

ĐẠI HỌC HUẾ

# KHOA KỸ THUẬT VÀ CÔNG NGHỆ

🙠🙟🕮🙝🙢

****

**BÁO CÁO ĐỒ ÁN**

**(hoặc TIỂU LUẬN, BÀI TẬP LỚN)**

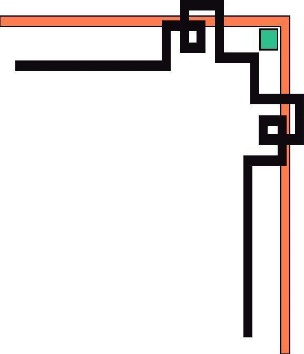
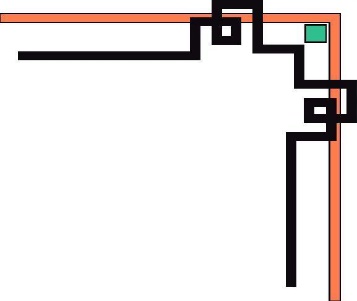
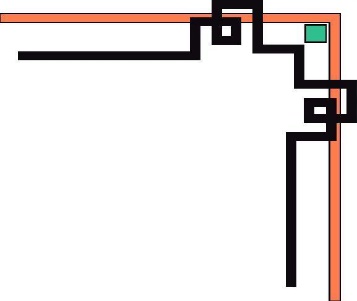
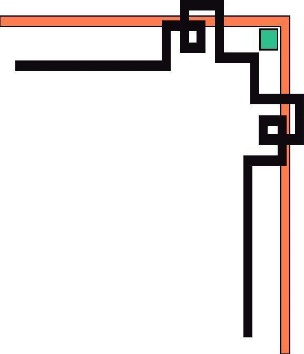
**Học kỳ II, năm học 2022 - 2023**

**Học phần:**

**Các thuật toán tối ưu cho phân tích dữ liệu**

|  |
| --- |
| **Số phách**  *(Do hội đồng chấm thi ghi)* |

**Thừa Thiên Huế, 18 tháng 5 năm 2023**



ĐẠI HỌC HUẾ

# KHOA KỸ THUẬT VÀ CÔNG NGHỆ

🙠🙟🕮🙝🙢

****

**(Bìa phụ 2)**

**BÁO CÁO ĐỒ ÁN**

**(hoặc TIỂU LUẬN, BÀI TẬP LỚN)**

**Học kỳ II, năm học 2022 - 2023**

**Học phần:**

**Các thuật toán tối ưu cho phân tích dữ liệu**

**Giảng viên hướng dẫn: Hoàng Trọng Lợi**

**Lớp: Khoa học dữ liệu & Trí tuệ nhân tạo**

**Sinh viên thực hiện: Nguyễn Trịnh Tấn Đạt**

*(ký tên và ghi rõ họ tên)*

|  |
| --- |
| **Số phách**  *(Do hội đồng chấm thi ghi)* |

**Thừa Thiên Huế, 18 tháng 5 năm 2023**

**MỤC LỤC**

[CHƯƠNG I: GIỚI THIỆU SUPPORT VECTOR MACHINE, PROXIMAL GRADIENT .............................................................................4](#_Toc135351438)

[1. Bài toán tối ưu và các phương pháp giải quyết 4](#_Toc135351439)

[1.1. Khái niệm bài toán tối ưu 4](#_Toc135351440)

[1.2. Các phương pháp giải quyết bài toán tối ưu 4](#_Toc135351441)

[2. SVM (Support Vector Machine) 5](#_Toc135351442)

[3. Thuật toán Proximal Gradient và lý do sử dụng 6](#_Toc135351443)

[3.1. Khái niệm Proximal Gradient 6](#_Toc135351444)

[3.2. Lý do sử dụng proximal gradient 7](#_Toc135351445)

[4. Cách thức hoạt động 7](#_Toc135351446)

[5. Ưu và nhược điểm proximal gradient 8](#_Toc135351447)

[5.1. Ưu điểm của phương pháp Proximal Gradient. 8](#_Toc135351448)

[5.2. Nhược điểm của phương pháp Proximal Gradient. 10](#_Toc135351449)

[6. Ứng dụng proximal gradient trong thực tế 11](#_Toc135351450)

[CHƯƠNG II: CÀI ĐẶT THUẬT TOÁN ...................................................13](#_Toc135351451)

[1. Chọn ngôn ngữ lập trình và môi trường phù hợp 13](#_Toc135351452)

[2. Cài đặt thuật toán Proximal Gradient 13](#_Toc135351453)

[3. Sử dụng thư viện hỗ trợ 14](#_Toc135351454)

[CHƯƠNG III: LỰA CHỌN DATASET VÀ KIỂM THỬ....................... 15](#_Toc135351455)

[1. Mô tả dữ liệu 15](#_Toc135351456)

[2. Mục đích kiểm thử 15](#_Toc135351457)

[3. Kiểm thử 16](#_Toc135351458)

[4. Kết quả 17](#_Toc135351459)

[CHƯƠNG IV: KẾT LUẬN .........................................................................18](#_Toc135351460)

1. GIỚI THIỆU SUPPORT VECTOR MACHINE, PROXIMAL GRADIENT
   1. Bài toán tối ưu và các phương pháp giải quyết
      1. Khái niệm bài toán tối ưu

Bài toán tối ưu là một lĩnh vực quan trọng trong khoa học máy tính và toán học, nơi chúng ta cố gắng tìm ra giá trị tối ưu của một hàm mục tiêu dưới sự ảnh hưởng của các ràng buộc. Đây là một bài toán có nhiều ứng dụng thực tế, từ tối ưu hóa các hàm mất mát trong học máy, xử lý tín hiệu, đến tối ưu hoá tổ chức và quản lý.

