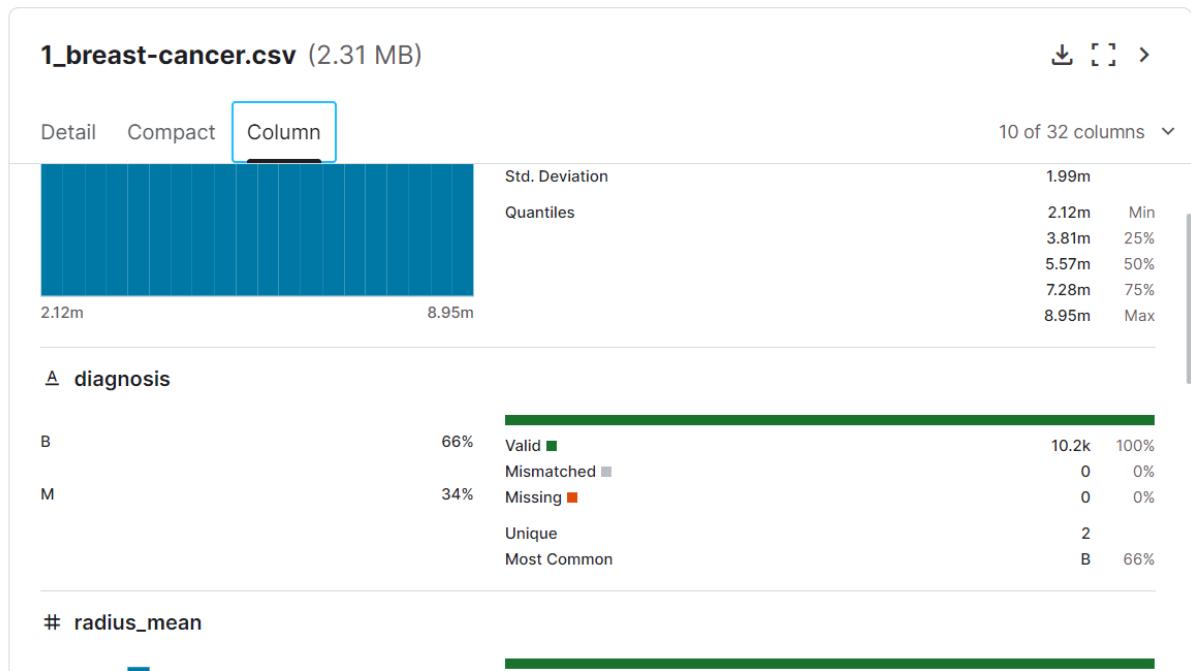
Bộ dữ liệu [breast-cancer.csv](#) chứa các số liệu về việc chẩn đoán ung thư vú.



1_breast-cancer.csv (2.31 MB)

Detail Compact Column 10 of 32 columns ▾

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
8494032	B	9.653	33.08	106.94	503.2	0.14854
8277134	B	9.909	15.22	77.27	836.9	0.16332
6433438	M	16.59	27.63	56.24	290.9	0.05767
8120296	M	16.529	18.91	112.9	878.6	0.07389
8213475	M	15.55	27.7	106.07	387	0.13174
5664018	M	21.15	18.33	111.7	1260.8	0.0972
3353440	M	23.67	28.21	88.4	719.4	0.11282
7407142	M	13.84	16.61	166.5	1392.2	0.10206
8771952	B	16.623	18.98	49.51	310.9	0.15271
8587827	B	24.88	29.97	83.7	2485.7	0.09485
5568435	M	25.94	33.53	147.1	2158.2	0.10754



Ung thư vú là loại ung thư phổi biến nhất ở phụ nữ trên thế giới. Nó chiếm 25% tổng số trường hợp ung thư và ảnh hưởng đến hơn 2,1 triệu người chỉ trong năm 2015. Nó bắt đầu khi các tế bào ở vú bắt đầu phát triển ngoài tầm kiểm soát. Những tế bào này thường hình thành các khối u có thể nhìn thấy qua tia X hoặc cảm nhận được dưới dạng cục u ở vùng vú.

Những thách thức chính đối với việc phát hiện nó là làm thế nào để phân loại khối u thành ác tính (ung thư) hoặc lành tính (không ung thư).

Gồm 10225 hàng và 32 cột gồm

id	mã bản ghi, đây là con số duy nhất
Diagnosis	Đây là chẩn đoán của khối u, trong đó "M" đại diện cho khối u ác tính (ung thư) và "B" đại diện cho khối u lành tính (không ung thư)
Average_radius Bán kính trung bình	Bán kính trung bình của khối u, được tính từ chu vi và diện tích của khối u
Average_Texture Kết cấu trung bình	Một thước đo của sự biến đổi thang độ xám của các pixel trong hình ảnh khối u.
Average_perimeter Chu vi trung bình	Chu vi trung bình của khối u
Average_area Diện tích trung bình	Diện tích trung bình của khối u

Average_Smoothness Độ mượt trung bình	Một thước đo của các biến thể cục bộ trong độ dài của bán kính khối u.
Average_compactness Độ đặc trung bình	Chu vi bình phương của khối u chia cho diện tích khối u trừ đi 1.0
Mean_concavity Độ lõm trung bình	Mức độ nghiêm trọng của các phần lõm trong đường viền khối u
Mean_concave_points Số điểm lõm trung bình	Số lượng các phần lõm trong đường viền khối u
Median_symmetry Độ đối xứng trung bình	Một thước đo của sự đối xứng của khối u
Median_fractal_dimension Chiều đo fractal trung bình	Một thước đo của độ phức tạp của đường viền khối u
Các thuật ngữ kết thúc bằng "_se"	đại diện cho "sai số chuẩn" của các phép đo trước đó
Các thuật ngữ kết thúc bằng "_worst"	đại diện cho giá trị "tệ nhất" hoặc lớn nhất của các phép đo này, tức là giá trị trung bình của ba phép đo lớn nhất.

