

## Bài 1: Giới thiệu về Machine Learning

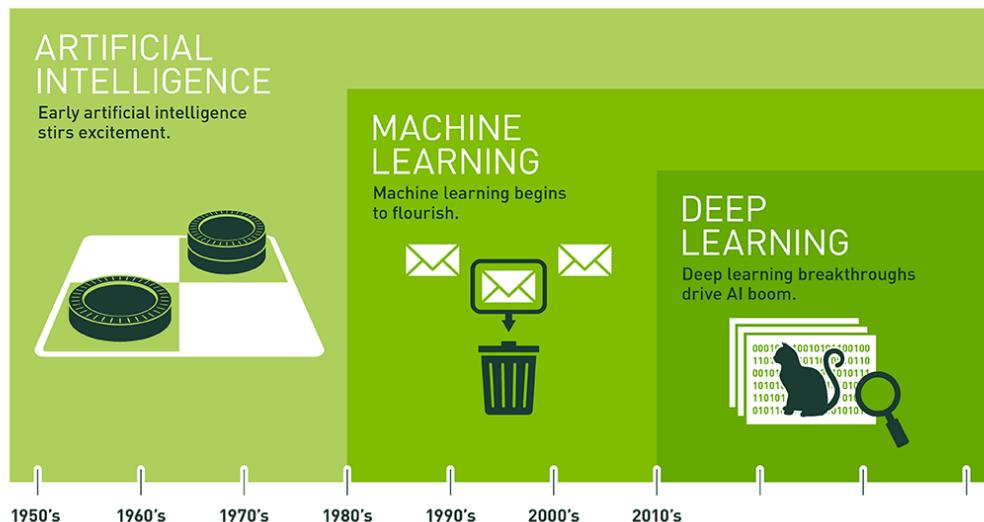
General (/tags#General)

Dec 26, 2016

Những năm gần đây, AI - Artificial Intelligence (Trí Tuệ Nhân Tạo), và cụ thể hơn là Machine Learning (Học Máy hoặc Máy Học) nổi lên như một bỗng chốc của cuộc cách mạng công nghiệp lần thứ tư (1 - động cơ hơi nước, 2 - năng lượng điện, 3 - công nghệ thông tin). Trí Tuệ Nhân Tạo đang len lỏi vào mọi lĩnh vực trong đời sống mà có thể chúng ta không nhận ra. Xe tự hành của Google và Tesla, hệ thống tự tag khuôn mặt trong ảnh của Facebook, trợ lý ảo Siri của Apple, hệ thống gợi ý sản phẩm của Amazon, hệ thống gợi ý phim của Netflix, máy chơi cờ vây AlphaGo của Google DeepMind, ..., chỉ là một vài trong vô vàn những ứng dụng của AI/Machine Learning. (Xem thêm Jarvis - trợ lý thông minh cho căn nhà của Mark Zuckerberg (<https://www.facebook.com/zuck/posts/10103351073024591>))

Machine Learning là một tập con của AI. Theo định nghĩa của Wikipedia, *Machine learning is the subfield of computer science that “gives computers the ability to learn without being explicitly programmed”*. Nói đơn giản, Machine Learning là một lĩnh vực nhỏ của Khoa Học Máy Tính, nó có khả năng tự học hỏi dựa trên dữ liệu đưa vào mà không cần phải được lập trình cụ thể. Bạn Nguyễn Xuân Khánh tại đại học Maryland đang viết một cuốn sách về Machine Learning bằng tiếng Việt khá thú vị, các bạn có thể tham khảo bài Machine Learning là gì? (<https://ml-book-vn.khanhxnguyen.com/intro.html>).

Những năm gần đây, khi mà khả năng tính toán của các máy tính được nâng lên một tầm cao mới và lượng dữ liệu khổng lồ được thu thập bởi các hãng công nghệ lớn, Machine Learning đã tiến thêm một bước dài và một lĩnh vực mới được ra đời gọi là Deep Learning (Học Sâu - *thực sự tôi không muốn dịch từ này ra tiếng Việt*). Deep Learning đã giúp máy tính thực thi những việc tưởng chừng như không thể vào 10 năm trước: phân loại cá ngan vật thể khác nhau trong các bức ảnh, tự tạo chú thích cho ảnh, bắt chước giọng nói và chữ viết của con người, giao tiếp với con người, hay thậm chí cả sáng tác văn hay âm nhạc (Xem thêm 8 Inspirational Applications of Deep Learning (<http://machinelearningmastery.com/inspirational-applications-deep-learning/>))



Mối quan hệ giữa AI, Machine Learning và Deep Learning.