Bài toán tối ưu thường được biểu diễn dưới dạng tìm giá trị của biến x mà làm giảm đến mức tối thiểu hàm mục tiêu f(x), trong khi tuân thủ các ràng buộc liên quan. Điều quan trọng là tìm ra một giải pháp tối ưu trong một thời gian hợp lý và với độ chính xác đủ cao.

* + 1. Các phương pháp giải quyết bài toán tối ưu

Trong thực tế, bài toán tối ưu có thể rất phức tạp và không thể giải trực tiếp. Do đó, đã có nhiều phương pháp và thuật toán được phát triển để giải quyết các bài toán tối ưu này. Dưới đây là một số phương pháp phổ biến:

Gradient Descent: Đây là một thuật toán tối ưu đơn giản và phổ biến nhất. Nó sử dụng thông tin đạo hàm của hàm mục tiêu để điều chỉnh các tham số và tìm điểm tối ưu. Gradient Descent thường được áp dụng cho các bài toán tối ưu trơn và khả vi.

Stochastic Gradient Descent (SGD): Là phiên bản tiến hóa của Gradient Descent, SGD sử dụng một mẫu ngẫu nhiên để tính toán đạo hàm, giúp giảm thiểu thời gian tính toán và dễ dàng áp dụng cho dữ liệu lớn.

Newton's Method: Đây là một phương pháp tối ưu sử dụng đạo hàm bậc hai của hàm mục tiêu để xác định hướng di chuyển. Newton's Method thường hội tụ nhanh hơn Gradient Descent nhưng yêu cầu tính toán đạo hàm bậc hai.

Quasi-Newton Methods: Đây là một nhóm các phương pháp sử dụng xấp xỉ ma trận đạo hàm bậc hai để xác định hướng di chuyển và tốc độ học tương tự như Newton's Method. Tuy nhiên, thay vì tính toán đạo hàm bậc hai chính xác, các phương pháp này xây dựng một xấp xỉ từ các đạo hàm bậc nhất để giảm độ phức tạp tính toán.

Coordinate Descent: Phương pháp này tối ưu hóa từng biến một trong hàm mục tiêu, giữ các biến khác cố định. Nó thường hiệu quả đối với các bài toán có số chiều lớn và các biến không tương quan mạnh.

L-BFGS: Đây là một thuật toán tối ưu sử dụng kỹ thuật xấp xỉ ma trận đạo hàm bậc hai. Nó kết hợp sự linh hoạt của Quasi-Newton Methods với việc giảm bộ nhớ và tính toán.

Genetic Algorithms: Đây là một phương pháp tối ưu dựa trên cơ chế tiến hóa trong tự nhiên. Thuật toán tạo ra các cá thể (giải pháp) ngẫu nhiên và áp dụng các phép lai ghép và đột biến để tạo ra các thế hệ tiếp theo với chất lượng tốt hơn.

Simulated Annealing: Phương pháp này mô phỏng quá trình gia nhiệt và làm mát trong luyện kim để tìm kiếm giải pháp tối ưu. Nó cho phép nhảy ra khỏi các điểm cực tiểu cục bộ và khám phá không gian tìm kiếm rộng hơn.

Particle Swarm Optimization: Đây là một phương pháp tối ưu dựa trên mô phỏng quá trình di cư của bầy đàn động vật. Các "hạt" (particle) di chuyển trong không gian tìm kiếm và tương tác với nhau để tìm ra giải pháp tối ưu.

* 1. SVM (Support Vector Machine)

Support Vector Machine (SVM) là một thuật toán học máy được sử dụng rộng rãi trong bài toán phân loại và hồi quy. SVM được phát triển bởi Vapnik và đồng nghiệp vào những năm 1990 và đã trở thành một trong những thuật toán phân loại mạnh mẽ và phổ biến nhất hiện nay. SVM có khả năng xử lý cả các bài toán phân loại tuyến tính và phi tuyến tính, và có độ hiệu quả cao trên các tập dữ liệu lớn và phức tạp.

Ý tưởng cơ bản của SVM là tìm một siêu phẳng (hyperplane) tối ưu trong không gian đa chiều để phân tách các điểm dữ liệu thuộc vào các nhóm khác nhau. Hyperplane là một đường thẳng (trong không gian hai chiều), một mặt phẳng (trong không gian ba chiều) hoặc một siêu phẳng (trong không gian cao chiều hơn) mà chia tách không gian thành hai phần, mỗi phần chứa các điểm dữ liệu của một nhóm.