2. Tiền xử lý dữ liệu

```
df = pd.read_csv('/content/breast-cancer.csv')
# để đọc dữ liệu từ tập tin
```

Xem cấu trúc của data

```
df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10224 entries, 0 to 10223
Data columns (total 33 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   id               10224 non-null   int64  
 1   diagnosis        10224 non-null   object  
 2   radius_mean      10224 non-null   float64 
 3   texture_mean     10224 non-null   float64 
 4   perimeter_mean   10224 non-null   float64 
 5   area_mean         10224 non-null   float64 
 6   smoothness_mean  10224 non-null   float64 
 7   compactness_mean 10224 non-null   float64 
 8   concavity_mean   10224 non-null   float64 
 9   concave points_mean 10224 non-null   float64 
 10  symmetry_mean   10224 non-null   float64 
 11  fractal_dimension_mean 10224 non-null   float64 
 12  radius_se        10224 non-null   float64 
 13  texture_se       10224 non-null   float64 
 14  perimeter_se    10224 non-null   float64 
 15  area_se          10224 non-null   float64 
 16  smoothness_se   10224 non-null   float64 
 17  compactness_se  10224 non-null   float64 
 18  concavity_se    10224 non-null   float64 
 19  concave points_se 10224 non-null   float64 
 ...
 31  fractal_dimension_worst 10224 non-null   float64 
 32  Unnamed: 32       0 non-null      float64 
dtypes: float64(31), int64(1), object(1)
memory usage: 2.6+ MB

```

```

df.head()
# xem trước 1 số hàng đầu tiên của data

```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst
0	8494032	B	9.653	33.08	106.94	503.2	0.14854	0.11601	0.1119	0.04561	...	9.62	30.42
1	8277134	B	9.909	15.22	77.27	836.9	0.16332	0.21563	0.0992	0.07729	...	8.87	40.17
2	6433438	M	16.590	27.63	56.24	290.9	0.05767	0.03576	0.2013	0.02446	...	16.35	13.88
3	8120296	M	16.529	18.91	112.90	878.6	0.07389	0.13996	0.0173	0.05447	...	13.67	24.39
4	8213475	M	15.550	27.70	106.07	387.0	0.13174	0.21961	0.0285	0.00471	...	8.50	34.17

5 rows × 32 columns

Thực hiện việc chuẩn loại bỏ dữ liệu sai sót, dư thừa

```

df.drop('id', axis=1, inplace=True)
# loại bỏ cột id ra khỏi data
# axis = 0: loại bỏ các hàng , axis = 1 loại bỏ các cột
# inplace = true : thực hiện trực tiếp trên data hiện tại

# Loại bỏ các hàng có giá trị NaN
df = df.dropna()

```

```

df.head()
# xem trước 1 số hàng đầu tiên của data
df.info()
# kiểm tra lại cấu trúc

```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst
0	8494032	B	9.653	33.08	106.94	503.2	0.14854	0.11601	0.1119	0.04561	...	9.62	30.42
1	8277134	B	9.909	15.22	77.27	836.9	0.16332	0.21563	0.0992	0.07729	...	8.87	40.17
2	6433438	M	16.590	27.63	56.24	290.9	0.05767	0.03576	0.2013	0.02446	...	16.35	13.88
3	8120296	M	16.529	18.91	112.90	878.6	0.07389	0.13996	0.0173	0.05447	...	13.67	24.39
4	8213475	M	15.550	27.70	106.07	387.0	0.13174	0.21961	0.0285	0.00471	...	8.50	34.17

...

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10224 entries, 0 to 10223
Data columns (total 32 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   diagnosis        10224 non-null   object 
 1   radius_mean      10224 non-null   float64
 2   texture_mean     10224 non-null   float64
 3   perimeter_mean   10224 non-null   float64
 4   area_mean        10224 non-null   float64
 5   smoothness_mean  10224 non-null   float64
 6   compactness_mean 10224 non-null   float64
 7   concavity_mean   10224 non-null   float64
 8   concave points_mean 10224 non-null   float64
 9   symmetry_mean   10224 non-null   float64
 10  fractal_dimension_mean 10224 non-null   float64
 11  radius_se        10224 non-null   float64
 12  texture_se       10224 non-null   float64
 13  perimeter_se    10224 non-null   float64
 14  area_se          10224 non-null   float64
 15  smoothness_se   10224 non-null   float64
 16  compactness_se  10224 non-null   float64
 17  concavity_se   10224 non-null   float64
 18  concave points_se 10224 non-null   float64
 19  symmetry_se    10224 non-null   float64
 ...
 30  fractal_dimension_worst 10224 non-null   float64
 31  Unnamed: 32      0 non-null      float64
dtypes: float64(31), object(1)
memory usage: 2.5+ MB

```

cột id đã bị loại bỏ

3. EDA (trực quan hóa dữ liệu)

```

px.pie(df, 'diagnosis',
color='diagnosis', color_discrete_sequence=['#007500', '#5CFF5C'], title='Data Distribution')

```

