**Tối ưu hóa tối thiểu tuần tự:**

**Thuật toán nhanh cho máy vector hỗ trợ đào tạo**

John C. Platt

Nghiên cứu của Microsoft jplatt@microsoft.com

Báo cáo kỹ thuật MSR-TR-98-14

ngày 21 tháng 4 năm 1998

© 1998 John Platt

# TÓM TẮT

Bài báo này đề xuất một thuật toán mới để hỗ trợ đào tạo máy vectơ: *Tối ưu hóa tối thiểu tuần tự* hoặc *SMO*. Việc đào tạo một máy vectơ hỗ trợ đòi hỏi lời giải của một bài toán tối ưu hóa lập trình bậc hai (QP) rất lớn. SMO chia vấn đề QP lớn này thành một loạt các vấn đề QP nhỏ nhất có thể. Những vấn đề QP nhỏ này được giải quyết bằng phân tích, tránh sử dụng tối ưu hóa QP số tốn thời gian như một vòng lặp bên trong. Dung lượng bộ nhớ cần thiết cho SMO là tuyến tính trong kích thước tập huấn luyện, điều này cho phép SMO xử lý các tập huấn luyện rất lớn. Vì tránh tính toán ma trận, nên SMO chia tỷ lệ ở đâu đó giữa tuyến tính và bậc hai trong kích thước tập huấn luyện cho các bài toán thử nghiệm khác nhau, trong khi thuật toán SVM phân khối tiêu chuẩn cân bằng đâu đó giữa tuyến tính và khối trong kích thước tập huấn luyện. Thời gian tính toán của SMO bị chi phối bởi đánh giá SVM, do đó SMO là nhanh nhất đối với SVM tuyến tính và các tập dữ liệu thưa thớt. Trên các tập dữ liệu thưa thớt của thế giới thực, SMO có thể nhanh hơn 1000 lần so với thuật toán phân khúc.

# 1. LỜI MỞ ĐẦU

Trong vài năm gần đây, có một sự quan tâm gia tăng đối với Máy vectơ hỗ trợ (SVM) [19] [20] [4]. Theo kinh nghiệm, SVM đã được chứng minh là mang lại hiệu suất tổng quát hóa tốt trong nhiều vấn đề như nhận dạng ký tự viết tay [12], phát hiện khuôn mặt [15], phát hiện người đi bộ [14] và phân loại văn bản [9].

Tuy nhiên, việc sử dụng SVM vẫn còn hạn chế trong một nhóm nhỏ các nhà nghiên cứu. Một lý do có thể là do các thuật toán huấn luyện cho SVM chậm, đặc biệt là đối với các bài toán lớn. Một cách giải thích khác là các thuật toán đào tạo SVM rất phức tạp, tinh vi và khó thực hiện đối với một kỹ sư trung bình.

Bài báo này mô tả một thuật toán học SVM mới về mặt khái niệm, đơn giản, dễ thực hiện, nói chung là nhanh hơn và có các đặc tính mở rộng tốt hơn cho các vấn đề SVM khó hơn so với thuật toán đào tạo SVM tiêu chuẩn. Thuật toán học SVM mới được gọi là *Tối ưu hóa tối thiểu tuần tự* (hoặc *SMO*). Thay vì các thuật toán học SVM trước đây sử dụng lập trình bậc hai số (QP) như một vòng lặp bên trong, SMO sử dụng bước QP phân tích.

Bài báo này trước tiên cung cấp một cái nhìn tổng quan về SVM và đánh giá các thuật toán đào tạo SVM hiện tại.

Sau đó, thuật toán SMO được trình bày chi tiết, bao gồm giải pháp cho bước QP phân tích, kinh nghiệm để chọn biến nào cần tối ưu hóa trong vòng lặp bên trong, mô tả cách đặt ngưỡng của SVM, một số tối ưu hóa cho các trường hợp đặc biệt, giả -code của thuật toán và mối quan hệ của SMO với các thuật toán khác.

SMO đã được thử nghiệm trên hai tập dữ liệu thế giới thực và hai tập dữ liệu nhân tạo. Bài báo này trình bày kết quả tính thời gian SMO so với thuật toán "phân khúc" tiêu chuẩn cho các tập dữ liệu này và trình bày kết luận dựa trên các định thời này. Cuối cùng, có một phụ lục mô tả nguồn gốc của việc tối ưu hóa phân tích.

## 1.1 Tổng quan về Máy hỗ trợ Vectơ



Vladimir Vapnik đã phát minh ra Máy vectơ hỗ trợ vào năm 1979 [19]. Ở dạng tuyến tính, đơn giản nhất của nó, SVM là một siêu phẳng phân tách một tập hợp các ví dụ tích cực khỏi một tập hợp các ví dụ tiêu cực với lợi nhuận tối đa (xem hình 1). Trong trường hợp tuyến tính, lề được xác định bằng khoảng cách của siêu phẳng đến gần nhất của ví dụ dương và âm. Công thức cho đầu ra của SVM tuyến tính là

rr

### u = wx⋅ -b, (1)

trong đó *w* là vectơ pháp tuyến của siêu phẳng và *x* là vectơ đầu vào. Siêu phẳng phân cách là mặt phẳng *u =*0. Các điểm gần nhất nằm trên mặt phẳng *u*= ±1. Biên độ *m* do đó

1

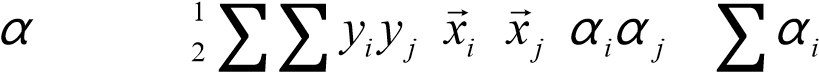
|  |  |
| --- | --- |
| *m* = .  || ||*w* 2 Việc  tối đa hóa lợi nhuận có thể được thể hiện qua bài toán tối ưu hóa sau [4]: | (2) |
| r rr min || ||1 *w* 2 tuân theo *ywxi* ( ⋅ *i* -*b*) ≥1,∀*i*, | (3) |

*wb*r, 2

trong đó *xi* là *i*thứdụ đào tạo và *yi* là đầu ra chính xác của SVM cho *i*ví dụ đào tạo thứ. Giá trị *yi* là +1 cho các ví dụ tích cực trong một lớp và –1 cho các ví dụ tiêu cực.

Sử dụng Lagrangian, bài toán tối ưu hóa này có thể được chuyển đổi thành dạng đối ngẫu, là bài toán QP trong đó hàm mục tiêu Ψ chỉ phụ thuộc vào tập hợp các nhân Lagrange α*i*,

|  |  |
| --- | --- |
| *i*=1 *j*=1 *i*=1  (trong đó *N* là số ví dụ huấn luyện), chịu sự ràng buộc bất bình đẳng, |  |
| α*i* ≥ 0,∀i,  và một chế bình đẳng tuyến tính, | (5) |
| *N yi*α*i* = 0. | (6) |

r *N N N*

phútr Ψ() = minr  ( ⋅ ) - , (4) α α

*i*=1

Có mối quan hệ một-một giữa mỗi nhân Lagrange và mỗi ví dụ huấn luyện.r

Khi nhân tử Lagrange được xác định, các vector bình thường *w* và ngưỡng *b* có thể được

|  |  |  |
| --- | --- | --- |
| bắt nguồn từ nhân tử Lagrange: |  |  |
| r *N* r  *w* = Σ *yi*α*i x*i, | rr *b* = *wx*⋅ *k* - *yk* đối với một số αk > 0. | (7) |

*i*=1

r

Bởi vì *w* có thể được tính qua phương trình (7) từ dữ liệu huấn luyện trước khi sử dụng, lượng tính toán cần thiết để đánh giá một SVM tuyến tính là không đổi trong số các vectơ hỗ trợ khác không.