Với bài toán phân loại tuyến tính, SVM tìm siêu phẳng sao cho khoảng cách từ các điểm dữ liệu gần nhất của các nhóm đến siêu phẳng là lớn nhất. Các điểm dữ liệu gần nhất của các nhóm được gọi là support vectors, và chúng là những điểm dữ liệu quan trọng nhất trong quá trình xác định siêu phẳng phân loại.

Để xây dựng siêu phẳng tối ưu, SVM tối thiểu hóa một hàm mục tiêu (objective function) kết hợp giữa độ lớn của vectơ trọng số và lỗi phân loại. Hàm mục tiêu của SVM có dạng:

Trong đó:

* w là vectơ trọng số của siêu phẳng,
* b là hệ số điều chỉnh đường phân chia,
* yi là nhãn lớp của điểm dữ liệu xi,
* C là tham số đánh đổi giữa regularization (độ lớn của vectơ trọng số) và lỗi phân loại.
  1. Thuật toán Proximal Gradient và lý do sử dụng
     1. Khái niệm Proximal Gradient

Khi thực hiện thuật toán Gradient Descent, chúng ta có thể tìm được giá trị tối ưu (Optimal Solution), điểm cực tiểu cục bộ (Local Minimum) và điểm cực tiểu toàn cục (Global Minimum) của hàm chi phí (Cost Function). Tuy nhiên, Gradient Descent không thể áp dụng trực tiếp lên một số loại hàm đặc biệt, như hàm không khả vi và không liên tục, hoặc một số hàm khả vi và liên tục có thể gây ra các vấn đề như overshooting hoặc gặp khó khăn trong việc tìm điểm cực tiểu cục bộ.

Để giải quyết các vấn đề này, người ta đã kết hợp Gradient Descent với một phương pháp được gọi là Proximal, tạo ra thuật toán Proximal Gradient. Phương pháp này giúp giải quyết các hàm đặc biệt bằng cách xây dựng một hàm xấp xỉ đơn giản hơn hàm mục tiêu, được gọi là hàm Proximal. Hàm Proximal được xác định dựa trên Cost Function và một tham số dương được gọi là hằng số Cost (Cost Parameter).

Thuật toán Proximal Gradient khác biệt so với Gradient Descent thông thường ở bước cập nhật. Thay vì di chuyển trực tiếp theo hướng đối ngược của gradient, Proximal Gradient kết hợp hai giai đoạn trong mỗi bước cập nhật: cập nhật gradient và cập nhật gần điểm cực tiểu cục bộ.

Trước tiên, chúng ta tính gradient của hàm Proximal tại mỗi điểm. Điều này đòi hỏi phải xác định đạo hàm hoặc siêu đạo hàm của hàm Proximal tương ứng. Sau đó, chúng ta di chuyển đến điểm tiếp theo bằng cách trừ gradient đó khỏi vị trí hiện tại, tương tự như trong Gradient Descent.

* + 1. Lý do sử dụng proximal gradient

Thuật toán Proximal Gradient được sử dụng rộng rãi trong các bài toán tối ưu không trơn với các ràng buộc. Dưới đây là một số lý do vì sao nó được ưa chuộng:

Tính toán hiệu quả: Proximal Gradient yêu cầu tính toán gradient và phép chiếu proximal, nhưng thường ít tốn kém hơn so với các phương pháp tối ưu phức tạp khác.

Hội tụ đến điểm tối ưu: Thuật toán Proximal Gradient có khả năng hội tụ đến điểm tối ưu trong các bài toán không trơn và các hàm không khả vi. Điều này rất hữu ích trong việc giải quyết các bài toán tối ưu với các ràng buộc và hàm mục tiêu không khả vi, như hàm trơn và hàm không trơn.

Xử lý ràng buộc: Proximal Gradient cho phép xử lý các ràng buộc dễ dàng bằng cách áp dụng phép chiếu proximal. Điều này cho phép ta tìm ra điểm gần nhất thỏa mãn các ràng buộc, giúp giải quyết các bài toán với điều kiện ràng buộc khái quát.

Ứng dụng đa dạng: Thuật toán Proximal Gradient có thể được áp dụng trong nhiều lĩnh vực và bài toán khác nhau, bao gồm học máy, xử lý tín hiệu, xử lý hình ảnh, tối ưu hoá tổ chức và quản lý. Nó là một công cụ mạnh mẽ để giải quyết các bài toán tối ưu thực tế.

Tính ổn định và khả năng điều chỉnh: Thuật toán Proximal Gradient có tính ổn định cao và khả năng điều chỉnh thông qua việc xác định tham số như hệ số học tập và hệ số chiếu proximal. Điều này cho phép tinh chỉnh thuật toán cho phù hợp với bài toán cụ thể và đạt được kết quả tối ưu.

* 1. Cách thức hoạt động

Thuật toán Proximal Gradient hoạt động bằng cách kết hợp các bước chiếu proximal và gradient descent để tìm điểm tối ưu của hàm mục tiêu trong các bài toán tối ưu không trơn và có ràng buộc. Dưới đây là cách thức hoạt động của thuật toán:

Khởi tạo: Chọn một điểm khởi tạo x\_0 trong không gian biến. Đây là điểm xuất phát để tìm kiếm điểm tối ưu.