```
# tạo ra lược đồ hình tròn từ data
```

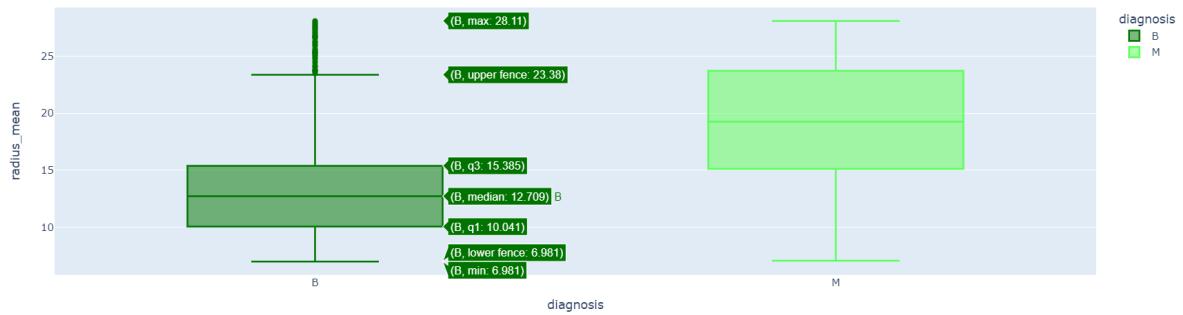
Data Distribution

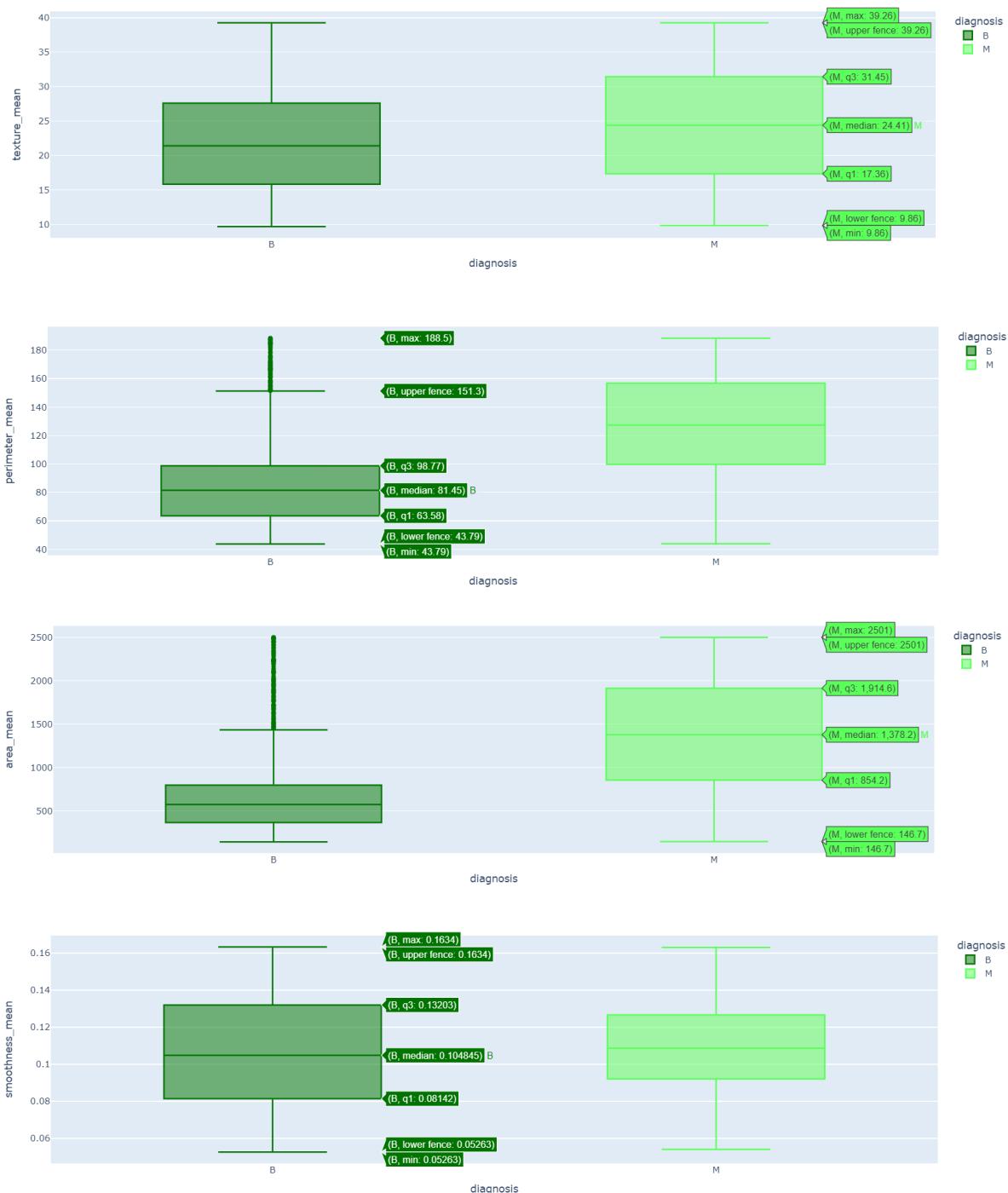


Từ biểu đồ này, chúng ta kết luận rằng:

Dữ liệu không cân bằng, độ chính xác sẽ không phải là một độ đo đánh giá tốt cho tập dữ liệu này.

```
for column in df.drop('diagnosis',axis=1).columns[:5]:  
    fig =  
    px.box(data_frame=df,x='diagnosis',color='diagnosis',y=column,color_discrete_sequence=['#007500','#5CFF5C'],orientation='v')  
    fig.show()  
# tạo ra một loạt các biểu đồ boxplot cho 5 cột đầu tiên
```



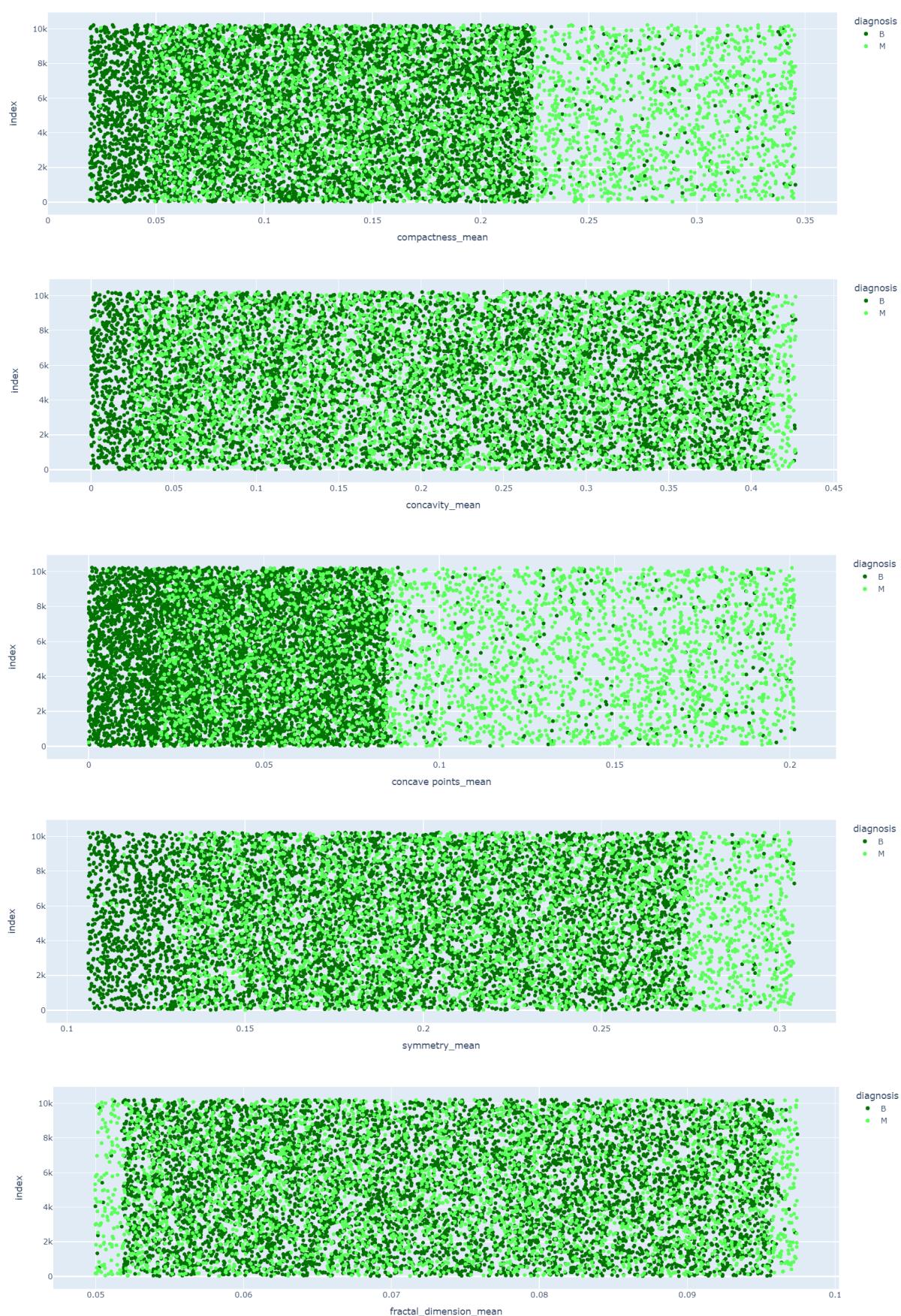