Tất nhiên, không phải tất cả các tập dữ liệu đều có thể phân tách tuyến tính. Có thể không có siêu phẳng nào tách các ví dụ tích cực khỏi các ví dụ tiêu cực. Trong công thức trên, trường hợp không phân tách được sẽ tương ứng với một nghiệm vô hạn. Tuy nhiên, vào năm 1995, Cortes & Vapnik [7] đã đề xuất một sửa đổi đối với tuyên bố tối ưu hóa ban đầu (3) cho phép, nhưng lại bị phạt, việc một ví dụ không đạt được lợi nhuận chính xác. Sửa đổi đó là:

r *N* rr

1 2

min || ||r r 2 *w* +*C*∑ξ*i* tuân theo *ywxi* ( ⋅ *i* -*b*) ≥ -1 ξ*i* ,∀*i*, (8)

*wb*,,ξ *i*=1

trong đó ξ*i* là các biến chùng cho phép lỗi ký quỹ và *C* là một tham số giao dịch với tỷ lệ ký quỹ rộng với một số ít thất bại ký quỹ. Khi bài toán tối ưu hóa mới này được chuyển thành dạng đối ngẫu, nó chỉ cần thay đổi ràng buộc (5) thành ràng buộc hộp:

0 ≤ ≤ ∀α*i C*, *i*. (9)

Các biến ξ*i hoàn* toàn không xuất hiện trong công thức kép.

SVM thậm chí có thể được tổng quát hóa thành các bộ phân loại phi tuyến tính [2]. Đầu ra của SVM phi tuyến tính được tính toán rõ ràng từ các nhân Lagrange:

*N* rr

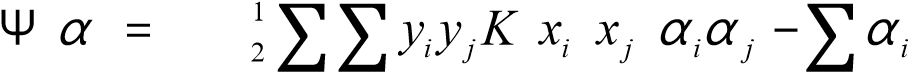
### u = ∑ y jαj K xx( j ,) -b, (10)

*j*=1

trong đó r *K* là một hàm nhân đo lường độ tương tự hoặc khoảng cách giữa vectơ đầu vàor *x* và vectơ huấn luyện được lưu trữ *x j* . Ví dụ về *K* bao gồm Gaussian, đa thức vànơron

mạngphi tuyến tính [4]. Nếu *K* là tuyến tính, thì phương trình của SVM tuyến tính (1) được phục hồi.

Các số nhân Lagrange α*i* vẫn được tính thông qua một chương trình bậc hai. Các điểm không tuyến tính làm thay đổi dạng bậc hai, nhưng hàm mục tiêu kép Ψ vẫn là bậc hai trong α:

r *N N* rr *N*

minr () minr  ( , ) , α α

*i*=1 *j*=1 *i*=1

0 ≤ ≤ ∀α*i C*, *i*, (11)

*N*

∑ *yi*α*i* = 0.

*i*=1

Bài toán QP trong phương trình (11), ở trên, là bài toán QP mà thuật toán SMO sẽ giải. Để làm cho bài toán QP trên là xác định dương, hàm nhân *K* phải tuân theo các điều kiện của Mercer [4].

Điều kiện Karush-Kuhn-Tucker (KKT) là điều kiện cần và đủ cho một điểm tối ưu của một bài toán QP xác định dương. Các điều kiện KKT cho vấn đề QP (11) đặc biệt đơn giản. Bài toán QP được giải khi, với mọi *i*:

α*i* = 0 ⇔ *yui i i* ≥1,

0 <α*i* < *C* ⇔ *yui i* = 1, (12) α*i* = *C* ⇔ *yui i* ≤1.

đâu *utôi* là đầu ra của SVM cho *i*ví dụ đào tạo thứ.Lưu ý rằng các điều kiện KKT có thể được đánh giá trên một ví dụ tại một thời điểm, điều này sẽ hữu ích trong việc xây dựng thuật toán SMO.

## 1,2 Các phương pháp trước đây để huấn luyện máy vectơ hỗ trợ

Do kích thước quá lớn, vấn đề QP (11) phát sinh từ SVM không thể dễ dàng giải quyết thông qua các kỹ thuật QP tiêu chuẩn. Dạng bậc hai trong (11) liên quan đến một ma trận có số phần tử bằng bình phương của số ví dụ đào tạo. Ma trận này không thể vừa với 128 Megabyte nếu có hơn 4000 ví dụ đào tạo.

Vapnik [19] mô tả một phương pháp để giải SVM QP, từ đó được gọi là

"phân khúc". Thuật toán phân khúc sử dụng thực tế là giá trị của dạng bậc hai là giống nhau nếu bạn loại bỏ các hàng và cột của ma trận tương ứng với số nhân Lagrange bằng không. Do đó, bài toán QP lớn có thể được chia thành một loạt các bài toán QP nhỏ hơn, mà mục tiêu cuối cùng là xác định tất cả các nhân Lagrange khác không và loại bỏ tất cả các nhân Lagrange bằng không. Ở mỗi bước, chunking giải quyết một vấn đề QP bao gồm các ví dụ sau: mọi nhân Lagrange khác 0 từ bước cuối cùng và *M* ví dụ xấu nhất vi phạm điều kiện KKT (12) [4], với một số giá trị của *M* ( xem hình 2). Nếu có ít hơn *M* ví dụ vi phạm điều kiện KKT ở một bước, tất cả các ví dụ vi phạm sẽ được thêm vào. Mỗi bài toán con QP được khởi tạo bằng kết quả của bài toán con trước đó. Ở bước cuối cùng, toàn bộ tập hợp các số nhân Lagrange khác không đã được xác định, do đó bước cuối cùng giải được bài toán QP lớn.

Chunking làm giảm nghiêm trọng kích thước của ma trận từ số lượng các ví dụ huấn luyện được bình phương xuống xấp xỉ số lượng các nhân Lagrange khác không được bình phương. Tuy nhiên, phân khúc vẫn không thể xử lý các vấn đề đào tạo quy mô lớn, vì ngay cả ma trận rút gọn này cũng không thể vừa với bộ nhớ.



**Hình 2.** Ba phương pháp thay thế để đào tạo SVM: Chunking, thuật toán Osuna và SMO. Đối với mỗi phương pháp, ba bước được minh họa. Đường mảnh nằm ngang ở mỗi bước biểu thị tập huấn luyện, trong khi các hộp dày biểu thị hệ số nhân Lagrange đang được tối ưu hóa ở bước đó. Đối với phân đoạn, một số ví dụ cố định được thêm vào mỗi bước, trong khi số nhân Lagrange bằng không sẽ bị loại bỏ ở mỗi bước. Do đó, số lượng các ví dụ được đào tạo mỗi bước có xu hướng tăng lên. Đối với thuật toán của Osuna, một số lượng cố định các ví dụ được tối ưu hóa mỗi bước: cùng một số lượng ví dụ được thêm vào và loại bỏ vấn đề ở mỗi bước. Đối với SMO, chỉ có hai ví dụ được tối ưu hóa về mặt phân tích ở mỗi bước, để mỗi bước diễn ra rất nhanh.

Năm 1997, Osuna, et al. [16] đã chứng minh một định lý gợi ý một bộ thuật toán QP hoàn toàn mới cho SVM. Định lý chứng minh rằng bài toán QP lớn có thể được chia thành một loạt bài toán nhỏ QP nhỏ hơn. Miễn là ít nhất một ví dụ vi phạm các điều kiện KKT được thêm vào các ví dụ cho bài toán con trước đó, mỗi bước sẽ giảm hàm mục tiêu tổng thể và duy trì một điểm khả thi tuân theo tất cả các ràng buộc. Do đó, một chuỗi các vấn đề QP phụ luôn có thêm ít nhất một người vi phạm sẽ được đảm bảo hội tụ. Lưu ý rằng thuật toán phân khúc tuân theo các điều kiện của định lý và do đó sẽ hội tụ.

Osuna và cộng sự. đề xuất giữ một ma trận kích thước không đổi cho mọi bài toán con QP, điều này ngụ ý thêm và xóa cùng một số lượng ví dụ ở mỗi bước [16] (xem hình 2). Sử dụng ma trận kích thước không đổi sẽ cho phép đào tạo trên các tập dữ liệu có kích thước tùy ý. Thuật toán được đưa ra trong bài báo của Osuna [16] đề xuất thêm một ví dụ và trừ một ví dụ mỗi bước. Rõ ràng điều này sẽ không hiệu quả, bởi vì nó sẽ sử dụng toàn bộ bước tối ưu hóa QP bằng số để khiến một ví dụ huấn luyện tuân theo các điều kiện KKT. Trong thực tế, các nhà nghiên cứu cộng và trừ nhiều ví dụ theo phương pháp heuristics chưa được công bố [17]. Trong mọi trường hợp, cần có bộ giải QP số cho tất cả các phương pháp này. QP số nổi tiếng là khôn lanh để đi đúng; có nhiều vấn đề về độ chính xác số cần được giải quyết.