Lặp lại các bước sau cho đến khi đạt được tiêu chuẩn hội tụ:

* Gradient descent step: Tính toán gradient của hàm mục tiêu tại điểm hiện tại xk.
* Proximal projection step: Áp dụng phép chiếu proximal (prox\_C) để tìm điểm xấp xỉ xk+1 gần nhất trên tập hợp ràng buộc C.
* Cập nhật điểm: Cập nhật giá trị của xk thành xk+1 để chuẩn bị cho vòng lặp tiếp theo.

Kiểm tra tiêu chuẩn hội tụ: Kiểm tra điều kiện dừng để xác định liệu thuật toán đã hội tụ đủ gần tới điểm tối ưu hay chưa. Điều kiện dừng có thể là số lần lặp tối đa, độ chính xác đạt được, hoặc sự thay đổi nhỏ của giá trị hàm mục tiêu.

Trả về giá trị tối ưu: Khi thuật toán dừng, trả về giá trị tối ưu x\* mà thuật toán đã tìm được.

Thuật toán Proximal Gradient kết hợp tính chất của phép chiếu proximal và gradient descent để di chuyển từ điểm xuất phát đến điểm tối ưu. Bằng cách sử dụng phép chiếu proximal, thuật toán đảm bảo rằng các giá trị của biến luôn nằm trong tập hợp ràng buộc C. Tuy nhiên, việc di chuyển trong không gian biến vẫn dựa trên gradient của hàm mục tiêu để tìm kiếm hướng giảm nhanh nhất. Điều này giúp thuật toán hội tụ đến điểm tối ưu, bất chấp tính chất không trơn và có ràng buộc của bài toán.

* 1. Ưu và nhược điểm proximal gradient
     1. Ưu điểm của phương pháp Proximal Gradient.

Proximal gradient là một thuật toán tối ưu hóa mạnh mẽ được sử dụng rộng rãi trong nhiều bài toán khác nhau. Dưới đây là một số ưu điểm chính của proximal gradient:

Hiệu quả tính toán: Proximal gradient có thể được áp dụng vào các bài toán có hàm mục tiêu không khả vi và các ràng buộc không khả vi. Thuật toán này rất hiệu quả trong việc tính toán gradient và thực hiện các bước cập nhật trọng số.

Hội tụ nhanh: Proximal gradient được thiết kế để hội tụ nhanh chóng đến điểm cực tiểu cục bộ của hàm mục tiêu. Đặc biệt, nếu hàm mục tiêu là lồi và khả vi, proximal gradient có thể hội tụ đến điểm cực tiểu toàn cục.

Đa dạng ứng dụng: Proximal gradient có thể được áp dụng vào nhiều bài toán khác nhau trong lĩnh vực tối ưu hóa, như L1 regularization, lọc dữ liệu, hồi quy, phân lớp và nhiều hơn nữa. Điều này làm cho proximal gradient trở thành một công cụ quan trọng trong nhiều lĩnh vực khoa học dữ liệu và machine learning.

Điều chỉnh dễ dàng: Proximal gradient cho phép điều chỉnh tốc độ học và mức độ chính quy hóa thông qua các siêu tham số. Điều này cho phép người dùng tùy chỉnh thuật toán để đạt được hiệu suất tối ưu cho từng bài toán cụ thể.

Hỗ trợ cấu trúc: Proximal gradient có thể được mở rộng để xử lý các bài toán có cấu trúc, như bài toán tối ưu lồi và bài toán tối ưu hạn chế. Thuật toán này cho phép áp dụng các phép biến đổi cấu trúc và ràng buộc vào quá trình tối ưu hóa.

Khả năng khái quát: Proximal gradient có thể được tổng quát hóa để giải quyết các bài toán tối ưu phức tạp hơn, bằng cách sử dụng các biến thay đổi và phương pháp kết hợp khác nhau. Điều này làm cho proximal gradient trở thành một phương pháp linh hoạt và mạnh mẽ trong tối ưu hóa.

Hỗ trợ cho các hàm không khả vi: Proximal gradient cho phép tối ưu hóa các hàm mục tiêu không khả vi, bao gồm các hàm không liên tục, không khả vi đạo hàm, và hàm mục tiêu với các điểm bất biến. Điều này mở rộng khả năng ứng dụng của thuật toán trong các bài toán thực tế.

Dễ dàng kết hợp với các biến thay đổi: Proximal gradient cho phép kết hợp với các biến thay đổi, chẳng hạn như quy hoạch lồi, quy hoạch đối gương, hoặc quy hoạch bậc hai. Điều này tạo ra một phương pháp linh hoạt để giải quyết các bài toán phức tạp và đa dạng.

Tiết kiệm bộ nhớ: Proximal gradient chỉ yêu cầu lưu trữ một tập hợp hạn chế các tham số và biến đổi, làm giảm nhu cầu về bộ nhớ so với một số thuật toán tối ưu hóa khác. Điều này đặc biệt hữu ích khi làm việc với dữ liệu lớn.