```

for column in df.drop('diagnosis', axis=1).columns[5:10]:
    fig =
    px.scatter(data_frame=df, color='diagnosis', x=column, color_discrete_sequence=[ '#007500', '#5CFF5C'], )
    fig.show()
# tạo ra một loạt các biểu đồ scatter plot cho các biến tính năng trong

```

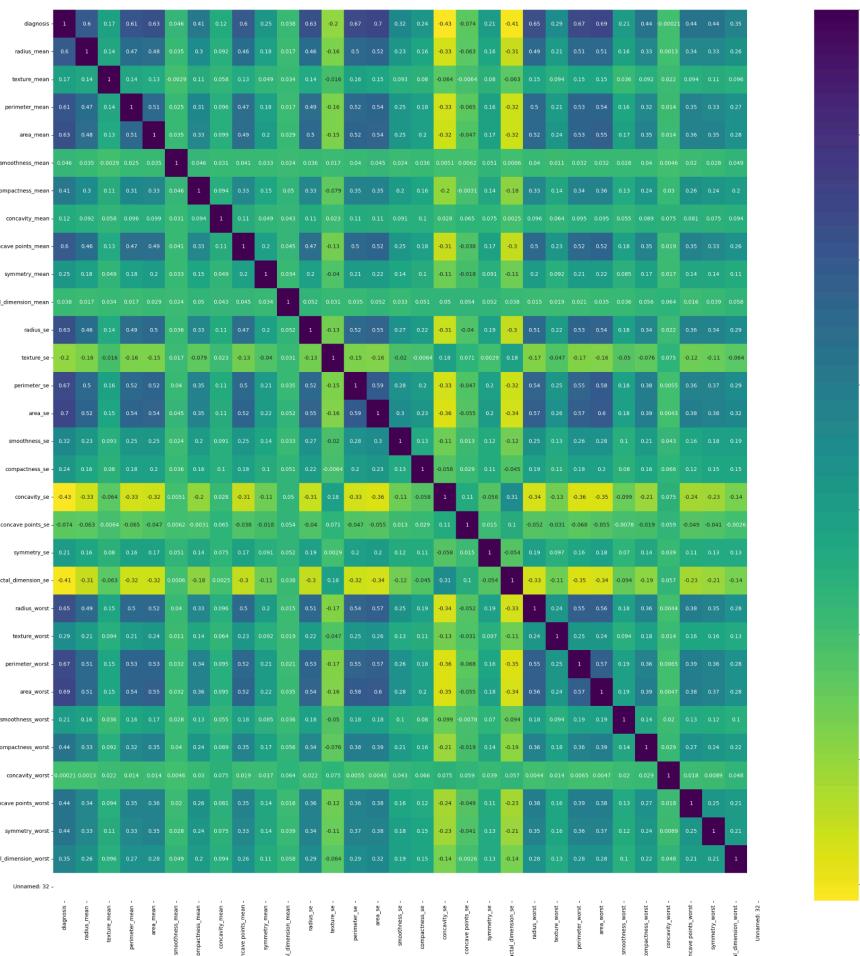
DataFrame df, bắt đầu từ cột thứ 5 đến cột thứ 9



Kiểm tra tương quan

```
df['diagnosis'] = (df['diagnosis'] == 'M').astype(int) #encode the label into 1/0
corr = df.corr()
plt.figure(figsize=(30,30))
sns.heatmap(corr, cmap='viridis_r', annot=True)
plt.show()

# thực hiện việc mã hóa nhãn của cột 'diagnosis' thành các giá trị số nguyên 0 và 1, sau đó tính ma trận tương quan
```



4. Trích rút đặc trưng

```

# Lấy giá trị tuyệt đối của hệ số tương quan
cor_target = abs(corr["diagnosis"])

# Chọn ra các đặc trưng có giá trị tương quan tuyệt đối lớn hơn 0.25
relevant_features = cor_target[cor_target>0.25]

# Tạo danh sách tên các đặc trưng có giá trị tương quan lớn hơn 0.25 bằng cách lặp
# qua các mục (items) trong relevant_features.
names = [index for index, value in relevant_features.items()]

# Loại bỏ diagnosis ra khỏi danh sách tên các đặc trưng vì đây là biến mục tiêu
# chứ không phải đặc trưng
names.remove('diagnosis')

pprint.pprint(names)

# phân tích mối tương quan giữa các đặc trưng (features) trong một DataFrame và
# chọn ra các đặc trưng có mối tương quan cao với biến mục tiêu

```

```

['radius_mean',
'perimeter_mean',
'area_mean',
'compactness_mean',
'concave points_mean',
'radius_se',
'perimeter_se',
'area_se',
'smoothness_se',
'concavity_se',
'fractal_dimension_se',
'radius_worst',
'texture_worst',
'perimeter_worst',
'area_worst',
'compactness_worst',
'concave points_worst',
'symmetry_worst',
'fractal_dimension_worst']

```

- Gán dữ liệu huấn luyện và phân loại nhãn

```

X = df[names].values
y = df['diagnosis'].values.reshape(-1,1)
# Chuẩn bị các đặc trưng (features) để làm biến đầu vào (X):
# Chuẩn bị biến mục tiêu (target variable) để làm biến đầu ra (y):
# reshape(-1, 1): Chuyển đổi mảng 1 chiều thành mảng 2 chiều với một cột và số

```

hàng tương ứng với số mẫu dữ liệu

```
# Đoạn mã này định nghĩa một hàm scale để chuẩn hóa (standardize) dữ liệu trong mảng X.
```

```
def scale(X):
```

```
    """
```

```
    Chuẩn hóa dữ liệu trong mảng X.
```

Tham số:

X (numpy.ndarray): Mảng đặc trưng có kích thước (n_samples, n_features).

Trả về:

numpy.ndarray: Mảng đặc trưng đã được chuẩn hóa.