# 2. TỐI ƯU TỐI THIỂU TỐI THIỂU TỐI THIỂU

Tối ưu hóa tối thiểu tuần tự (SMO) là một thuật toán đơn giản có thể nhanh chóng giải quyết vấn đề SVM QP mà không cần lưu trữ thêm ma trận và hoàn toàn không sử dụng các bước tối ưu hóa QP số. SMO phân tích bài toán QP tổng thể thành các bài toán con QP, sử dụng định lý Osuna để đảm bảo sự hội tụ.

Không giống như các phương pháp trước đây, SMO chọn giải quyết vấn đề tối ưu hóa nhỏ nhất có thể ở mọi bước. Đối với bài toán SVM QP tiêu chuẩn, bài toán tối ưu hóa nhỏ nhất có thể liên quan đến hai số nhân Lagrange, bởi vì các số nhân Lagrange phải tuân theo một ràng buộc bình đẳng tuyến tính. Ở mỗi bước, SMO chọn hai hệ số nhân Lagrange để cùng tối ưu hóa, tìm các giá trị tối ưu cho các hệ số nhân này và cập nhật SVM để phản ánh các giá trị tối ưu mới (xem hình 2).

Lợi thế của SMO nằm ở chỗ, việc giải hai số nhân Lagrange có thể được thực hiện bằng phân tích. Do đó, tối ưu hóa QP số hoàn toàn tránh được. Vòng lặp bên trong của thuật toán có thể được thể hiện trong một lượng ngắn mã C, thay vì gọi toàn bộ một quy trình thư viện QP. Mặc dù nhiều bài toán con tối ưu hóa hơn được giải trong quá trình giải thuật, mỗi bài toán con quá nhanh nên bài toán QP tổng thể được giải quyết nhanh chóng.

Ngoài ra, SMO hoàn toàn không yêu cầu lưu trữ ma trận bổ sung. Do đó, các bài toán đào tạo SVM rất lớn có thể nằm gọn trong bộ nhớ của máy tính cá nhân hoặc máy trạm thông thường. Bởi vì không có thuật toán ma trận nào được sử dụng trong SMO, nó ít bị ảnh hưởng bởi các vấn đề về độ chính xác số.

Có hai thành phần đối với SMO: một phương pháp phân tích để giải hai số nhân Lagrange và một phương pháp heuristic để chọn những nhân nào cần tối ưu hóa.



**Hình 1.** Hai nhân Lagrange phải đáp ứng tất cả các ràng buộc của bài toán đầy đủ. Các ràng buộc bất bình đẳng làm cho các số nhân Lagrange nằm trong hộp. Ràng buộc bình đẳng tuyến tính khiến chúng nằm trên một đường chéo. Do đó, một bước của SMO phải tìm một điểm tối ưu của hàm mục tiêu trên một đoạn đường chéo.

## 2.1 Giải cho hai nhân Lagrange

Để giải hai nhân Lagrange, trước tiên SMO tính toán các ràng buộc trên các nhân này và sau đó giải cho giá trị nhỏ nhất bị ràng buộc. Để thuận tiện, tất cả các đại lượng tham chiếu đến cấp số nhân đầu tiên sẽ có chỉ số con 1, trong khi tất cả các đại lượng tham chiếu đến cấp số nhân thứ hai sẽ có chỉ số phụ 2. Bởi vì chỉ có hai cấp số nhân, các ràng buộc có thể dễ dàng được hiển thị theo hai chiều ( xem hình 3). Các ràng buộc liên kết (9) làm cho các nhân Lagrange nằm trong một hộp, trong khi ràng buộc bình đẳng tuyến tính (6) làm cho các nhân Lagrange nằm trên một đường chéo. Do đó, cực tiểu bị ràng buộc của hàm mục tiêu phải nằm trên một đoạn đường chéo (như trong hình 3). Ràng buộc này giải thích tại sao hai là số nhân Lagrange tối thiểu có thể được tối ưu hóa: nếu SMO chỉ tối ưu hóa một hệ số, nó không thể đáp ứng ràng buộc bình đẳng tuyến tính ở mọi bước.

Các đầu của đoạn đường chéo có thể được thể hiện khá đơn giản. Không mất tính tổng quát, thuật toán đầu tiên tính hệ số nhân Lagrange thứ hai α2 và tính các điểm cuối của đoạn đường chéo theo α2. Nếu mục tiêu *y*1 không bằng mục tiêu *y*2, thì các giới hạn sau áp dụng cho α2:

|  |  |
| --- | --- |
| *L* = max (, 0 α2 -α1), *H* = min (*C*,*C* + -α2 α1).  Nếu mục tiêu *y*1 bằng mục tiêu *y*2, thì các giới hạn sau áp dụng cho α2: | (13) |
| *L* = max (0,α2 + -α1 *C*), *H* = min (*C*,α2 +α1) . | (14) |

Đạo hàm cấp hai của hàm mục tiêu dọc theo đường chéo có thể được biểu diễn dưới dạng:

rr rr rr

### η= K xx( 1, 1) + K xx( 2 , 2 ) - 2K xx( 1, 2 ). (15)

Trong các trường hợp bình thường, hàm mục tiêu sẽ là xác định dương, sẽ có cực tiểu dọc theo hướng của giới hạn bình đẳng tuyến tính và η sẽ lớn hơn 0. Trong trường hợp này, SMO tính giá trị nhỏ nhất theo hướng của ràng buộc:

new *y*2 (*E*1 - *E*2 )

α α2 = 2 + , (16)

η

trong đó *Ei* = -*ui yi* là sai số trên *tôi là*ví dụ đào tạo. Bước tiếp theo, giá trị tối thiểu bị ràng buộc được tìm thấy bằng cách cắt điểm tối thiểu không bị ràng buộc vào cuối đoạn thẳng:

|  |  |  |  |
| --- | --- | --- | --- |
| α2new, clipped =&Kα2new | if | *L*<α2new <*H*; | (17) |

%K *H* nếu *H*;

' L nếu .

Bây giờ, hãy đặt *s* = *yy*1 2 . Giá trị củagiá trị α1 được tính từmới, đã cắt, α2:

α1new =α1 + *s*(α2 -α2new, clipped ). (18)

Trong những trường hợp bất thường, η sẽ không tích cực. Mộtâm η sẽ xảy ra nếu hạt nhân *K* không tuân theo điều kiện của Mercer, điều này có thể làm cho hàm mục tiêu trở nên vô thời hạn. Một số không η có thể xảy ra ngay cả với một nhân chính xác, nếu nhiều ví dụ huấn luyện có cùng một vectơ đầu vào *x*. Trong mọi trường hợp, SMO sẽ hoạt động ngay cả khi η không dương, trong trường hợp đó, hàm mục tiêu Ψ phải được đánh giá tại mỗi đầu đoạn thẳng:

rr rr

*f*1 = *y E*1( 1 + -*b*) α1*K xx*( 1, 1) - *s*α2 *K*(*x x*1, 2 ), rr rr

*f*2 = *y*2 (*E*2 + -*b*) *s*α1*K xx*( 1, 2 ) -α2 *K xx*( 2 , 2 ),

*L*1 = +α α1 *s*( 2 - *L*),

(19)

*H*1 = +α α1 *s*( 2 - *H*),

1 2 rr 1 2 rr rr

Ψ*L* = *L f*1 1 + *Lf*2 + 2*LK xx*1 ( 1, 1) + 2 *LK xx*( 2 , 2 ) + *sLL K xx*1 ( 1, 2 ),

1 2 rr1 2 rr rr

Ψ*H* = *H f*1 1 + *Hf*2 + 2 *HK xx*1 ( 1, 1) + 2 *HK xx*( 2 , 2 ) + *sHH K xx*1 ( 1, 2 ).