(Nguồn: What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning? (<https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>))

## Mục đích viết Blog

Nhu cầu về nhân lực ngành Machine Learning (Deep Learning) đang ngày một cao, kéo theo đó nhu cầu học Machine Learning trên thế giới và ở Việt Nam ngày một lớn. Cá nhân tôi cũng muốn hệ thống lại kiến thức của mình về lĩnh vực này để chuẩn bị cho tương lai (đây là một trong những mục tiêu của tôi trong năm 2017). Tôi sẽ cố gắng đi từ những thuật toán cơ bản nhất của Machine Learning kèm theo các ví dụ và mã nguồn trong mỗi bài viết. Tôi sẽ viết 1-2 tuần 1 bài (việc viết các công thức toán và code trên blog thực sự tốn nhiều thời gian hơn tôi từng nghĩ). Đồng thời, tôi cũng mong muốn nhận được phản hồi của bạn đọc để qua những thảo luận, tôi và các bạn có thể nắm bắt được các thuật toán này.

Với những từ chuyên ngành, tôi sẽ dùng song song cả tiếng Anh và tiếng Việt, tuy nhiên sẽ ưu tiên tiếng Anh vì thuận tiện hơn cho các bạn trong việc tra cứu các tài liệu tiếng Anh.

Khi chuẩn bị các bài viết, tôi sẽ giả định rằng bạn đọc có một chút kiến thức về Đại Số Tuyến Tính (Linear Algebra), Xác Suất Thống Kê (Probability and Statistics) và có kinh nghiệm về lập trình Python. Nếu bạn chưa có nhiều kinh nghiệm về các lĩnh vực này, đừng quá lo lắng vì mỗi bài sẽ chỉ sử dụng một vài kỹ thuật cơ bản. Hãy để lại câu hỏi của bạn ở phần Comment bên dưới mỗi bài, tôi sẽ thảo luận thêm với các bạn.

Trong bài tiếp theo của blog này (/2016/12/27/categories/), tôi sẽ giới thiệu về các nhóm thuật toán Machine learning cơ bản. Mời các bạn theo dõi.

## Tham khảo thêm

### Các khóa học

#### Tiếng Anh

1. Machine Learning với thầy Andrew Ng trên Coursera (<https://www.coursera.org/learn/machine-learning>) (*Khóa học nổi tiếng nhất về Machine Learning*)
2. Deep Learning by Google trên Udacity (<https://www.udacity.com/course/deep-learning--ud730>) (*Khóa học nâng cao hơn về Deep Learning với Tensorflow*)
3. Machine Learning mastery (<http://machinelearningmastery.com/>) (*Các thuật toán Machine Learning cơ bản*)

### Các trang Machine Learning tiếng Việt

1. Machine Learning trong Xử Lý Ngôn Ngữ Tự Nhiên - Nhóm Đông Du Nhật Bản (<http://viet.jnlp.org/kien-thuc-co-ban-ve-xu-ly-ngon-ngu-tu-nhien/machine-learning-trong-nlp>)
2. Machine Learning cho người mới bắt đầu - Ông Xuân Hồng JAIST (<https://ongxuanhong.wordpress.com/>).
3. Machine Learning book for Vietnamese - Nguyễn Xuân Khánh University of Maryland (<https://ml-book-vn.khanhxnguyen.com/>)

Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (<https://www.facebook.com/groups/257768141347267/>) để nhận được câu trả lời sớm hơn.

Bạn đọc có thể ủng hộ blog qua 'Buy me a coffee' (/buymeacoffee/) ở góc trên bên trái của blog.

Tôi đang trong quá trình viết cuốn sách 'Machine Learning cơ bản I', các bạn có thể đặt trước tại đây (/ebook/). Cảm ơn bạn.

Bài 2: Phân nhóm các thuật toán Machine Learning » (/2016/12/27/categories/)

41 Comments tiepvu

 1 Login ▾

 Recommend 17  Share

Sort by Best ▾

 Join the discussion...

LOG IN WITH OR SIGN UP WITH DISQUS [?](#)

Name

Phong Phan • 7 months ago

Cảm ơn anh rất nhiều

3 ^ | v • Reply • Share >

Cát Thành • 2 months ago

bạn đang là người tạo bàn đạp cho sự phát triển của cộng đồng IT Việt Nam nói chung và lĩnh vực AI nói riêng đấy !! Tiếc tục phát huy nhé, mọi người luôn ủng hộ bạn.

2 ^ | v • Reply • Share >

Nguyễn Anh Nguyên • 3 months ago

Tôi nhận thấy khởi động của anh là thật sự tốt cho Việt Nam. Chúng ta tụt hơi xa, phải guồng chân. Tôi đã 50, nhưng tôi vẫn học mỗi ngày môn này.