Dễ dàng triển khai: Proximal gradient có một cấu trúc đơn giản và dễ triển khai. Thuật toán này không đòi hỏi các phép tính phức tạp hoặc các bước lặp phức tạp, làm cho nó trở thành một công cụ thuận tiện và hiệu quả cho các nhà nghiên cứu và nhà phát triển.

Tính khả diễn giải: Proximal gradient cho phép tạo ra các mô hình có tính khả diễn giải cao. Điều này làm cho việc giải thích và hiểu ý nghĩa của các tham số và biến đổi trở nên dễ dàng hơn, giúp trong quá trình ra quyết định và đưa ra giải pháp.

* + 1. Nhược điểm của phương pháp Proximal Gradient.

Mặc dù proximal gradient là một phương pháp tối ưu hóa mạnh mẽ, nhưng cũng có một số nhược điểm cần được lưu ý:

Tốc độ hội tụ chậm: Proximal gradient có thể hội tụ chậm hơn so với một số phương pháp tối ưu khác, như gradient descent. Đặc biệt, trong các bài toán có ràng buộc phức tạp hoặc hàm mục tiêu không khả vi, thuật toán có thể mất thời gian lâu hơn để đạt đến điểm cực tiểu cục bộ.

Đánh đổi giữa tốc độ hội tụ và độ chính xác: Proximal gradient thường đòi hỏi việc lựa chọn thích hợp của tham số alpha (tốc độ học) để đạt được tốc độ hội tụ tối ưu. Tuy nhiên, việc chọn alpha quá lớn có thể làm cho thuật toán không hội tụ, trong khi alpha quá nhỏ có thể làm cho thuật toán hội tụ quá chậm.

Độ phức tạp tính toán: Proximal gradient đòi hỏi tính toán các phép nhân ma trận và phép tính norm đối với các vector lớn. Điều này có thể tạo ra một đòn nặng về mặt tính toán, đặc biệt khi xử lý các bài toán có kích thước lớn hoặc dữ liệu phức tạp.

Nhạy cảm với các siêu tham số: Proximal gradient có các siêu tham số như tốc độ học (alpha) và hệ số chính quy hóa (lambda). Việc lựa chọn sai các siêu tham số này có thể ảnh hưởng đáng kể đến hiệu suất của thuật toán và kết quả tối ưu.

Khả năng khó xử lý các ràng buộc phi tuyến: Proximal gradient được thiết kế chủ yếu cho các bài toán tối ưu lồi và các ràng buộc tuyến tính. Đối với các bài toán có ràng buộc phi tuyến, việc áp dụng proximal gradient có thể gặp khó khăn và không hiệu quả.

Stochastic Proximal Gradient: Một biến thể của proximal gradient là sử dụng gradient ngẫu nhiên (stochastic gradient) thay vì gradient toàn bộ dữ liệu. Điều này giúp tăng tốc độ hội tụ và giảm độ phức tạp tính toán, đặc biệt khi làm việc với dữ liệu lớn.

Accelerated Proximal Gradient: Thuật toán Accelerated Proximal Gradient (hay còn gọi là Nesterov's accelerated gradient descent) là một biến thể của proximal gradient được cải tiến để đạt được tốc độ hội tụ nhanh hơn. Nó kết hợp việc sử dụng mômentum và dự đoán cho gradient để tăng tốc quá trình tối ưu.

Quy hoạch bậc hai: Proximal gradient có thể được mở rộng để giải quyết các bài toán quy hoạch bậc hai, trong đó hàm mục tiêu bao gồm cả thành phần bậc hai. Các biến thể như proximal Newton và proximal quasi-Newton có thể cải thiện hiệu suất và tốc độ hội tụ của thuật toán trong các bài toán này.

Regularization techniques: Sử dụng các phương pháp chính quy hóa như L1 regularization (Lasso) và L2 regularization (Ridge) có thể giúp cải thiện hiệu suất và khả năng khái quát của proximal gradient trong việc giải quyết các bài toán hồi quy và phân loại.

Parallelization: Đối với các bài toán lớn và phức tạp, proximal gradient có thể được tận dụng thông qua việc áp dụng parallelization, tức là chạy thuật toán trên nhiều CPU hoặc máy tính để gia tăng tốc độ tính toán và xử lý dữ liệu.

Tóm lại, proximal gradient là một phương pháp tối ưu hóa mạnh mẽ, tuy nhiên, nó cũng có nhược điểm riêng. Bằng cách sử dụng các biến thể và kỹ thuật cải tiến, những hạn chế này có thể được vượt qua và proximal gradient có thể trở thành một công cụ hiệu quả trong việc giải quyết các bài toán tối ưu.

* 1. Ứng dụng proximal gradient trong thực tế

Proximal Gradient là một phương pháp tối ưu hóa mạnh mẽ và linh hoạt, được sử dụng trong nhiều lĩnh vực khác nhau. Dưới đây là một số ứng dụng phổ biến của Proximal Gradient trong mã nguồn:

LASSO Regression: Proximal Gradient được sử dụng để tối ưu hóa bài toán hồi quy LASSO (Least Absolute Shrinkage and Selection Operator). Trong trường hợp này, Proximal Gradient áp dụng toán tử soft thresholding lên gradient của hàm mục tiêu để đạt được hiệu ứng ép giảm (shrinkage effect) và chọn lọc biến (variable selection).