SMO sẽ di chuyển các số nhân Lagrange đến điểm cuối có giá trị thấp nhất của hàm mục tiêu. Nếu hàm mục tiêu giống nhau ở cả hai đầu (trong phạm vinhỏ ε đối với sai số làm tròn) và hạt nhân tuân theo các điều kiện của Mercer, thì quá trình thu nhỏ khớp không thể đạt được tiến triển. Kịch bản đó được mô tả dưới đây.

## 2,2 Heuristics để chọn số nhân nào cần tối ưu hóa

Miễn là SMO luôn tối ưu hóa và thay đổi hai số nhân Lagrange ở mỗi bước và ít nhất một trong các số nhân Lagrange vi phạm điều kiện KKT trước bước đó, thì mỗi bước sẽ giảm hàm mục tiêu theo định lý Osuna [ 16]. Do đó, sự hội tụ được đảm bảo. Để tăng tốc độ hội tụ, SMO sử dụng phương pháp heuristics để chọn hai nhân Lagrange để cùng tối ưu hóa.

Có hai phép phỏng đoán lựa chọn riêng biệt: một cho hệ số nhân Lagrange đầu tiên và một cho hệ số thứ hai. Việc lựa chọn phương pháp heuristic đầu tiên cung cấp vòng lặp bên ngoài của thuật toán SMO. Vòng ngoài lần đầu tiên lặp lại trên toàn bộ tập huấn luyện, xác định xem mỗi ví dụ có vi phạm các điều kiện KKT hay không (12). Nếu một ví dụ vi phạm các điều kiện KKT, thì ví dụ đó đủ điều kiện để tối ưu hóa. Sau khi một lần đi qua toàn bộ tập huấn luyện, vòng lặp bên ngoài lặp lại trên tất cả các ví dụ có số nhân Lagrange không phải là 0 hoặc C (các ví dụ không bị ràng buộc). Một lần nữa, mỗi ví dụ được kiểm tra dựa trên các điều kiện KKT và các ví dụ vi phạm đủ điều kiện để tối ưu hóa. Vòng lặp bên ngoài thực hiện lặp đi lặp lại các ví dụ không bị ràng buộc cho đến khi tất cả các ví dụ không bị ràng buộc tuân theo các điều kiện KKT trong ε. Vòng lặp bên ngoài sau đó quay trở lại và lặp lại trên toàn bộ tập huấn luyện. Vòng lặp bên ngoài tiếp tục xen kẽ giữa các lần vượt qua đơn trên toàn bộ tập huấn luyện và nhiều lần vượt qua tập con không bị ràng buộc cho đến khi toàn bộ tập huấn luyện tuân theo các điều kiện KKT trong ε khi thuật toán kết thúc.

Lựa chọn đầu tiên heuristic tập trung thời gian CPU vào các ví dụ có nhiều khả năng vi phạm các điều kiện KKT nhất: tập con không bị ràng buộc. Khi thuật toán SMO tiến triển, các ví dụ ở giới hạn có khả năng ở lại giới hạn, trong khi các ví dụ không ở giới hạn sẽ di chuyển khi các ví dụ khác được tối ưu hóa. Do đó, thuật toán SMO sẽ lặp lại tập hợp con không liên kết cho đến khi tập hợp con đó tự nhất quán, sau đó SMO sẽ quét toàn bộ tập dữ liệu để tìm kiếm bất kỳ ví dụ bị ràng buộc nào đã trở thành KKT bị vi phạm do tối ưu hóa tập hợp con không liên kết.

Lưu ý rằng các điều kiện KKT được kiểm tra để nằm trong khoảng ε sau khi hoàn thành. Thông thường, ε được đặt là 10-3. Các hệ thống nhận dạng thường không cần phải đáp ứng các điều kiện KKT với độ chính xác cao: có thể chấp nhận các ví dụ trên biên dương có kết quả đầu ra từ 0,999 đến 1,001. Thuật toán SMO (và các thuật toán SVM khác) sẽ không hội tụ nhanh nếu nó được yêu cầu tạo ra đầu ra có độ chính xác rất cao.

Khi hệ số nhân Lagrange đầu tiên được chọn, SMO sẽ chọn hệ số nhân Lagrange thứ hai để tối đa hóa kích thước của bước được thực hiện trong quá trình tối ưu hóa chung. Bây giờ, đánh giá hàm nhân *K* tốn nhiều thời gian, vì vậy SMO xấp xỉ kích thước bước bằng giá trị tuyệt đối của tử số trong phương trình (16): |*E*1 - *E*2|. SMO giữ giá trị lỗilưu trong bộ nhớ cache *E được* cho mọi ví dụ không bị ràng buộc trong tập huấn luyện và sau đó chọn một lỗi để gần như tối đa hóa kích thước bước. Nếu *E*1 dương, SMO chọn một ví dụ có sai số tối thiểu *E*2. Nếu *E*1 là âm, SMO chọn một ví dụ có sai số tối đa là *E*2.

Trong các trường hợp bất thường, SMO không thể đạt được tiến bộ tích cực bằng cách sử dụng phương pháp phỏng đoán lựa chọn thứ hai được mô tả ở trên. Ví dụ: không thể đạt được tiến bộ tích cực nếu ví dụ đào tạo thứ nhất và thứ hai chia sẻ các vectơ đầu vào giống hệt nhau *x,* điều này làm cho hàm mục tiêu trở nên bán xác định. Trong trường hợp này, SMO sử dụng một hệ thống phân cấp của phép suy đoán lựa chọn thứ hai cho đến khi nó tìm thấy một cặp số nhân Lagrange có thể đạt được tiến bộ tích cực. Tiến trình tích cực có thể được xác định bằng cách tạo ra một kích thước bước khác 0 dựa trên việc tối ưu hóa chung của hai nhân Lagrange. Hệ thống phân cấp của phương pháp phỏng đoán lựa chọn thứ hai bao gồm những điều sau đây. Nếu phương pháp phỏng đoán ở trên không tạo ra tiến bộ tích cực, thì SMO bắt đầu lặp lại các ví dụ không bị ràng buộc, tìm kiếm ví dụ thứ hai có thể tạo ra tiến bộ tích cực. Nếu không có ví dụ không bị ràng buộc nào tạo ra tiến bộ tích cực, thì SMO bắt đầu lặp lại qua toàn bộ tập huấn luyện cho đến khi tìm thấy một ví dụ có tiến bộ tích cực. Cả quá trình lặp qua các ví dụ không bị ràng buộc và quá trình lặp qua toàn bộ tập huấn luyện đều được bắt đầu ở các vị trí ngẫu nhiên, để không làm sai lệch SMO đối với các ví dụ ở đầu tập huấn luyện. Trong những trường hợp cực kỳ suy thoái, không ví dụ nào sẽ làm ví dụ thứ hai thích hợp. Khi điều này xảy ra, ví dụ đầu tiên được bỏ qua và SMO tiếp tục với một ví dụ đầu tiên được chọn khác.

## 2.3 Tính toán

ngưỡng Ngưỡng *b* được tính toán lại sau mỗi bước, để các điều kiện KKT được đáp ứng cho cả hai ví dụ được tối ưu hóa. Ngưỡngsau đây *b*1 hợp lệ khimới α1 không ở giới hạn, vì nó buộc đầu ra của SVM là *y*1 khi đầu vào là *x*1:

new rr new, clipped rr

### b1 = E1 + y1(α1 -α1)K xx( 1, 1) + y2 (α2 -α2 )K xx( 1, 2 ) +b. (20)

Ngưỡngsau đây *b*2 hợp lệ khimới α2 không ở giới hạn, vì nó buộc đầu ra của SVM là *y*2 khi đầu vào là *x*2:

new rr new, clipped rr

### b2 = E2 + y1(α1 -α1)K xx( 1, 2 ) + y2 (α2 -α2 )K xx( 2 , 2 ) +b. (21)

Khi cả *b*1 và *b*2 đều hợp lệ thì chúng bằng nhau. Khi cả hai số nhân Lagrange mới đều có giới hạn và nếu *L* không bằng *H*, thì khoảng giữa *b*1 và *b*2 là tất cả các ngưỡng phù hợp với điều kiện KKT. SMO chọn ngưỡng nằm trong khoảng giữa *b*1 và *b*2.