2 ^ | v • Reply • Share >

Anh Viet Dang • 4 months ago

Minh là sinh viên kỹ thuật hóa học, mình cũng muốn tìm hiểu về AI, các bạn sinh viên học Khoa học máy tính học những môn cơ sở ngành nào để có thể hiểu được những vấn đề này vậy,

Hiện tại thì mình cũng đã học đại số tuyến tính và xác suất thống kê, lập trình thì mình mới biết C, chưa biết python là gì

2 ^ | v · Reply · Share >

**Tiep Vu Huu** Mod → Anh Viet Dang • 3 months ago

Chào bạn, xin lỗi vì trả lời bạn muộn.

Theo mình thì bạn nên học python vì tài liệu về Machine Learning có code chủ yếu được viết trên python. Còn về toán thì là Đại Số Tuyển Tính và Xác Suất Thống Kê, việc học kỹ về Khoa học máy tính cũng chưa cần thiết lắm cho người mới bắt đầu.

Nếu bạn có thêm câu hỏi, bạn có thể lên trên Forum đặt câu hỏi để mọi người cùng tham gia trao đổi, cảm ơn bạn.

1 ^ | v · Reply · Share >

**Anh Viet Dang** → Tiep Vu Huu • 3 months ago

cụ thể python a viết = phần mềm gì thế, hiện tại e đã cài python2 và có thể dùng anaconda 2 để viết ko

1 ^ | v · Reply · Share >

**Tiep Vu Huu** Mod → Anh Viet Dang • 3 months ago

Anh viết trên Sublime Text thôi.

1 ^ | v · Reply · Share >

**Duy Thanh Tran** • 9 months ago

Một trong những blog viết về ML hay và dễ tiếp cận nhất mà em từng đọc. Cảm ơn anh rất nhiều!

2 ^ | v · Reply · Share >

**Tiep Vu Huu** Mod → Duy Thanh Tran • 9 months ago

Nếu bạn thấy blog hữu ích, hãy share để nhiều người được biết.

Cảm ơn bạn.

^ | v · Reply · Share >

**Thành Trần Công** • 6 months ago

Chào anh em thấy các bài viết của anh rất tâm đắc , về đại số tuyến tính và xác suất thống Kê thì em thấy cở nào cũng dính Toán Cao Cấp. thống kê thì ko rồi . Hiện tại em đang ôn lại một số toán , mà anh cho em hỏi là có cần phải master toán tuyến tính ko anh . Em hiện tại đang tính học trên khanacademy .

2 ^ | v · Reply · Share >

**Pê Hú** • a month ago

Cảm ơn anh rất nhiều! Kiến thức rất bổ ích!

P/s: Báo tuổi trẻ đưa ta tới đây :)

1 ^ | v · Reply · Share >

**Tiep Vu Huu** Mod → Pê Hú • a month ago

Chào bạn. Bạn cho mình xin link báo Tuổi trẻ được không :). Cảm ơn bạn.

^ | v · Reply · Share >

**Pê Hú** → Tiep Vu Huu • 25 days ago

Một trong nhiều trang giải thích ML bằng tiếng Việt hay có thể kể đến là: machinelearningcoban.com của nghiên cứu sinh Vũ Hữu Tiệp (Penn State University, Hoa Kỳ) :))

^ | v · Reply · Share >

**Pê Hú** → Tiep Vu Huu • 25 days ago

em đọc báo giấy a ori! Báo tuổi trẻ, mục đối thoại tuổi 20. em không nhớ số, nhưng là cuối tháng 9/2017  
search thì thấy web a trên này! :))

<http://www.vietnamcentrepoli...>

^ | v · Reply · Share >

**Tuân Nguyễn** • 3 months ago

Cảm ơn anh :D

1 ^ | v · Reply · Share >

**Văn Sỹ** • 3 months ago

cảm ơn bạn rất nhiều, những đóng góp cho cộng đồng IT Việt Nam

1 ^ | v · Reply · Share >

**thib1 luyện** • 4 months ago

Cảm ơn Bạn rất nhiều về những bài viết rất hay!

1 ^ | v · Reply · Share >

**Ran Taro** • 6 months ago

cảm ơn vì blog của a  
 1 ^ | v • Reply • Share >

**Luongphat Nguyen** • 6 months ago

Chào anh,  
 Blog của anh thật hay và em rất cảm ơn anh. Em đã học hỏi được rất nhiều từ trang blog của anh.  
 1 ^ | v • Reply • Share >

**Tuan Nguyen Thanh** • 6 months ago

Chào bạn,  
 Cảm ơn bạn về loạt bài viết rất hay và bổ ích cho những ai mới bắt đầu tìm hiểu Machine Learning.  
 Minh chỉ góp ý về một số lỗi đánh máy trong bài viết:  
 "Đồng thời, tôi cũng mong muôn nhận" -> chữ thời.  
 "Xác Suất Thống Kê (Probability and Statistics)" -> chữ Suất  
 Hy vọng được học hỏi thêm nhiều từ bạn.  
 1 ^ | v • Reply • Share >

**Tiep Vu Huu** Mod → Tuan Nguyen Thanh • 6 months ago

Cảm ơn bạn đã quan tâm tới blog và chỉ ra các lỗi đánh máy. Tôi đã sửa lại những chỗ đó.