```
    """
```

```
# Tính giá trị trung bình và độ lệch chuẩn của mỗi đặc trưng
```

```
mean = np.mean(X, axis=0)
```

```
std = np.std(X, axis=0)
```

```
# Chuẩn hóa dữ liệu
```

```
X = (X - mean) / std
```

```
return X
```

```
X = scale(X)
```

```
def train_test_split(X, y, test_size=0.2, random_state=None):
    if random_state:
        np.random.seed(random_state)
    n_samples = X.shape[0]
    n_test = int(test_size * n_samples)
    test_indices = np.random.choice(n_samples, n_test, replace=False)
    train_indices = np.array([i for i in range(n_samples) if i not in
test_indices])
    X_train, X_test = X[train_indices], X[test_indices]
    y_train, y_test = y[train_indices], y[test_indices]
```

```
return X_train, X_test, y_train, y_test
```

VI. Xây dựng mô hình

1. KNN

Lớp KNN:

```
class KNN:  
    def __init__(self, k=3):  
        self.k = k  
  
    def fit(self, X_train, y_train):  
        self.X_train = X_train  
        self.y_train = y_train  
  
    def predict(self, X_test):  
        predictions = []  
        for x in X_test:  
            # Tính khoảng cách Euclid  
            distances = [np.sqrt(np.sum((x - x_train) ** 2)) for x_train in  
self.X_train]  
            # Sắp xếp khoảng cách và lấy chỉ số của k khoảng cách nhỏ nhất.  
            nearest_indices = np.argsort(distances)[:self.k]  
            # Lấy nhãn của k điểm gần nhất  
            nearest_labels = [self.y_train[i] for i in nearest_indices]  
            # Xác định nhãn xuất hiện nhiều nhất trong k nhãn gần nhất.  
            most_common_label = max(set(nearest_labels), key=nearest_labels.count)  
            # Thêm nhãn dự đoán  
            predictions.append(most_common_label)  
        return predictions
```

Huấn luyện mô hình:

```

X = df[names].values
y = df['diagnosis'].values

# Chia tập dữ liệu
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Chuẩn hóa dữ liệu
X_train = scale(X_train)
X_test = scale(X_test)

# Huấn luyện mô hình
k = 5
clf = KNN(k)
clf.fit(X_train, y_train)

# Dự đoán nhãn cho dữ liệu kiểm tra
predictions = clf.predict(X_test)

```

2. Decision tree

- xây dựng class Decision Tree của cây quyết định

```

# Lớp cho cây quyết định
class DecisionTree:
    def __init__(self, min_samples_split=2, max_depth=10):
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.tree = None

    def create_leaf(self, y):
        return np.argmax(np.bincount(y.astype(int)))

    def build_tree(self, dataset, current_depth=0):
        X, y = dataset[:, :-1], dataset[:, -1]
        if len(y) < self.min_samples_split or current_depth >= self.max_depth or
len(np.unique(y)) == 1:
            return self.create_leaf(y)

```

```

best_split = find_best_split(dataset)
if not best_split or best_split["gain"] == 0:
    return self.create_leaf(y)

left_node = self.build_tree(best_split["left_dataset"], current_depth +
1)
right_node = self.build_tree(best_split["right_dataset"], current_depth
+ 1)

return {
    "feature": best_split["feature"],
    "value": best_split["value"],
    "left": left_node,
    "right": right_node
}

def fit(self, X, y):
    dataset = np.concatenate((X, y.reshape(-1, 1)), axis=1)
    self.tree = self.build_tree(dataset)

def predict_one(self, x, tree):
    if not isinstance(tree, dict):
        return tree
    feature = tree["feature"]
    if x[feature] <= tree["value"]:
        return self.predict_one(x, tree["left"])
    else:
        return self.predict_one(x, tree["right"])

def predict(self, X):
    return np.array([self.predict_one(x, self.tree) for x in X])

```

```

def calculate_entropy(y):
    class_labels = np.unique(y)
    entropy = 0
    for label in class_labels:
        p_label = len(y[y == label]) / len(y)
        entropy -= p_label * np.log2(p_label)
    return entropy

```

```

# Hàm tính toán độ lợi thông tin của một lần chia
def calculate_information_gain(left, right, parent_entropy):
    num_left = len(left)
    num_right = len(right)
    num_total = num_left + num_right
    weighted_entropy = (num_left / num_total) *
calculate_entropy(left) + (num_right / num_total) *
calculate_entropy(right)
    return parent_entropy - weighted_entropy

# Hàm chia tập dữ liệu dựa trên một thuộc tính và giá trị
def split_dataset(dataset, feature, value):
    left = dataset[dataset[:, feature] <= value]
    right = dataset[dataset[:, feature] > value]
    return left, right

# Hàm tìm điểm chia tốt nhất
def find_best_split(dataset):
    best_split = {}
    max_gain = -float("inf")
    parent_entropy = calculate_entropy(dataset[:, -1])

    for feature in range(dataset.shape[1] - 1):
        unique_values = np.unique(dataset[:, feature])
        for value in unique_values:
            left, right = split_dataset(dataset, feature, value)
            if len(left) == 0 or len(right) == 0:
                continue
            gain = calculate_information_gain(left[:, -1], right[:, -1], parent_entropy)
            if gain > max_gain:
                max_gain = gain
                best_split = {
                    "feature": feature,
                    "value": value,
                    "left_dataset": left,
                    "right_dataset": right,
                    "gain": gain
                }
    return best_split

```

```
+ def accuracy(y_true, y_pred):
```

```
def accuracy(y_true, y_pred):
```

```
    """
```

```
    Tham số:
```

- y_true (numpy array): Một mảng numpy của các nhãn thực cho mỗi điểm dữ liệu.
- y_pred (numpy array): Một mảng numpy của các nhãn dự đoán cho mỗi điểm dữ liệu.

```
Trả về:
```

- float: Độ chính xác của mô hình

```
    """
```

```
    y_true = y_true.flatten()
    total_samples = len(y_true)
    correct_predictions = np.sum(y_true == y_pred)
    return (correct_predictions / total_samples)
```

```
+ def balanced_accuracy(y_true, y_pred):
```

```
def balanced_accuracy(y_true, y_pred):
```

```
    """Tham số:
```

- y_true (numpy array): Một mảng numpy của các nhãn thực cho mỗi điểm dữ liệu.

- y_pred (numpy array): Một mảng numpy của các nhãn dự đoán cho mỗi điểm dữ liệu.