## 2,4 Tối ưu hóa cho SVM tuyến tính r

Để tính SVM tuyến tính, chỉmột vectơ trọng số *w* cần lưu trữ, thay vì tất cả các ví dụ huấn luyện tương ứng với các số nhân Lagrange khác không. Nếu tối ưu hóa chung thành công, vectơ trọng số được lưu trữ cần được cập nhật để phản ánh các giá trị nhân Lagrange mới. Dễ dàng cập nhật vectơ trọng số do độ tuyến tính của SVM:

r new r new r new, clipped r

*w* = *w* + *y*1(α1 -α1)*x*1 + *y*2 (α2 -α2 )*x*2. (22)

## 2,5 Chi tiết mã Mã

giả bên dưới mô tả toàn bộ thuật toán SMO:

target = điểm vectơ đầu ra mong muốn =ma trận điểm huấn luyện

thủ tụcTakeBước (i1, i2) if (i1 == i2) return 0 alph1 = Lagrange nhân cho i1 y1 = target [i1 ]

E1 = Ngõ ra SVM trên điểm [i1] - y1 (kiểm tra bộ đệm lỗi) s = y1 \* y2

Tính L, H thông qua phương trình (13) và (14) nếu (L == H) trả về 0 k11 = kernel (điểm [i1], point [i1]) k12 = kernel (point [i1], point [i2]) k22 = kernel (point [i2], point [i2]) eta = k11 + k22-2 \* k12 if (eta> 0) {a2 = alph2 + y2 \* (E1-E2) / eta if (a2 <L) a2 = L else if (a2> H) a2 = H

} else

{

Lobj = hàm mục tiêu tại a2 = L Hobj = hàm mục tiêu tại a2 = H if (Lobj <Hobj-eps) a2 = L else if (Lobj> Hobj + eps) a2 = H else a2 = alph2

} if (| a2-alph2 | <eps \* (a2 + alph2 + eps)) return 0 a1 = alph1 + s \* (alph2-a2)

Cập nhật ngưỡng để phản ánh thay đổi trong hệ số Lagrange

Cập nhật vectơ trọng số để phản ánh thay đổi trong a1 & a2, nếu SVM là tuyến tính

Cập nhật bộ đệm lỗi bằng cách sử dụng hệ số Lagrange mới

Lưu trữ a1 trong mảng alpha Lưu trữ a2 trong mảng alpha trả về 1

thủ tục quy trình cuối kiểm tra Ví dụ (i2) y2 = target [i2]

alph2 = Hệ số Lagrange cho i2

E2 = đầu ra SVM tại điểm [i2] - y2 (kiểm tra trong bộ đệm lỗi) r2 = E2 \* y2

if ( (r2 <-tol && alph2 <C) || (r2> tol && alph2> 0))

{if (số khác 0 & không phải C alpha> 1)

{i1 = kết quả của lựa chọn thứ hai heuristic (phần 2.2) if takeStep (i1, i2) return 1

}

lặp lại tất cả các alpha khác 0 và không phải C, bắt đầu từ điểm ngẫu nhiên

{i1 = danh tính của alpha hiện tại nếu takeStep (i1, i2) return 1

} lặp lại trên tất cả các i1 có thể, bắt đầu tại điểm ngẫu nhiên

{i1 = biến vòng lặp nếu ( takeStep (i1, i2) return 1

}} trả về 0endprocedure

quy trình chính:

numChanged = 0; ExamAll = 1; while (numChanged> 0 | examAll)

{numChanged = 0; if (ExamAll) vòng lặp Tôi qua tất cả các ví dụ đào tạo numChanged + = ExamExample (I) else vòng lặp Tôi qua các ví dụ trong đó alpha không phải 0 & không phải C numChanged + = ExamExample (I) if (ExamAll == 1) examAll = 0 else if (numChanged == 0) examAll = 1}

## 2.6 Mối quan hệ với các thuật toán trước đây Thuật toán

SMO có liên quan đến cả SVM và thuật toán tối ưu hóa trước đó. Thuật toán SMO có thể được coi là một trường hợp đặc biệt của thuật toán Osuna, trong đó kích thước của tối ưu hóa là hai và cả hai hệ số nhân Lagrange đều được thay thế ở mỗi bước bằng các hệ số nhân mới được chọn thông qua phương pháp phỏng đoán tốt.

Thuật toán SMO có liên quan chặt chẽ với một họ các thuật toán tối ưu hóa được gọi là phương pháp Bregman [3] hoặc phương pháp hành động theo hàng [5]. Các phương pháp này giải quyết các vấn đề lập trình lồi với các ràng buộc tuyến tính. Chúng là các phương thức lặp trong đó mỗi bước chiếu điểm ban đầu hiện tại lên mỗi ràng buộc. Một phương pháp Bregman không sửa đổi không thể giải quyết vấn đề QP (11) trực tiếp, bởi vì ngưỡng trong SVM tạo ra một ràng buộc bình đẳng tuyến tính trong bài toán kép. Nếu chỉ có một ràng buộc được chiếu cho mỗi bước, thì ràng buộc bình đẳng tuyến tính sẽ bị vi phạm. Trong thuật ngữ kỹ thuật nhiều hơn, vấn đề nguyên thủy của việc giảm thiểu tiêu chuẩn củatrọng lượngr r vector *w* trên không gian kết hợp của tất cả các vector trọng lượng càng tốt *w* với ngưỡng *b* tạo ra một Bregman *D*-projection mà không có tối thiểu độc đáo [3] [6 ].

Thật thú vị khi xem xét một SVM trong đó ngưỡng *b* được giữ cố định ở mức 0, thay vì được giải quyết cho. SVM ngưỡng cố định sẽ không có ràng buộc bình đẳng tuyến tính (6). Do đó, mỗi lần chỉ cần cập nhật một hệ số Lagrange và có thể sử dụng phương pháp hành động theo hàng. Thật không may, phương pháp Bregman truyền thống vẫn không thể áp dụng cho các SVM như vậy, do các biến chùng ξ*i* trong phương trình (8). Sự hiện diện của các biến chùng khiếnBregman r *D*phép chiếutrở nên không duy nhất trong không gian kết hợp của vectơ trọng số *w* và biến chùng

ξ*i*

May mắn thay, SMO có thể được sửa đổi để giải các SVM ngưỡng cố định. SMO sẽ cập nhật các số nhân Lagrange riêng lẻ để có giá trị nhỏ nhất là Ψ cùng với thứ nguyên tương ứng. Quy tắc cập nhật là

mới *y E*1 1

α α1 = 1 + ~~rr~~ . (23)

*K xx*( 1, 1)

Phương trình cập nhật này buộc đầu ra của SVM phải là *y*1 (tương tự như phương pháp Bregman hoặc

phương pháp QP của Hildreth [10]). Sau khimới α tính toán, nó được cắt thành khoảng [0,*C*] (không giống như các phương pháp trước đây). Việc lựa chọn hệ số Lagrange nào để tối ưu hóa cũng giống như sự lựa chọn đầu tiên heuristic được mô tả trong phần 2.2.

SMO ngưỡng cố định cho SVM tuyến tính có khái niệm tương tự như quy tắc thư giãn perceptron [8], trong đó đầu ra của perceptron được điều chỉnh bất cứ khi nào có lỗi, để đầu ra nằm chính xác trên lề. Tuy nhiên, thuật toán SMO ngưỡng cố định đôi khi sẽ giảm tỷ lệ đầu vào huấn luyện trong vectơ trọng số để tối đa hóa lợi nhuận. Quy tắc thư giãn liên tục tăng số lượng đầu vào huấn luyện trong vectơ trọng lượng và do đó, không phải là lợi nhuận tối đa. SMO ngưỡng cố định cho hạt nhân Gaussian cũng liên quan đến thuật toán mạng cấp phát tài nguyên (RAN) [18]. Khi RAN phát hiện một số loại lỗi, nó sẽ cấp phát một nhân để sửa lỗi chính xác. SMO cũng sẽ hoạt động tương tự. Tuy nhiên SMO / SVM sẽ điều chỉnh chiều cao của các nhân để tối đa hóa lề trong không gian đặc trưng, ​​trong khi RAN sẽ chỉ sử dụng LMS để điều chỉnh chiều cao và trọng lượng của các nhân.