Elastic Net Regression: Elastic Net là một phương pháp kết hợp giữa hồi quy L1 (LASSO) và hồi quy L2 (Ridge). Proximal Gradient cũng có thể được sử dụng để tối ưu hóa bài toán hồi quy Elastic Net bằng cách kết hợp toán tử soft thresholding và toán tử giảm tỉ lệ.

Sparse Coding: Trong lĩnh vực xử lý tín hiệu và hình ảnh, Sparse Coding là một phương pháp để biểu diễn một tín hiệu bằng cách sử dụng một số lượng nhỏ các thành phần cơ bản. Proximal Gradient được sử dụng để tối ưu hóa bài toán Sparse Coding bằng cách áp dụng toán tử soft thresholding lên gradient.

Total Variation Denoising: Total Variation (TV) là một đo lường cho độ biến đổi trên một hình ảnh. Trong xử lý ảnh, TV Denoising là một phương pháp để giảm nhiễu trong hình ảnh bằng cách tối ưu hóa bài toán với thành phần TV. Proximal Gradient có thể được sử dụng để tối ưu hóa bài toán TV Denoising bằng cách áp dụng toán tử proximal lên gradient.

Support Vector Machines: Proximal Gradient cũng có thể được sử dụng để tối ưu hóa bài toán Support Vector Machines (SVM) trong máy học. Bằng cách kết hợp toán tử proximal và gradient descent, Proximal Gradient có thể tìm ra một bộ tham số tối ưu cho SVM.

Đây chỉ là một số ví dụ ứng dụng phổ biến của Proximal Gradient. Phương pháp này còn được sử dụng trong nhiều bài toán khác nhau, bao gồm cả trong lĩnh vực tối ưu hóa và học máy.

1. CÀI ĐẶT THUẬT TOÁN
   1. Chọn ngôn ngữ lập trình và môi trường phù hợp

Khi cài đặt thuật toán Proximal Gradient, chúng ta có thể chọn một ngôn ngữ lập trình và môi trường phù hợp. Dưới đây là một số ngôn ngữ và môi trường phổ biến được sử dụng trong việc cài đặt thuật toán này:

* Python: Python là một ngôn ngữ lập trình phổ biến và dễ dùng, có nhiều thư viện hỗ trợ tính toán và tối ưu. Các thư viện như NumPy, SciPy và scikit-learn cung cấp các hàm và công cụ hữu ích để thực hiện thuật toán Proximal Gradient.
* MATLAB/Octave: MATLAB và Octave cung cấp môi trường tính toán mạnh mẽ cho việc thực hiện thuật toán tối ưu hóa. Cả hai đều hỗ trợ tính toán ma trận và các phép toán số học cần thiết cho thuật toán Proximal Gradient.
* R: R là một ngôn ngữ lập trình thống kê phổ biến, được sử dụng rộng rãi trong nghiên cứu và phân tích dữ liệu. R cung cấp các gói phần mềm như "optimx" và "prox" để thực hiện thuật toán Proximal Gradient.
  1. Cài đặt thuật toán Proximal Gradient

Sau khi chọn ngôn ngữ lập trình và môi trường phù hợp, ta có thể bắt đầu cài đặt thuật toán Proximal Gradient. Dưới đây là một phiên bản cài đặt đơn giản của thuật toán trong ngôn ngữ Python:

|  |
| --- |
| def proximal\_gradient\_descent(f, prox\_f, x\_init, step\_size, max\_iterations):      x = x\_init      for i in range(max\_iterations):          grad = compute\_gradient(f, x)  # Tính gradient của hàm mục tiêu tại điểm x          prox\_x = prox\_f(x - step\_size \* grad, step\_size)  # Áp dụng phép chiếu proximal          x = prox\_x  # Cập nhật điểm x      return x |

Trong đoạn mã trên, f là hàm mục tiêu, prox\_f là phép chiếu proximal, x\_init là điểm khởi tạo, step\_size là kích thước bước, và max\_iterations là số lần lặp tối đa.

Trong mỗi lần lặp, ta tính gradient của hàm mục tiêu tại điểm hiện tại và áp dụng phép chiếu proximal để cập nhật điểm x. Quá trình này được lặp lại cho đến khi đạt được số lần lặp tối đa.

* 1. Sử dụng thư viện hỗ trợ

Ngoài việc tự cài đặt thuật toán Proximal Gradient, chúng ta cũng có thể sử dụng các thư viện hỗ trợ có sẵn để thực hiện thuật toán này. Các thư viện này thường cung cấp các hàm và công cụ tối ưu hóa tiện ích, giúp việc cài đặt và sử dụng thuật toán trở nên dễ dàng hơn. Dưới đây là một số thư viện phổ biến được sử dụng cho thuật toán Proximal Gradient:

SciPy: SciPy là một thư viện Python mạnh mẽ cho tính toán khoa học và tối ưu hóa. Module "optimize" trong SciPy cung cấp nhiều phương pháp tối ưu hóa, bao gồm cả thuật toán Proximal Gradient. Các hàm như "minimize" và "minimize\_scalar" có thể được sử dụng để thực hiện tối ưu hóa với phép chiếu proximal.