```
    """
```

```
    y_pred = np.array(y_pred)
    y_true = y_true.flatten()
    # Lấy số lớp.
    n_classes = len(np.unique(y_true))
```

```
    # Khởi tạo một mảng để lưu trữ độ nhạy và độ đặc hiệu cho mỗi lớp.
```

```
    sen = []
    spec = []
```

```
    # Lặp qua mỗi lớp.
```

```
    for i in range(n_classes):
```

```
        # Tạo một mặt nạ cho các giá trị thực và dự đoán cho lớp i.
```

```
        mask_true = y_true == i
```

```
        mask_pred = y_pred == i
```

```

# Tính true positive, true negative, false positive, và false negative
values
TP = np.sum(mask_true & mask_pred)
TN = np.sum((mask_true != True) & (mask_pred != True))
FP = np.sum((mask_true != True) & mask_pred)
FN = np.sum(mask_true & (mask_pred != True))

# Tính the sensitivity (true positive rate) và specificity (true
negative rate)
sensitivity = TP / (TP + FN) if (TP + FN) > 0 else 0
specificity = TN / (TN + FP) if (TN + FP) > 0 else 0

# Lưu the sensitivity và specificity cho class i
sen.append(sensitivity)
spec.append(specificity)

# Tính toán ma trận nhầm lẫn
confusion_matrix = np.zeros((n_classes, n_classes), dtype=int)
for i in range(n_classes):
    for j in range(n_classes):
        confusion_matrix[i, j] = np.sum((y_true == i) & (y_pred == j))

# Tính precision, recall và F1-score cho mỗi lớp
precision = []
recall = []
f1_score = []
for i in range(n_classes):
    TP = confusion_matrix[i, i]
    FP = np.sum(confusion_matrix[:, i]) - TP
    FN = np.sum(confusion_matrix[i, :]) - TP
    precision.append(TP / (TP + FP) if (TP + FP) > 0 else 0)
    recall.append(TP / (TP + FN) if (TP + FN) > 0 else 0)
    f1_score.append(2 * (precision[i] * recall[i]) / (precision[i] +
recall[i]) if (precision[i] + recall[i]) > 0 else 0)

return confusion_matrix, precision, recall, f1_score

```

```

#Tạo một phiên bản mô hình
model = DecisionTree(100, 16)

# Điều chỉnh mô hình cây quyết định với dữ liệu huấn luyện
model.fit(X_train, y_train)

# "Sử dụng mô hình đã huấn luyện để dự đoán trên dữ liệu kiểm thử."
predictions = model.predict(X_test)

```

3. SVM

Hàm chia dữ liệu thành tập huấn luyện và tập kiểm tra

```

def train_test_split(X, y, random_state=42, test_size=0.2):
    """
    Chia dữ liệu thành tập huấn luyện và tập kiểm tra.

    #### Tham số:
    - **X (numpy.ndarray)**: Mảng đặc trưng với hình dạng (n_samples,
    n_features).
    - **y (numpy.ndarray)**: Mảng mục tiêu với hình dạng (n_samples).
    - **random_state (int)**: Hạt giống cho bộ tạo số ngẫu nhiên. Mặc định là 42.
    - **test_size (float)**: Tỷ lệ mẫu sẽ được bao gồm trong tập kiểm tra.
    Mặc định là 0.2.

    #### Trả về:
    - **Tuple[numpy.ndarray]**: Một bộ chứa X_train, X_test, y_train, y_test.

    """
    # lấy ra số lượng mẫu
    n_samples = X.shape[0]

    # Đặt seed cho sinh số ngẫu nhiên
    np.random.seed(random_state)

    # Tạo một hoán vị ngẫu nhiên của các chỉ mục từ 0 đến (n_samples - 1):
    shuffled_indices = np.random.permutation(np.arange(n_samples))

    # tính toán số lượng mẫu cho tập ktra
    test_size = int(n_samples * test_size)

```

```

# tạo ra tập chỉ mục cho tập kiểm tra
test_indices = shuffled_indices[:test_size]
train_indices = shuffled_indices[test_size:]

# Chia các mảng đặc trưng và mục tiêu thành tập kiểm tra và tập huấn
luyện
X_train, X_test = X[train_indices], X[test_indices]
y_train, y_test = y[train_indices], y[test_indices]

return X_train, X_test, y_train, y_test

```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state=42)
#Chia dữ liệu thành tập huấn luyện và tập validation

```

Lớp SVM:

```

class SVM:
    """
    A Support Vector Machine (SVM) implementation using gradient descent.

    Tham số:
    -----
    iterations : int, mặc định=1000
    Số lần lặp cho gradient descent.
    lr : float, mặc định=0.01
    Tốc độ học cho gradient descent.
    lambdaa : float, mặc định=0.01
    Tham số điều chỉnh.

    Thuộc tính:
    -----
    lambdaa : float
        Tham số điều chỉnh.
    iterations : int
        Số lần lặp cho gradient descent.

```

```

lr : float
    Tốc độ học cho gradient descent.

w : numpy array
    Các trọng số.

b : float
    Sai số.

Phương thức:
-----
initialize_parameters(X) : Khởi tạo các trọng số và sai số.
gradient_descent(X, y) : Cập nhật các trọng số và sai số sử dụng gradient
descent.
update_parameters(dw, db) : Cập nhật các trọng số và sai số.
fit(X, y) :Điều chỉnh SVM với dữ liệu.
predict(X) :Dự đoán nhãn cho dữ liệu được cho
"""

#Hàm thiết lập các tham số
def __init__(self, iterations=1000, lr=0.01, lambdaaa=0.01):
    """
    Khởi tạo mô hình SVM.

    """
    self.lambdaaa = lambdaaa
    self.iterations = iterations
    self.lr = lr
    self.w = None
    self.b = None

#Hàm khởi tạo trọng số và sai số ban đầu
def initialize_parameters(self, X):
    m, n = X.shape
    self.w = np.zeros(n)
    self.b = 0

    def gradient_descent(self, X, y):

```

```

"""
Cập nhật các trọng số và bias sử dụng gradient descent.

### Tham số:
- **X (numpy array)**: Dữ liệu đầu vào.
- **y (numpy array)**: Các giá trị mục tiêu.

"""

y_ = np.where(y <= 0, -1, 1)
for i, x in enumerate(X):
    if y_[i] * (np.dot(x, self.w) - self.b) >= 1:
        dw = 2 * self.lambaa * self.w
        db = 0
    else:
        dw = 2 * self.lambaa * self.w - np.dot(x, y_[i])
        db = y_[i]
    self.update_parameters(dw, db)

def update_parameters(self, dw, db):
    """
    Cập nhật các trọng số và bias.

    ### Tham số:
    - **dw (numpy array)**: Sự thay đổi của các trọng số.
    - **db (float)**: Sự thay đổi của bias.

    """

    self.w = self.w - self.lr * dw
    self.b = self.b - self.lr * db

def fit(self, X, y):
    """
    Điều chỉnh SVM cho dữ liệu.

    ### Tham số:
    - **X (numpy array)**: Dữ liệu đầu vào.
    - **y (numpy array)**: Các giá trị mục tiêu.

    """

    self.initialize_parameters(X)
    for i in range(self.iterations):
        self.gradient_descent(X, y)

def predict(self, X):
    """
    """