Chúc bạn cuối tuần vui vẻ.  
 ^ | v • Reply • Share >

**do chinh** • 7 months ago

thưa anh. em thấy anh thật là tâm huyết và blog cũng thật tuyệt vời. hiện tại em đang là một sinh viên năm 2 vẫn đang học đại cương. chuyên nghành của em là cơ khí hàng không. về lập trình hay kiến thức toán học vẫn không chắc chắn và em chưa biết gì về machine learning, vậy em có nên tìm hiểu về cái này được không. hay anh có thể cho em một số lời khuyên khi học về cái này ạ. em cảm ơn anh  
 1 ^ | v • Reply • Share >

**Tiep Vu Huu** Mod → do chinh • 7 months ago

Thực ra Machine Learning chỉ cần yêu cầu kiến thức về Đại số tuyến tính và Xác suất thống kê, cộng thêm một chút kiến thức lập trình nữa. Nếu em đã học các môn này rồi thì có thể bắt đầu. Càng sớm càng đỡ quên.

1 ^ | v • Reply • Share >

**do chinh** Tiep Vu Huu • 6 months ago

vâng.cảm ơn anh ạ. hi vọng blog sẽ ngày càng phát triển và có những bài hay howna j  
 ^ | v • Reply • Share >

**Thanh Nguyen** • 9 months ago

Chào anh,  
 Các bài viết của anh rất hay, rõ ràng và dễ hiểu. bản thân e học tập và làm việc trong mảng Finance, cũng có mối quan tâm đặc biệt về Machine Learning nhưng vì tự học nên vẫn rất mơ hồ chưa biết bắt đầu từ đâu. Nhưng hôm nay đọc dc 1 bài viết của a thì e đọc 1 mạch từ tối đến sáng luôn. Nhờ các bài viết có hệ thống của anh với những ngôn từ cách dẫn dắt rất hiệu, giúp e có 1 cách tiếp cận vào mảng này và tạo động lực cho e rất nhiều :)  
 Em cảm ơn anh.  
 Chúc anh và gia đình năm mới mạnh khỏe, vui vẻ và thành công. Mong anh tiếp tục viết thêm nhiều bài nữa, với tốc độ như thế này và cách tiếp cận từ với những ví dụ trực quan. Và đừng drop giữa chừng nhé ạ :)) Nếu a cần giúp đỡ gì trong việc viết bài, kiểu sai vặt thôi cũng dc, e sẵn sàng giúp ạ.  
 1 ^ | v • Reply • Share >

**Tiep Vu Huu** Mod → Thanh Nguyen • 9 months ago

Cảm ơn em đã quan tâm tới blog và dành những lời có cánh cho blog :).

Về việc viết bài, hiện tại mọi việc vẫn trong tầm kiểm soát của anh. Cảm ơn em đã gợi ý giúp đỡ.

Chúc em năm mới mạnh khỏe, hạnh phúc, và thành công.  
 ^ | v • Reply • Share >

**Thanh Nguyen** Tiep Vu Huu • 9 months ago

Cảm ơn a.  
 A ơi, cho e hỏi ý kiến của a chút nữa ạ.  
 Nếu học Machine learning mà biết mỗi ngôn ngữ Python thì liệu có đủ để áp dụng nó không ạ? E thì ngôn ngữ lập trình và ML là cũng tự học và e muốn áp dụng nó vào mảng e đang làm thì theo ý kiến của a a việc tự học ML và ngôn ngữ, liệu nó có đủ để sử dụng không ạ?  
 hay phải đi học 1 master chuyên ngành thì mới chắc chắn hiểu rõ và áp dụng được? tại e thấy ML là 1 ngành mới, và rất rộng thấy thú vị nhưng nhiều ng bảo cũng khó nứa. Đa phần những ng theo ngành này e đoán là những geeks hay những ng chuyên biệt học computer science. Vậy thì ko biết 1 người amateur đá chéo chân sang ngành này liệu có quá mạo hiểm không.  
 ^ | v • Reply • Share >

**Tiep Vu Huu** Mod → Thanh Nguyen • 9 months ago

Anh cũng chỉ biết 'mỗi' Python và Matlab, giờ chủ yếu dùng Python. Khi nào cần cái gì thì mình học cái đó.

Hầu hết là tự học với tài liệu hoặc các khoa trên mạng thời em. Các thay đổi master hay PhD cũng chưa chắc đã hay bằng các khóa học online. Tài liệu không sơ thiêу, chỉ sơ thiêу đam mê và ý chí thôi.