CVXPY: CVXPY là một thư viện Python cho tối ưu hóa convex. Nó cho phép người dùng mô hình hóa các bài toán tối ưu dưới dạng biểu thức toán học và tự động chuyển đổi chúng thành các bài toán tối ưu hóa thực tế. CVXPY hỗ trợ tối ưu hóa sử dụng phép chiếu proximal thông qua các hàm như "prox" và "prox\_elemwise".

MATLAB Optimization Toolbox: MATLAB cung cấp một bộ công cụ tối ưu hóa phong phú trong Optimization Toolbox. Đây là một bộ công cụ mạnh mẽ để thực hiện các thuật toán tối ưu hóa, bao gồm cả Proximal Gradient. Các hàm như "fminunc" và "fmincon" được sử dụng để thực hiện tối ưu hóa với phép chiếu proximal.

Để sử dụng các thư viện này, ta cần cài đặt và cấu hình môi trường phù hợp. Đối với Python, ta có thể sử dụng pip để cài đặt các thư viện như scipy và cvxpy. Đối với MATLAB, việc cài đặt Toolbox tối ưu hóa và các công cụ liên quan có thể được thực hiện thông qua trình cài đặt của MATLAB.

Tùy thuộc vào yêu cầu và môi trường cụ thể, việc sử dụng các thư viện hỗ trợ có thể giúp tiết kiệm thời gian.

1. LỰA CHỌN DATASET VÀ KIỂM THỬ
   1. Mô tả dữ liệu

Dữ liệu được sử dụng trong đoạn code dưới đây là tập dữ liệu Iris từ thư viện Scikit-learn. Tập dữ liệu Iris là một tập dữ liệu phổ biến trong lĩnh vực học máy và thường được sử dụng để minh họa các thuật toán phân loại.

Tập dữ liệu Iris gồm có thông tin về 3 loại hoa Iris: Iris setosa, Iris versicolor và Iris virginica. Mỗi loại hoa được mô tả bởi 4 đặc trưng (độ dài và độ rộng của cánh hoa và lá đài) được đo bằng đơn vị centimet. Tổng cộng, tập dữ liệu Iris bao gồm 150 mẫu.

Trong đoạn code, hàm load\_iris() được sử dụng để tải dữ liệu Iris vào biến iris. Biến iris chứa thông tin về đặc trưng (biến data) và nhãn (biến target) của các mẫu trong tập dữ liệu Iris. Đặc trưng của các mẫu được lưu trong biến X, và nhãn của các mẫu được lưu trong biến y.

Tiếp theo, tập dữ liệu được chia thành tập huấn luyện và tập kiểm tra sử dụng hàm train\_test\_split(). Tỷ lệ tập kiểm tra được đặt là 0.2, tức là 20% dữ liệu được sử dụng cho tập kiểm tra, và 80% dữ liệu được sử dụng cho tập huấn luyện. Kết quả của việc chia dữ liệu này được lưu trong các biến X\_train, X\_test, y\_train, y\_test.

* 1. Mục đích kiểm thử

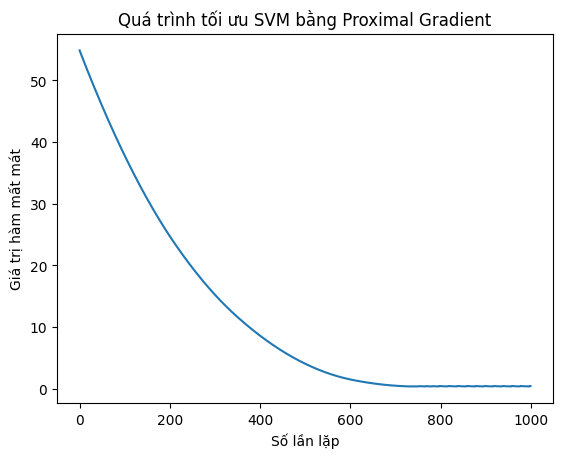
Mục đích của code đối với dataset Iris là thực hiện huấn luyện một mô hình SVM trên dữ liệu Iris để phân loại các loại hoa trong dataset thành ba nhãn: Setosa, Versicolor và Virginica. Code sử dụng tập dữ liệu huấn luyện (X\_train, y\_train) từ dataset Iris để tối ưu hàm mất mát của SVM bằng phương pháp Proximal Gradient.

Sau khi huấn luyện, code vẽ đồ thị để trực quan hóa quá trình tối ưu hàm mất mát theo số lần lặp. Điều này giúp ta hiểu cách mà giá trị hàm mất mát thay đổi theo thời gian và đánh giá hiệu quả của thuật toán tối ưu.