```

Dự đoán nhãn lớp cho dữ liệu kiểm tra.

```
### Tham số:  
- **X (array-like, shape (n_samples, n_features))**: Dữ liệu đầu vào.  
  
### Trả về:  
- **y_pred (array-like, shape (n_samples,))**: Các nhãn lớp được dự đoán.  
    """  
    # lấy các giá trị đầu ra  
    output = np.dot(X, self.w) - self.b  
    # lấy dấu của các nhãn tùy thuộc vào nó lớn hơn/ít hơn 0  
    label_signs = np.sign(output)  
    # đặt các dự đoán thành 0 nếu chúng nhỏ hơn hoặc bằng -1 nếu không  
    # đặt chúng thành 1  
    predictions = np.where(label_signs <= -1, 0, 1)  
    return predictions
```

Hàm tính toán độ chính xác của một mô hình phân loại

```
def accuracy(y_true, y_pred):  
    total_samples = len(y_true)  
    correct_predictions = np.sum(y_true == y_pred)  
    return (correct_predictions / total_samples)  
  
def confusion_matrix(y_true, y_pred):  
    unique_classes = np.unique(np.concatenate((y_true, y_pred)))  
    num_classes = len(unique_classes)  
    cm = np.zeros((num_classes, num_classes), dtype=int)  
    for true_label, pred_label in zip(y_true, y_pred):  
        cm[true_label, pred_label] += 1  
    return cm  
  
def precision(y_true, y_pred, class_label, unique_classes):  
    """  
    Computes the precision.  
  
    Parameters:  
    -----  
    y_true (numpy array): A numpy array of true labels for each data  
    point.  
    y_pred (numpy array): A numpy array of predicted labels for each data
```

point.

```
    class_label: The label for which precision is to be calculated.  
    unique_classes (numpy array): A numpy array of unique class labels.
```

Returns:

```
    float: The precision score for the specified class label.
```

```
"""
```

```
cm = confusion_matrix(y_true, y_pred)  
class_index = np.where(unique_classes == class_label)[0][0]  
tp = cm[class_index, class_index]  
fp = np.sum(cm[:, class_index]) - tp  
precision = tp / (tp + fp) if (tp + fp) > 0 else 0  
return precision
```

```
def recall(y_true, y_pred, class_label, unique_classes):
```

```
"""
```

Computes the recall.

Parameters:

```
    y_true (numpy array): A numpy array of true labels for each data  
    point.
```

```
    y_pred (numpy array): A numpy array of predicted labels for each data  
    point.
```

```
    class_label: The label for which recall is to be calculated.
```

```
    unique_classes (numpy array): A numpy array of unique class labels.
```

Returns:

```
    float: The recall score for the specified class label.
```

```
"""
```

```
cm = confusion_matrix(y_true, y_pred)  
class_index = np.where(unique_classes == class_label)[0][0]  
tp = cm[class_index, class_index]  
fn = np.sum(cm[class_index, :]) - tp  
recall = tp / (tp + fn) if (tp + fn) > 0 else 0  
return recall
```

```
def f1_score(y_true, y_pred, class_label, unique_classes):
```

```
"""
```

Computes the F1-score.

```

Parameters:
-----
y_true (numpy array): A numpy array of true labels for each data
point.
y_pred (numpy array): A numpy array of predicted labels for each data
point.
class_label: The label for which F1-score is to be calculated.
unique_classes (numpy array): A numpy array of unique class labels.

Returns:
-----
float: The F1-score for the specified class label.

"""
precision_val = precision(y_true, y_pred, class_label, unique_classes)
recall_val = recall(y_true, y_pred, class_label, unique_classes)
f1 = 2 * (precision_val * recall_val) / (precision_val + recall_val) if
(precision_val + recall_val) > 0 else 0
return f1

```

```

model = SVM()
model.fit(X_train,y_train)
predictions = model.predict(X_test)

```

model = SVM() :Tạo một đối tượng SVM từ triển khai đã được cung cấp. Đối tượng này sẽ đại diện cho mô hình SVM

model.fit(X_train,y_train):Huấn luyện mô hình SVM trên tập dữ liệu huấn luyện X_train với nhãn tương ứng y_train Trong quá trình này, mô hình sẽ học các tham số như trọng số và sai số để phân loại dữ liệu.

predictions = model.predict(X_test):Dùng mô hình đã huấn luyện để dự đoán nhãn cho tập dữ liệu kiểm tra X_test , phương thức predict sẽ trả về một mảng chứa các nhãn dự đoán cho mỗi mẫu trong X_test.

VII. Chứng minh tính đúng đắn của thuật toán đã xây dựng

1. SVM

Mô hình SVM được khởi tạo và huấn luyện từ sklearn.

- + Ta so sánh kết quả dự đoán giữa 2 mô hình SVM được xây dựng và mô hình SVM từ thư viện sklearn.

```
#So sánh dự đoán giữa các mô hình
matches = np.sum(predictions == sk_predictions)

#Tính phần trăm dự đoán giống nhau
total_samples = len(predictions)

percentage_matches = (matches / total_samples) * 100
print(f"Percentage of matching predictions: {percentage_matches:.2f}%")

Percentage of matching predictions: 99.71%
```

2. KNN

Mô hình KNN được khởi tạo và huấn luyện từ sklearn.

- + Ta so sánh kết quả dự đoán giữa 2 mô hình KNN được xây dựng và mô hình KNN từ thư viện sklearn.

```
1 #So sánh dự đoán giữa các mô hình
2 matches = np.sum(predictions == sk_predictions)
3
4 #Tính phần trăm dự đoán giống nhau
5 total_samples = len(predictions)
6
7 percentage_matches = (matches / total_samples) * 100
8 print(f"Percentage of matching predictions: {percentage_matches:.2f}%")

Percentage of matching predictions: 100.00%
```

3. Decision tree

Mô hình Decision tree được khởi tạo và huấn luyện từ sklearn.