# 3 BENCHMARKING SMO

Thuật toán SMO đã được thử nghiệm dựa trên thuật toán học SVM phân khúc tiêu chuẩn trên một loạt các điểm chuẩn. Cả hai thuật toán đều được viết bằng C ++, sử dụng trình biên dịch Visual C ++ 5.0 của Microsoft. Cả hai thuật toán đều được chạy trên bộ xử lý Pentium II 266 MHz không tải chạy Windows NT 4.

Cả hai thuật toán đều được viết để khai thác sự thưa thớt của vectơ đầu vào. Cụ thể hơn, các hàm nhân dựa vào các sản phẩm dấu chấm trong vòng lặp bên trong. Nếu đầu vào là một vectơ thưa thớt, thì đầu vào có thể được lưu trữ dưới dạng một mảng thưa thớt và sản phẩm dấu chấm sẽ chỉ lặp lại trên các đầu vào khác 0, tích lũy các đầu vào khác 0 nhân với trọng số tương ứng. Nếu đầu vào là vectơ nhị phân thưa thớt, thì vị trí của các "1" trong đầu vào có thể được lưu trữ và tích chấm sẽ tính tổng các trọng số tương ứng với vị trí của "1" trong đầu vào.

Thuật toán phân khúc sử dụng thuật toán gradient liên hợp dự phóng [11] làm bộ giải QP của nó, theo đề xuất của Burges [4]. Để đảm bảo rằng thuật toán phân khúc là một điểm chuẩn công bằng, Burges đã so sánh tốc độ của mã phân khúc của mình trên Pentium II 200 MHz chạy Solaris với tốc độ của mã phân khúc điểm chuẩn (với mã sản phẩm chấm thưa đã bị tắt). Các tốc độ được tìm thấy là có thể so sánh được, điều này cho thấy rằng mã phân khúc điểm chuẩn là điểm chuẩn hợp lý.

Đảm bảo rằng mã phân khúc và mã SMO đạt được độ chính xác như nhau cần phải cẩn thận. Mã SMO và mã phân khúc đều sẽ xác định một ví dụ là vi phạm điều kiện KKT nếu đầu rahơn 10-3 cách giá trị chính xác của nóhoặc nửa khoảng trắng. Ngưỡng 10-3 được chọn là một sai số không đáng kể trong các nhiệm vụ phân loại. Mã gradient liên hợp dự phóng có ngưỡng dừng, mô tả sự cải thiện tương đối tối thiểu trong hàm mục tiêu ở mỗi bước [4]. Nếu gradient liên hợp dự kiến ​​thực hiện một bước trong đó sự cải thiện tương đối nhỏ hơn mức tối thiểu này, mã gradient liên hợp kết thúc và một bước phân khúc khác được thực hiện. Burges [4] khuyến nghị sử dụng hằng số 10-10 cho mức tối thiểu này.

Trong các thí nghiệm dưới đây, dừng gradient liên hợp dự phóng ở độ chính xác 10-10 thường để lại sai số KKT lớn hơn 10-3, đặc biệt đối với các bài toán quy mô rất lớn. Do đó, thuật toán phân khúc điểm chuẩn đã sử dụng phương pháp heuristic sau đây để đặt ngưỡng dừng gradient liên hợp. Ngưỡng bắt đầu từ 3x10-10. After every chunking step, the output is computed for all examples whose Lagrange multipliers are not at bound. These outputs are computed in order to compute the value for the threshold (see [4]). Every example suggests a proposed threshold. If the largest proposed threshold is more than 2x10-3 above the smallest proposed threshold, then the KKT conditions cannot possibly be fulfilled within 10-3. Therefore, starting at the next chunk, the conjugate gradient threshold is decreased by a factor of 3. This heuristic will optimize the speed of the conjugate gradient: it will only use high precision on the most difficult problems. For most of the tests described below, the threshold stayed at 3x10-10. The smallest threshold used was 3.7x10-12, which occurred at the end of the chunking for the largest web page classification problem.

The SMO algorithm was tested on an income prediction task, a web page classification task, and two different artificial data sets. All times listed in all of the tables are in CPU seconds.

## 3.1 Income Prediction

The first data set used to test SMO's speed was the UCI "adult" data set, which is available at ftp://ftp.ics.uci.edu/pub/machine-learning-databases/adult. The SVM was given 14 attributes of a census form of a household. The task of the SVM was to predict whether that household has an income greater than $50,000. Out of the 14 attributes, eight are categorical and six are continuous. For ease of experimentation, the six continuous attributes were discretized into quintiles, which yielded a total of 123 binary attributes, of which 14 are true. There were 32562 examples in the "adult" training set. Two different SVMs were trained on the problem: a linear SVM, and a radial basis function SVM that used Gaussian kernels with variance of 10. This variance was chosen to minimize the error rate on a validation set. The limiting value of *C* was chosen to be 0.05 for the linear SVM and 1 for the RBF/SVM. Again, this limiting value was chosen to minimize error on a validation set.

The timing performance of the SMO algorithm versus the chunking algorithm for the linear SVM on the adult data set is shown in the table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Training Set Size | SMO time | Chunking time | Number of Non-Bound Support Vectors | Number of Bound Support Vectors |
| 1605 | 0.4 | 37.1 | 42 | 633 |
| 2265 | 0.9 | 228.3 | 47 | 930 |
| 3185 | 1.8 | 596.2 | 57 | 1210 |
| 4781 | 3.6 | 1954.2 | 63 | 1791 |
| 6414 | 5.5 | 3684.6 | 61 | 2370 |
| 11221 | 17.0 | 20711.3 | 79 | 4079 |
| 16101 | 35.3 | N/A | 67 | 5854 |
| 22697 | 85.7 | N/A | 88 | 8209 |
| 32562 | 163.6 | N/A | 149 | 11558 |

The training set size was varied by taking random subsets of the full training set. These subsets are nested. The "N/A" entries in the chunking time column had matrices that were too large to fit into 128 Megabytes, hence could not be timed due to memory thrashing. The number of nonbound and the number of bound support vectors were determined from SMO: the chunking results vary by a small amount, due to the tolerance of inaccuracies around the KKT conditions.

By fitting a line to the log-log plot of training time versus training set size, an empirical scaling for SMO and chunking can be derived. The SMO training time scales as ~*N* 1.9, while chunking scales as *~N* 3.1. Thus, SMO improves empirical scaling for this problem by more than one order.

The timing performance of SMO and chunking using a Gaussian SVM is shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Training Set Size | SMO time | Chunking time | Number of Non-Bound Support Vectors | Number of Bound Support Vectors |
| 1605 | 15.8 | 34.8 | 106 | 585 |
| 2265 | 32.1 | 144.7 | 165 | 845 |
| 3185 | 66.2 | 380.5 | 181 | 1115 |
| 4781 | 146.6 | 1137.2 | 238 | 1650 |
| 6414 | 258.8 | 2530.6 | 298 | 2181 |
| 11221 | 781.4 | 11910.6 | 460 | 3746 |
| 16101 | 1784.4 | N/A | 567 | 5371 |
| 22697 | 4126.4 | N/A | 813 | 7526 |
| 32562 | 7749.6 | N/A | 1011 | 10663 |

The SMO algorithm is slower for non-linear SVMs than linear SVMs, because the time is dominated by the evaluation of the SVM. Here, the SMO training time scales as *~N* 2.1, while chunking scales as *~N* 2.9. Again, SMO's scaling is roughly one order faster than chunking. The income prediction test indicates that for real-world sparse problems with many support vectors at bound, that SMO is much faster than chunking.

## 3.2 Classifying Web Pages

The second test of SMO was on text categorization: classifying whether a web page belongs to a category or not. The training set consisted of 49749 web pages, with 300 sparse binary keyword attributes extracted from each web page. Two different SVMs were tried on this problem: a linear SVM and a non-linear Gaussian SVM, which used a variance of 10. The *C* value for the linear SVM was chosen to be 1, while the *C* value for the non-linear SVM was chosen to be 5. Again, these parameters were chosen to maximize performance on a validation set.