Không có cái gì thú vị mà không có khó khăn cả. Cũng không có thành công nào tự đến với mình cả. Có đam mê thì cần đâm lên, cứ bắt tay vào làm, giải quyết từng thứ một thì một ngày sẽ thành công.

[2 ^](#) [|](#) [v](#) [• Reply](#) [• Share](#) >

**Thanh Nguyen**  Tiep Vu Huu • 9 months ago

Dạ vâng e hiểu rồi ạ. Cảm ơn những góp ý của a :)

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#) >

**Kim Anh Huynh (Kim)** • 9 months ago

Chào anh, Blog và những link liên kết của anh hay quá. Em mới bắt đầu tìm hiểu về Machine Learning và cũng lạc lõi giữa nhiều thông tin dữ liệu. Nhưng những bài của anh rất hay và rõ ràng. Mong anh tiếp tục viết thêm nhiều bài để em cũng như những người mới bước đầu tìm đến Machine Learning được hiểu rõ hơn về nó. Chúc anh và gia đình năm mới tràn đầy sức khỏe.

[1 ^](#) [|](#) [v](#) [• Reply](#) [• Share](#) >

**Tiep Vu Huu** Mod  Kim Anh Huynh (Kim) • 9 months ago

Cảm ơn em đã quan tâm đến blog.

Anh vẫn luôn cố gắng hết mình để ra thêm nhiều bài nữa.

Chúc em năm mới hạnh phúc, mạnh khỏe, bình an và thành công.

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#) >

**Hoang-anh PHAM** • 10 months ago

Xin chào anh.,

Sau khi đọc blog của anh thì em thấy quá hay và hứng thú.

Em hiện chưa có câu hỏi nào.

Nhưng xin chúc anh và blog sẽ ngày càng phát triển trong thời gian tới. :D

P/s: Em đọc thấy anh có nói phần viết công thức toán và code blog mất nhiều time của anh. Liệu mọi người có thể tình nguyện giúp đỡ được gì cho anh không?

[1 ^](#) [|](#) [v](#) [• Reply](#) [• Share](#) >

**Tiep Vu Huu** Mod  Hoang-anh PHAM • 10 months ago

Chào em, những comment như thế này giúp anh có động lực viết tiếp rất nhiều.

Phản toán thì anh nghĩ không ai giúp được vì anh cần có ý tưởng, và cũng phải edit liên tục.

Phản khung cho blog thì anh nghĩ giờ cũng hòm hòm rồi. Khi nào có nhiều bài viết hơn thì anh sẽ thêm một cái site map. Giờ tạm thời thế này đã.

Anh cũng vừa làm thêm 1 tab Index để mọi người tiện tra cứu.

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#) >

**Hoang-anh PHAM**  Tiep Vu Huu • 10 months ago

Vâng anh. Một lần nữa xin chúc blog ngày càng phát triển và là một nguồn tham khảo bằng tiếng Việt chất lượng cho mọi người. :D

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#) >

**Tường Nguyễn** • 4 days ago

Chào anh, em là sv năm 3 ngành Tự động hóa, em mới vừa tìm hiểu xong python cơ bản và 3 bài đầu trong FundaML.com, em dự định đọc về ML và hiện tại em muốn tìm hiểu về xe tự lái, nhưng em không biết bắt đầu từ đâu, anh có thể gợi ý cho em được không ạ? Em xin cảm ơn

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#) >

**linh tran** • 20 days ago

Cám ơn a nhiều nhé!

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#) >

**Mai Hiếu Dũng** • a month ago

Xin cảm ơn anh!

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#) >

**Trung Thanh Nguyen** • 2 months ago

những bài viết của anh rất hay , dễ hiểu nhất từ trước tới nay so với những bài em đã từng học và đọc

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#) >

**tung ha** • 6 months ago

Hi,

This bài viết thực sự hay, I'm glad khi bạn share những info này cho everyone

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#) >

**Quốc Nhât** • 6 months ago

[Quốc Nhật](#) • 6 months ago

Chào anh.

Mọi người cho em hỏi C++ có dùng tốt trong lĩnh vực machine learning không ạ?

[^](#) [v](#) [• Reply](#) [Share](#) >

**Tiep Vu Huu** Mod → Quốc Nhật • 6 months ago

Theo anh biết thì các thư viện Machine Learning trên C++ không phong phú như trên python. Python vẫn là ngôn ngữ được sử dụng nhiều nhất.

[^](#) [v](#) [• Reply](#) [Share](#) >

ALSO ON TIEPVU

## Machine Learning cơ bản

11 comments • 8 months ago

**CTVR** — Em cũng thấy yêu cầu 64 bit trên win. Chắc em chạy với linux vậy.

P/s: Giờ em mới đọc xong loạt bài về convex trên file pdf. Hồi mới vào ...