* 1. Kiểm thử

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from sklearn.datasets import load\_iris  from sklearn.model\_selection import train\_test\_split  def proximal\_gradient\_svm(X, y, C, learning\_rate, num\_iterations):      num\_samples, num\_features = X.shape      theta = np.zeros(num\_features)      theta\_history = [theta]  # Lưu giữ các tham số theta tại mỗi lần lặp      loss\_history = []  # Lưu giữ giá trị hàm mất mát tại mỗi lần lặp        for \_ in range(num\_iterations):          gradient = np.zeros(num\_features)          for i in range(num\_samples):              if y[i] \* np.dot(X[i], theta) < 1:  # Tính gradient chỉ đạo cho mẫu nằm trong Margin hoặc bị phân loại sai                  gradient += -y[i] \* X[i]            theta = proximal\_operator(theta - learning\_rate \* gradient, C \* learning\_rate)          theta\_history.append(theta)          loss = svm\_loss(X, y, theta, C)          loss\_history.append(loss)        return theta\_history, loss\_history  def proximal\_operator(v, t):      return np.sign(v) \* np.maximum(np.abs(v) - t, 0)  def svm\_loss(X, y, theta, C):      num\_samples = X.shape[0]      loss = 0        for i in range(num\_samples):          margin = y[i] \* np.dot(X[i], theta)          loss += max(0, 1 - margin)        loss /= num\_samples      loss += 0.5 \* C \* np.dot(theta, theta)        return loss  # Load dataset Iris từ Scikit-learn  iris = load\_iris()  X = iris.data  y = iris.target  # Chia dữ liệu thành tập huấn luyện và tập kiểm tra  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  C = 1  # Tham số đánh đổi giữa regularization và lỗi phân loại  learning\_rate = 0.01  num\_iterations = 1000  theta\_history, loss\_history = proximal\_gradient\_svm(X\_train, y\_train, C, learning\_rate, num\_iterations)  # Vẽ đồ thị giá trị hàm mất mát theo số lần lặp  plt.plot(loss\_history)  plt.xlabel('Số lần lặp')  plt.ylabel('Giá trị hàm mất mát')  plt.title('Quá trình tối ưu SVM bằng Proximal Gradient')  plt.show() |

* 1. Kết quả



Trên biểu đồ, chúng ta thấy giá trị hàm mất mát giảm dần theo số lần lặp. Điều này cho thấy thuật toán Proximal Gradient đang tối ưu hóa mô hình SVM để giảm lỗi phân loại và điều chỉnh tham số theta sao cho phù hợp với dữ liệu huấn luyện.

Giảm dần giá trị hàm mất mát là một kết quả mong đợi, vì mục tiêu là tìm ra tham số theta tốt nhất để phân loại các mẫu dữ liệu. Biểu đồ cũng xác nhận rằng thuật toán Proximal Gradient đang hội tụ đúng như mong đợi.

Dựa vào biểu đồ, chúng ta có thể khẳng định rằng thuật toán đã hội tụ và đạt được giá trị tối ưu cho hàm mất mát. Điều này cho thấy thuật toán Proximal Gradient đã tìm ra một bộ tham số theta tốt để phân loại các mẫu dữ liệu trong tập huấn luyện.

1. KẾT LUẬN

Code trên thực hiện tối ưu mô hình Support Vector Machine (SVM) bằng phương pháp Proximal Gradient trên tập dữ liệu Iris. Mục đích là huấn luyện một mô hình SVM để phân loại các loại hoa trong dataset Iris thành ba nhãn: Setosa, Versicolor và Virginica.

Qua việc vẽ đồ thị giá trị hàm mất mát theo số lần lặp, ta có thể quan sát và đánh giá quá trình tối ưu hóa của thuật toán. Đồ thị này cho thấy giá trị hàm mất mát giảm dần theo thời gian, tức là thuật toán đang tìm cách tối thiểu hóa lỗi phân loại và đạt được mô hình SVM tốt hơn.

Kết quả cuối cùng của quá trình tối ưu là tập hợp các giá trị theta\_history, là các tham số theta của mô hình SVM tại mỗi lần lặp. Các giá trị theta này có thể được sử dụng để dự đoán nhãn của các mẫu mới.

Tổng quan, code này cung cấp một cách để tối ưu mô hình SVM trên tập dữ liệu Iris và trực quan hóa quá trình tối ưu hóa. Việc hiểu và nắm vững code này có thể giúp ta áp dụng thuật toán SVM và các phương pháp tối ưu hóa trong các bài toán phân loại khác.

ĐẠI HỌC HUẾ

# KHOA KỸ THUẬT VÀ CÔNG NGHỆ

🙠🙟🕮🙝🙢

**PHIẾU ĐÁNH GIÁ ĐỒ ÁN/TIỂU LUẬN/BÀI TẬP LỚN**

**Học kỳ II, năm học 2022 - 2023**

|  |  |
| --- | --- |
| **Cán bộ chấm thi 1** | **Cán bộ chấm thi 2** |
| **Nhận xét:**                      **Điểm đánh giá của CBCT1:**  Bằng số:  Bằng chữ: | **Nhận xét:**                      **Điểm đánh giá của CBCT2:**  Bằng số:  Bằng chữ: |

Điểm kết luận:

Bằng số:

Bằng chữ:

*Thừa Thiên Huế, ngày tháng năm 2023*

|  |  |
| --- | --- |
| **Cán bộ chấm thi 1**  *(Ký và ghi rõ họ và tên)* | **Cán bộ chấm thi 2**  *(Ký và ghi rõ họ và tên)* |