The timings for SMO versus chunking for a linear SVM are shown in the table, below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Training Set Size | SMO time | Chunking time | Number of Non-Bound Support Vectors | Number of Bound Support Vectors |
| 2477 | 2.2 | 13.1 | 123 | 47 |
| 3470 | 4.9 | 16.1 | 147 | 72 |
| 4912 | 8.1 | 40.6 | 169 | 107 |
| 7366 | 12.7 | 140.7 | 194 | 166 |
| 9888 | 24.7 | 239.3 | 214 | 245 |
| 17188 | 65.4 | 1633.3 | 252 | 480 |
| 24692 | 104.9 | 3369.7 | 273 | 698 |
| 49749 | 268.3 | 17164.7 | 315 | 1408 |

For the linear SVM on this data set, the SMO training time scales as *~N* 1.6, while chunking scales as *~N* 2.5. This experiment is another situation where SMO is superior to chunking in computation time.

The timings for SMO versus chunking for a non-linear SVM are shown in the table, below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Training Set Size | SMO time | Chunking time | Number of Non-Bound Support Vectors | Number of Bound Support Vectors |
| 2477 | 26.3 | 64.9 | 439 | 43 |
| 3470 | 44.1 | 110.4 | 544 | 66 |
| 4912 | 83.6 | 372.5 | 616 | 90 |
| 7366 | 156.7 | 545.4 | 914 | 125 |
| 9888 | 248.1 | 907.6 | 1118 | 172 |
| 17188 | 581.0 | 3317.9 | 1780 | 316 |
| 24692 | 1214.0 | 6659.7 | 2300 | 419 |
| 49749 | 3863.5 | 23877.6 | 3720 | 764 |

For the non-linear SVM on this data set, the SMO training time scales as *~N* 1.7, while chunking scales as *~N* 2.0. In this case, the scaling for SMO is somewhat better than chunking: SMO is a factor of between two and six times faster than chunking. The non-linear test shows that SMO is still faster than chunking when the number of non-bound support vectors is large and the input data set is sparse.

## 3.3 Artificial Data Sets

SMO was also tested on artificially generated data sets to explore the performance of SMO in extreme scenarios. The first artificial data set was a perfectly linearly separable data set. The input data was random binary 300-dimensional vectors, with a 10% fraction of “1” inputs. A 300-dimensional weight vector was generated randomly in [-1,1]. If the dot product of the weight with an input point is greater than 1, then a positive label is assigned to the input point. If the dot product is less than –1, then a negative label is assigned. If the dot product lies between –1 and 1, the point is discarded. A linear SVM was fit to this data set.

The linearly separable data set is the simplest possible problem for a linear SVM. Not surprisingly, the scaling with training set size is excellent for both SMO and chunking. The running times are shown in the table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Training Set Size | SMO time | Chunking time | Number of Non-Bound Support Vectors | Number of Bound Support Vectors |
| 1000 | 15.3 | 10.4 | 275 | 0 |
| 2000 | 33.4 | 33.0 | 286 | 0 |
| 5000 | 103.0 | 108.3 | 299 | 0 |
| 10000 | 186.8 | 226.0 | 309 | 0 |
| 20000 | 280.0 | 374.1 | 329 | 0 |

Here, the SMO running time scales as *~N,* which is slightly better than the scaling for chunking, which is *~N* 1.2. For this easy sparse problem, therefore, chunking and SMO are generally comparable. Both algorithms were trained with *C* set to 100. The chunk size for chunking is set to be 500.

The acceleration of both the SMO algorithm and the chunking algorithm due to the sparse dot product code can be measured on this easy data set. The same data set was tested with and without the sparse dot product code. In the case of the non-sparse experiment, each input point was stored as a 300-dimensional vector of floats. The result of the sparse/non-sparse experiment is shown in the table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Training Set Size | SMO time  (sparse) | SMO time  (non-sparse) | Chunking time (sparse) | Chunking time (non-sparse) |
| 1000 | 15.3 | 145.1 | 10.4 | 11.7 |
| 2000 | 33.4 | 345.4 | 33.0 | 36.8 |
| 5000 | 103.0 | 1118.1 | 108.3 | 117.9 |
| 10000 | 186.8 | 2163.7 | 226.0 | 241.6 |
| 20000 | 280.0 | 3293.9 | 374.1 | 397.0 |

For SMO, use of the sparse data structure speeds up the code by more than a factor of 10, which shows that the evaluation time of the SVM totally dominates the SMO computation time. The sparse dot product code only speeds up chunking by a factor of approximately 1.1, which shows that the evaluation of the numerical QP steps dominates the chunking computation. For the linearly separable case, there are absolutely no Lagrange multipliers at bound, which is the worst case for SMO. Thus, the poor performance of non-sparse SMO versus non-sparse chunking in this experiment should be considered a worst case.

The sparse versus non-sparse experiment shows that part of the superiority of SMO over chunking comes from the exploitation of sparse dot product code. Fortunately, many real-world problems have sparse input. In addition to the real-word data sets described in section 3.1 and section 3.2, any quantized or fuzzy-membership-encoded problems will be sparse. Also, optical character recognition [12], handwritten character recognition [1], and wavelet transform coefficients of natural images [13] [14] tend to be naturally expressed as sparse data.

The second artificial data set stands in stark contrast to the first easy data set. The second set is generated with random 300-dimensional binary input points (10% “1”) and random output labels. The SVMs are thus fitting pure noise. The *C* value was set to 0.1, since the problem is fundamentally unsolvable. The results for SMO and chunking applied to a linear SVM are shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Training Set Size | SMO time | Chunking time | Number of Non-Bound Support Vectors | Number of Bound Support Vectors |
| 500 | 1.0 | 6.4 | 162 | 263 |
| 1000 | 3.5 | 57.9 | 220 | 632 |
| 2000 | 15.7 | 593.8 | 264 | 1476 |
| 5000 | 67.6 | 10353.3 | 283 | 4201 |
| 10000 | 187.1 | N/A | 293 | 9034 |

Scaling for SMO and chunking are much higher on the second data set. This reflects the difficulty of the problem. The SMO computation time scales as *~N* 1.8, while the chunking computation time scales as *~N* 3.2. The second data set shows that SMO excels when most of the support vectors are at bound. Thus, to determine the increase in speed caused by the sparse dot product code, both SMO and chunking were tested without the sparse dot product code:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Training Set Size | SMO time  (sparse) | SMO time  (non-sparse) | Chunking time (sparse) | Chunking time (non-sparse) |
| 500 | 1.0 | 6.0 | 6.4 | 6.8 |
| 1000 | 3.5 | 21.7 | 57.9 | 62.1 |
| 2000 | 15.7 | 99.3 | 593.8 | 614.0 |
| 5000 | 67.6 | 400.0 | 10353.3 | 10597.7 |
| 10000 | 187.1 | 1007.6 | N/A | N/A |

In the linear SVM case, sparse dot product code sped up SMO by about a factor of 6, while chunking sped up only minimally. In this experiment, SMO is faster than chunking even for nonsparse data.

The second data set was also tested using Gaussian SVMs that have a variance of 10. The *C* value is still set to 0.1. The results for the Gaussian SVMs are presented in the two tables below:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Training Set Size | | SMO time | | Chunking time | | Number of Non-Bound Support Vectors | | | Number of Bound Support Vectors | |
| 500 | | 5.6 | | 5.8 | | 22 | | | 476 | |
| 1000 | | 21.1 | | 41.9 | | 82 | | | 888 | |
| 2000 | | 131.4 | | 635.7 | | 75 | | | 1905 | |
| 5000 | | 986.5 | | 13532.2 | | 30 | | | 4942 | |
| 10000 | | 4226.7 | | N/A | | 48 | | | 9897 | |
|  | Training Set Size | | SMO time  (sparse) | | SMO time  (non-sparse) | | Chunking time (sparse) | Chunking time (non-sparse) | |
|  | 500 | | 5.6 | | 19.8 | | 5.8 | 6.8 | |
|  | 1000 | | 21.1 | | 87.8 | | 41.9 | 53.0 | |
|  | 2000 | | 131.4 | | 554.6 | | 635.7 | 729.3 | |
|  | 5000 | | 986.5 | | 3957.2 | | 13532.2 | 14418.2 | |
|  | 10000 | | 4226.7 | | 15743.8 | | N/A | N/A | |

For the Gaussian SVM's fit to pure noise, the SMO computation time scales as *~N* 2.2, while the chunking computation time scales as *~N* 3.4. The pure noise case yields the worst scaling so far, but SMO is superior to chunking by more than one order in scaling. The total run time of SMO is still superior to chunking, even when applied to the non-sparse data. The sparsity of the input data yields a speed up of approximately a factor of 4 for SMO for the non-linear case, which indicates that the dot product speed is still dominating the SMO computation time for non-linear

SVMs

# 4 CONCLUSIONS

SMO is an improved training algorithm for SVMs. Like other SVM training algorithms, SMO breaks down a large QP problem into a series of smaller QP problems. Unlike other algorithms, SMO utilizes the smallest possible QP problems, which are solved quickly and analytically, generally improving its scaling and computation time significantly.