- + Ta so sánh kết quả dự đoán giữa 2 mô hình Decision tree được xây dựng và mô hình Decision tree từ thư viện sklearn.

```

1 #So sánh dự đoán giữa các mô hình
2 matches = np.sum(predictions == sk_predictions)
3
4 #Tính phần trăm dự đoán giống nhau
5 total_samples = len(predictions)
6
7 percentage_matches = (matches / total_samples) * 100
8 print(f"Percentage of matching predictions: {percentage_matches:.2f}%")

```

Percentage of matching predictions: 93.40%

VIII. Kiểm nghiệm và đánh giá mô hình đã xây dựng

Confusion Matrix

Confusion matrix thể hiện có bao nhiêu điểm dữ liệu thực sự thuộc vào một nhãn, và được dự đoán là rơi vào một nhãn

		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

Trong bài toán dự đoán bệnh ung thư vú ta có 2 lớp: lớp ác tính Positive và lớp lành tính là Negative:

- TP (True Positive): Số lượng dự đoán chính xác. Là khi mô hình dự đoán đúng 1 u là ác tính
- TN (True Negative): Số lượng dự đoán chính xác một cách gián tiếp. Là khi mô hình dự đoán đúng 1 u lành tính

- FP (False Positive): Số lượng các dự đoán sai lầm. Là khi mô hình dự đoán 1 là ác tính nhưng nó là lành tính
- FN (False Negative): Số lượng các dự đoán sai lầm một cách gián tiếp. Là khi mô hình dự đoán 1 là lành tính nhưng nó là ác tính

Accuracy (Độ chính xác) : Độ chính xác là tỉ lệ giữa số bản ghi được dự đoán đúng và tổng số bản ghi trong tập dữ liệu kiểm thử. Độ chính xác giúp ta đánh giá hiệu quả dự báo của mô hình trên một bộ dữ liệu. Độ chính xác càng cao thì mô hình của chúng ta càng chuẩn xác

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{số lượng mẫu}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Precision (Độ chuẩn xác)

Precision được định nghĩa là tỉ lệ số điểm Positive mô hình dự đoán đúng trên tổng số điểm mô hình dự đoán là Positive.

Precision càng cao, tức là số điểm mô hình dự đoán là positive đều là positive càng nhiều. Precision = 1, tức là tất cả số điểm mô hình dự đoán là Positive đều đúng, hay không có điểm nào có nhãn là Negative mà mô hình dự đoán nhầm là Positive.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall (Độ phủ):

Recall đo lường tỷ lệ dự báo chính xác các trường hợp positive trên toàn bộ các mẫu thuộc nhóm positive.

Recall càng cao, tức là số điểm là positive bị bỏ sót càng ít. Recall = 1, tức là tất cả số điểm có nhãn là Positive đều được mô hình nhận ra

$$Recall = \frac{TP}{TP + FN}$$

F1-score:

Tuy nhiên, chỉ có Precision hay chỉ có Recall thì không đánh giá được chất lượng mô hình.

- Chỉ dùng Precision, mô hình chỉ đưa ra dự đoán cho một điểm mà nó chắc chắn nhất. Khi đó Precision = 1, tuy nhiên ta không thể nói là mô hình này tốt.
- Chỉ dùng Recall, nếu mô hình dự đoán tất cả các điểm đều là positive. Khi đó Recall = 1, tuy nhiên ta cũng không thể nói đây là mô hình tốt.

F1-score là trung bình điều hòa (harmonic mean) của precision và recall (giả sử hai đại lượng này khác 0). F1-score được tính theo công thức: F1 càng cao thì càng tốt. Khi lý tưởng nhất thì F1 = 1 (khi Recall = Precision=1).

$$F1\ Score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

So sánh 3 thuật toán SVM, KNN và Decision tree với các chỉ số trên 2 lớp '0' và '1' của tập Predictions:

SVM:

```
Unique labels in y_test: [0 1]
Unique labels in predictions: [0 1]
Custom Accuracy: 0.9461839530332681
Confusion Matrix:
[[1266   68]
 [ 42  668]]
Precision: [0.9678899082568807, 0.907608695652174]
Recall: [0.9490254872563718, 0.907608695652174]
F1-score: [0.9583648750946253, 0.907608695652174]
```

KNN:

```
K = 5
Unique labels in y_test: [0 1]
Unique labels in predictions: [0 1]
Custom Accuracy: 0.9408023483365949
Accuracy: 0.9408023483365949
Confusion Matrix:
[[1265   69]
 [ 52  658]]
Precision: [0.9605163249810175, 0.9050894085281981]
Recall [0.9482758620689655, 0.9050894085281981]
F1-score: [0.954356846473029, 0.9050894085281981]
```

Decision tree:

```
Unique labels in y_test: [0 1]
Unique labels in predictions: [0 1]
Custom Accuracy: 0.934931506849315
Balanced Accuracy: 0.9352071185617103
Confusion Matrix:
[[1282   90]
 [ 43  629]]
Precision: [0.9675471698113207, 0.874826147426982]
Recall: [0.934402332361516, 0.9360119047619048]
F1-score: [0.9506859473489062, 0.9043853342918764]
```

Đánh giá:

Đặc điểm	SVM	KNN	Decision Tree
Nguyên lý cơ bản	Tìm siêu phẳng phân cách tốt nhất	Dựa trên khoảng cách tới các điểm gần nhất	Xây dựng cây phân nhánh dựa trên các thuộc tính
Cấu trúc mô hình	Siêu phẳng (Hyperplane)	K điểm gần nhất	Cây với các nút và nhánh
Thời gian huấn luyện	Trung bình đến cao (phụ thuộc vào kernel)	Thấp	Thấp đến trung bình
Thời gian dự đoán	Thấp	Cao	Thấp đến trung bình
Khả năng mở rộng	Khó mở rộng với dữ liệu lớn	Khó mở rộng với dữ liệu lớn	Tốt với dữ liệu lớn
Hiệu suất với dữ liệu nhiều chiều	Tốt	Kém	Trung bình
Xử lý dữ liệu nhiễu	Tốt (với margin lớn)	Kém	Kém (có thể quá khớp nếu không cắt tỉa)
Điều giải kết quả	Khó	Dễ	Dễ
Khả năng đối phó với thiếu dữ liệu	Trung bình	Kém	Trung bình
Ứng dụng phổ biến	Phân loại văn bản, nhận dạng hình ảnh	Hệ thống khuyến nghị, phát hiện bất thường	Quyết định y khoa, khai thác dữ liệu

IX. Tài liệu tham khảo

[6 kỹ thuật khai phá dữ liệu phổ biến \(Data mining\)](#)

[Data mining là gì? Khám phá ứng dụng của data mining trên nhiều lĩnh vực khác nhau](#)

[FARES ELMENSHAWII](#)

[MARIO CAESAR](#)