## ebook Machine Learning cơ bản

26 comments • 3 months ago

**Xuân Xuân** — Anh ơi, cái email trường em cấp ý, em test nó không cho các mail ngoài gửi vào:"Hi. This is the qmail-send program at vnu.edu.vn.I'm ...

[✉ Subscribe](#) [Add Disqus to your site](#)[Add Disqus](#) [Privacy](#)

Total visits:

## Bài 2: Phân nhóm các thuật toán Machine Learning

General (/tags#General)

Dec 27, 2016

Có hai cách phổ biến phân nhóm các thuật toán Machine learning. Một là dựa trên phương thức học (learning style), hai là dựa trên chức năng (function) (của mỗi thuật toán).

**Trong trang này:**

- 1. Phân nhóm dựa trên phương thức học
  - Supervised Learning (Học có giám sát)
    - Classification (Phân loại)
    - Regression (Hồi quy)
  - Unsupervised Learning (Học không giám sát)
    - Clustering (phân nhóm)
    - Association
  - Semi-Supervised Learning (Học bán giám sát)
  - Reinforcement Learning (Học Củng Cố)
- 2. Phân nhóm dựa trên chức năng
  - Regression Algorithms
  - Classification Algorithms
  - Instance-based Algorithms
  - Regularization Algorithms
  - Bayesian Algorithms
  - Clustering Algorithms
  - Artificial Neural Network Algorithms
  - Dimensionality Reduction Algorithms
  - Ensemble Algorithms
- 3. Tài liệu tham khảo

### 1. Phân nhóm dựa trên phương thức học

Theo phương thức học, các thuật toán Machine Learning thường được chia làm 4 nhóm: Supervise learning, Unsupervised learning, Semi-supervised learning và Reinforcement learning. Có một số cách phân nhóm không có Semi-supervised learning hoặc Reinforcement learning.

#### Supervised Learning (Học có giám sát)

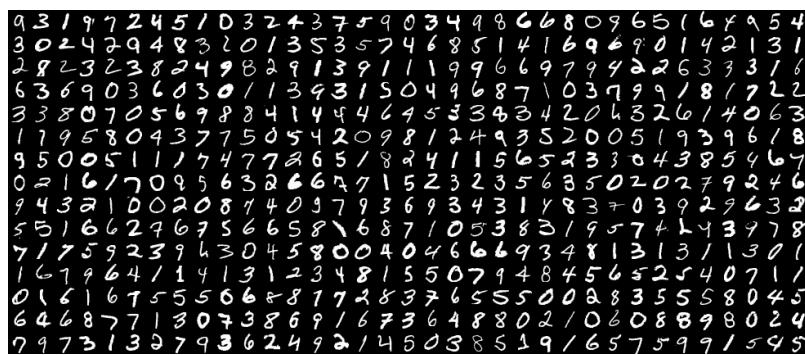
Supervised learning là thuật toán dự đoán đầu ra (outcome) của một dữ liệu mới (new input) dựa trên các cặp (*input, outcome*) đã biết từ trước. Cặp dữ liệu này còn được gọi là (*data, label*), tức (*dữ liệu, nhãn*). Supervised learning là nhóm phổ biến nhất trong các thuật toán Machine Learning.

Một cách toán học, Supervised learning là khi chúng ta có một tập hợp biến đầu vào  $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$  và một tập hợp nhãn tương ứng  $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N\}$ , trong đó  $\mathbf{x}_i, \mathbf{y}_i$  là các vector. Các cặp dữ liệu biết trước  $(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{X} \times \mathcal{Y}$  được gọi là tập *training data* (dữ liệu huấn luyện). Từ tập training data này, chúng ta cần tạo ra một hàm số ánh xạ mỗi phần tử từ tập  $\mathcal{X}$  sang một phần tử (xấp xỉ) tương ứng của tập  $\mathcal{Y}$ :

$$\mathbf{y}_i \approx f(\mathbf{x}_i), \quad \forall i = 1, 2, \dots, N$$

Mục đích là xấp xỉ hàm số  $f$  thật tốt để khi có một dữ liệu  $\mathbf{x}$  mới, chúng ta có thể tính được nhãn tương ứng của nó  $\mathbf{y} = f(\mathbf{x})$ .

**Ví dụ 1:** trong nhận dạng chữ viết tay, ta có ảnh của hàng nghìn ví dụ của mỗi chữ số được viết bởi nhiều người khác nhau. Chúng ta đưa các bức ảnh này vào trong một thuật toán và chỉ cho nó biết mỗi bức ảnh tương ứng với chữ số nào. Sau khi thuật toán tạo ra (sau khi *học*) một mô hình, tức một hàm số mà đầu vào là một bức ảnh và đầu ra là một chữ số, khi nhận được một bức ảnh mới mà mô hình **chưa nhìn thấy bao giờ**, nó sẽ dự đoán bức ảnh đó chứa chữ số nào.