SMO was tested on both real-world problems and artificial problems. From these tests, the following can be deduced:

* SMO can be used when a user does not have easy access to a quadratic programming package and/or does not wish to tune up that QP package.
* SMO does very well on SVMs where many of the Lagrange multipliers are at bound.
* SMO performs well for linear SVMs because SMO's computation time is dominated by SVM evaluation, and the evaluation of a linear SVM can be expressed as a single dot product, rather than a sum of linear kernels.
* SMO performs well for SVMs with sparse inputs, even for non-linear SVMs, because the kernel computation time can be reduced, directly speeding up SMO. Because chunking spends a majority of its time in the QP code, it cannot exploit either the linearity of the SVM or the sparseness of the input data.
* SMO will perform well for large problems, because its scaling with training set size is better than chunking for all of the test problems tried so far.

For the various test sets, the training time of SMO empirically scales between *~N* and *~N* 2.2. The training time of chunking scales between *~N* 1.2 and *~N* 3.4. The scaling of SMO can be more than one order better than chunking. For the real-world test sets, SMO can be a factor of 1200 times faster for linear SVMs and a factor of 15 times faster for non-linear SVMs.

Because of its ease of use and better scaling with training set size, SMO is a strong candidate for becoming the standard SVM training algorithm. More benchmarking experiments against other QP techniques and the best Osuna heuristics are needed before final conclusions can be drawn.

# ACKNOWLEDGEMENTS

Thanks to Lisa Heilbron for assistance with the preparation of the text. Thanks to Chris Burges for running a data set through his projected conjugate gradient code. Thanks to Leonid Gurvits for pointing out the similarity of SMO with Bregman methods.

# REFERENCES

1. Bengio, Y., LeCun, Y., Henderson, D., "Globally Trained Handwritten Word Recognizer using Spatial Representation, Convolutional Neural Networks and Hidden Markov Models," *Advances in Neural Information Processing Systems,* 5, J. Cowan, G. Tesauro, J. Alspector, eds., 937-944, (1994).
2. Boser, BE, Guyon, IM, Vapnik, V., "A Training Algorithm for Optimal Margin Classifiers", *Fifth Annual Workshop on Computational Learning Theory,* ACM, (1992).
3. Bregman, LM, "The Relaxation Method of Finding the Common Point of Convex Sets and Its Application to the Solution of Problems in Convex Programming," *USSR Computational Mathematics and Mathematical Physics,* 7:200-217, (1967).
4. Burges, CJC, "A Tutorial on Support Vector Machines for Pattern Recognition," submitted to Data Mining and Knowledge Discovery, http://svm.research.belllabs.com/SVMdoc.html, (1998).
5. Censor, Y., "Row-Action Methods for Huge and Sparse Systems and Their Applications", *SIAM Review,* 23(4):444-467, (1981).
6. Censor, Y., Lent, A., "An Iterative Row-Action Method for Interval Convex Programming," *J. Optimization Theory and Applications,* 34(3):321-353, (1981).
7. Cortes, C., Vapnik, V., "Support Vector Networks," *Machine Learning*, 20:273-297, (1995).
8. Duda, RO, Hart, PE, *Pattern Classification and Scene Analysis*, John Wiley & Sons, (1973).
9. Joachims, T., "Text Categorization with Support Vector Machines", LS VIII Technical Report, No. 23, University of Dortmund, ftp://ftp-ai.informatik.unidortmund.de/pub/Reports/report23.ps.Z, (1997).
10. Hildreth, C., "A Quadratic Programming Procedure," *Naval Research Logistics Quarterly,* 4:79-85, (1957).
11. Gill, PE, Murray, W., Wright, MH, *Practical Optimization,* Academic Press, (1981).
12. LeCun, Y., Jackel, LD, Bottou, L., Cortes, C., Denker, JS, Drucker, H., Guyon, I., Muller,

UA, Sackinger, E., Simard, P. and Vapnik, V., "Learning Algorithms for Classification: A Comparison on Handwritten Digit Recognition," *Neural Networks: The Statistical Mechanics Perspective*, Oh, JH, Kwon, C. and Cho, S. (Ed.), World Scientific, 261-276, (1995).

1. Mallat, S., *A Wavelet Tour of Signal Processing,* Academic Press, (1998).
2. Oren, M., Papageorgious, C., Sinha, P., Osuna, E., Poggio, T., "Pedestrian Detection Using Wavelet Templates," *Proc. Computer Vision and Pattern Recognition '97,* 193-199, (1997).
3. Osuna, E., Freund, R., Girosi, F., "Training Support Vector Machines: An Application to Face Detection," *Proc. Computer Vision and Pattern Recognition '97*, 130-136, (1997).
4. Osuna, E., Freund, R., Girosi, F., "Improved Training Algorithm for Support Vector Machines," *Proc. IEEE NNSP '97*, (1997).
5. Osuna, E., Personal Communication.
6. Platt, JC, "A Resource-Allocating Network for Function Interpolation," *Neural Computation,* 3(2):213-225, (1991).
7. Vapnik, V., *Estimation of Dependences Based on Empirical Data*, Springer-Verlag, (1982).
8. Vapnik, V., *The Nature of Statistical Learning Theory,* Springer-Verlag, (1995).

# APPENDIX: DERIVATION OF TWO-EXAMPLE MINIMIZATION

Each step of SMO will optimize two Lagrange multipliers. Without loss of generality, let these two multipliers be α1 and α2. The objective function Ψ from equation (11) can thus be written as

Ψ= 21 *K*11α12 + 21 *K*22α22 + *sK*12α1α2 + *y*1α1 1*v* + *y*2α2*v*2 − − +Ψα1 α2 constant , (24)

where

rr

*Kij* = *K xx*( *i* , *j* ),

## *vi* = ∑*N y j* *j Kij* = *ui* +*b* − *y K i* − *y K i* , (25)

*j*=3

and the starred variables indicate values at the end of the previous iteration. Ψconstant are terms that do not depend on either α1 or α2.

Each step will find the minimum along the line defined by the linear equality constraint (6). That linear equality constraint can be expressed as

α1 + *s*α2 =α1\* + *s*α\*2 = *w*. (26)

The objective function along the linear equality constraint can be expressed in terms on α2 alone:

Ψ= 21 *K*11(*w*− *s*α2 )2 + 21 *K*22α22 + *sK*12 (*w*− *s*αα2 ) 2

(27)

+ *yw*1( − *s*α2 )*v*1 − +*w s*α α α2 + *y*2 2*v*2 − +2 Ψconstant .

The extremum of the objective function is at *d*Ψ

 = −*sK*11(*w* − *s*α2 ) + *K*22α α2 − *K*12 2 + *sK*12 (*w*− *s*α2 ) − *yv*2 1 + *s*+ *yv*2 2 −1= 0. (28) *d*α2

If the second derivative is positive, which is the usual case, then the minimum of α2 can be expressed as

α2 (*K*11 + *K*22 − 2*K*12 ) = *s K*( 11 − *K*12 )*w*+ *yv*2 ( 1 − + −*v*2 ) 1 *s*. (29)

Expanding the equations for *w* and *v* yields

α2 (*K*11 + *K*22 − 2*K*12 ) =α\*2 (*K*11 + *K*22 −2*K*12 ) + *yu*2 ( 1 −*u*2 + *y*2 − *y*1). (30)

More algebra yields equation (16).