MNIST (<http://yann.lecun.com/exdb/mnist/>): bộ cơ sở dữ liệu của chữ số viết tay.(Nguồn: Simple Neural Network implementation in Ruby) (<http://www.rubylab.io/2015/03/18/simple-neural-network-implementation-in-ruby/>)

Ví dụ này khá giống với cách học của con người khi còn nhỏ. Ta đưa bảng chữ cái cho một đứa trẻ và chỉ cho chúng đây là chữ A, đây là chữ B. Sau một vài lần được dạy thì trẻ có thể nhận biết được đâu là chữ A, đâu là chữ B trong một cuốn sách mà chúng chưa nhìn thấy bao giờ.

**Ví dụ 2:** Thuật toán dò các khuôn mặt trong một bức ảnh đã được phát triển từ rất lâu. Thời gian đầu, facebook sử dụng thuật toán này để chỉ ra các khuôn mặt trong một bức ảnh và yêu cầu người dùng *tag friends* - tức gán nhãn cho mỗi khuôn mặt. Số lượng cặp dữ liệu (*khuôn mặt, tên người*) càng lớn, độ chính xác ở những lần tự động *tag* tiếp theo sẽ càng lớn.

**Ví dụ 3:** Bản thân thuật toán dò tìm các khuôn mặt trong 1 bức ảnh cũng là một thuật toán Supervised learning với training data (dữ liệu học) là hàng ngàn cặp (*anh, mặt người*) và (*anh, không phải mặt người*) được đưa vào. Chú ý là dữ liệu này chỉ phân biệt *mặt người* và *không phải mặt người* mà không phân biệt khuôn mặt của những người khác nhau.

Thuật toán supervised learning còn được tiếp tục chia nhỏ ra thành hai loại chính:

### Classification (Phân loại)

Một bài toán được gọi là *classification* nếu các *label* của *input data* được chia thành một số hữu hạn nhóm. Ví dụ: Gmail xác định xem một email có phải là spam hay không; các hãng tin dụng xác định xem một khách hàng có khả năng thanh toán nợ hay không. Ba ví dụ phía trên được chia vào loại này.

### Regression (Hồi quy)

(tiếng Việt dịch là *Hồi quy*, tôi không thích cách dịch này vì bản thân không hiểu nó nghĩa là gì)

Nếu *label* không được chia thành các nhóm mà là một giá trị thực cụ thể. Ví dụ: một căn nhà rộng  $x \text{ m}^2$ , có  $y$  phòng ngủ và cách trung tâm thành phố  $z \text{ km}$  sẽ có giá là bao nhiêu?

Gần đây Microsoft có một ứng dụng dự đoán giới tính và tuổi dựa trên khuôn mặt (<https://how-old.net/>). Phần dự đoán giới tính có thể coi là thuật toán **Classification**, phần dự đoán tuổi có thể coi là thuật toán **Regression**. Chú ý rằng phần dự đoán tuổi cũng có thể coi là **Classification** nếu ta coi tuổi là một số nguyên dương không lớn hơn 150, chúng ta sẽ có 150 class (lớp) khác nhau.

### Unsupervised Learning (Học không giám sát)

Trong thuật toán này, chúng ta không biết được *outcome* hay *nhãn* mà chỉ có dữ liệu đầu vào. Thuật toán unsupervised learning sẽ dựa vào cấu trúc của dữ liệu để thực hiện một công việc nào đó, ví dụ như phân nhóm (clustering) hoặc giảm số chiều của dữ liệu (dimension reduction) để thuận tiện trong việc lưu trữ và tính toán.

Một cách toán học, Unsupervised learning là khi chúng ta chỉ có dữ liệu vào  $\mathcal{X}$  mà không biết  $\mathcal{Y}$  tương ứng.

Những thuật toán loại này được gọi là Unsupervised learning vì không giống như Supervised learning, chúng ta không biết câu trả lời chính xác cho mỗi dữ liệu đầu vào. Giống như khi ta học, không có thầy cô giáo nào chỉ cho ta biết đó là chữ A hay chữ B. Cụm *không giám sát* được đặt tên theo nghĩa này.

Các bài toán Unsupervised learning được tiếp tục chia nhỏ thành hai loại:

### Clustering (phân nhóm)

Một bài toán phân nhóm toàn bộ dữ liệu  $\mathcal{X}$  thành các nhóm nhỏ dựa trên sự liên quan giữa các dữ liệu trong mỗi nhóm. Ví dụ: phân nhóm khách hàng dựa trên hành vi mua hàng. Điều này cũng giống như việc ta đưa cho một đứa trẻ rất nhiều mảnh ghép với các hình thù và màu sắc khác nhau, ví dụ tam giác, vuông, tròn với màu xanh và đỏ, sau đó yêu cầu trẻ phân chung thành từng nhóm. Mặc dù không cho trẻ biết mảnh nào tương ứng với hình nào hoặc màu nào, nhiều khả năng chúng vẫn có thể phân loại các mảnh ghép theo màu hoặc hình dạng.

### Association

Là bài toán khi chúng ta muốn khám phá ra một quy luật dựa trên nhiều dữ liệu cho trước. Ví dụ: những khách hàng nam mua quần áo thường có xu hướng mua thêm đồng hồ hoặc thắt lưng; những khán giả xem phim Spider Man thường có xu hướng xem thêm phim Bat Man, dựa vào đó tạo ra một hệ thống gợi ý khách hàng (Recommendation System), thúc đẩy nhu cầu mua sắm.

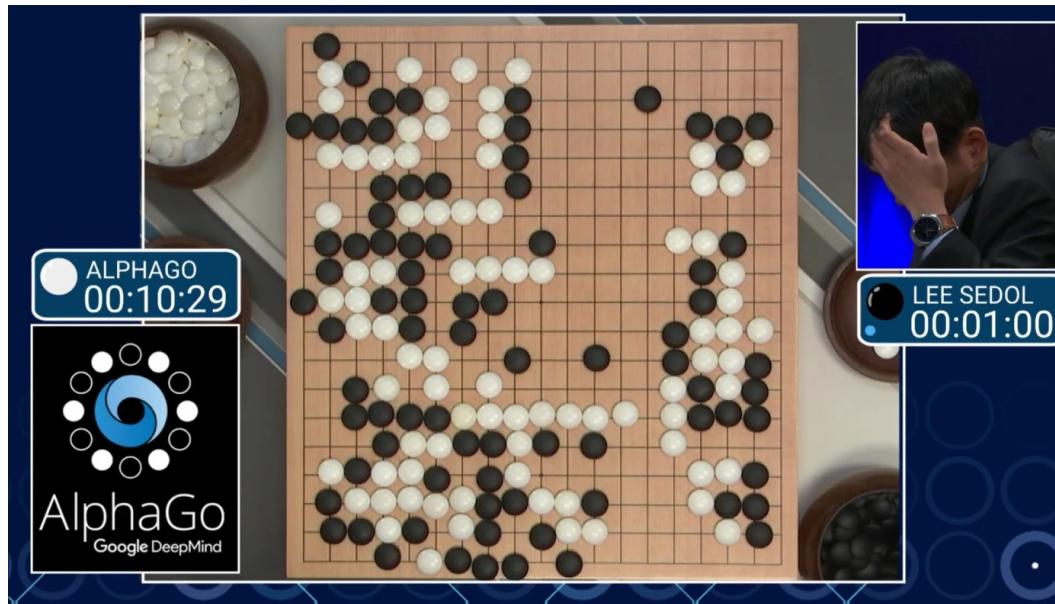
## Semi-Supervised Learning (Học bán giám sát)

Các bài toán khi chúng ta có một lượng lớn dữ liệu  $\mathcal{X}$  nhưng chỉ một phần trong chúng được gán nhãn được gọi là Semi-Supervised Learning. Những bài toán thuộc nhóm này nằm giữa hai nhóm được nêu bên trên.

Một ví dụ điển hình của nhóm này là chỉ có một phần ảnh hoặc văn bản được gán nhãn (ví dụ bức ảnh về người, động vật hoặc các văn bản khoa học, chính trị) và phần lớn các bức ảnh/văn bản khác chưa được gán nhãn được thu thập từ internet. Thực tế cho thấy rất nhiều các bài toán Machine Learning thuộc vào nhóm này vì việc thu thập dữ liệu có nhãn tốn rất nhiều thời gian và có chi phí cao. Rất nhiều loại dữ liệu thậm chí cần phải có chuyên gia mới gán nhãn được (ảnh y học chẳng hạn). Ngược lại, dữ liệu chưa có nhãn có thể được thu thập với chi phí thấp từ internet.

## Reinforcement Learning (Học Củng Cố)

Reinforcement learning là các bài toán giúp cho một hệ thống tự động xác định hành vi dựa trên hoàn cảnh để đạt được lợi ích cao nhất (maximizing the performance). Hiện tại, Reinforcement learning chủ yếu được áp dụng vào Lý Thuyết Trò Chơi (Game Theory), các thuật toán cần xác định nước đi tiếp theo để đạt được điểm số cao nhất.



AlphaGo chơi cờ vây với Lee Sedol. AlphaGo là một ví dụ của Reinforcement learning.

(Nguồn: AlphaGo AI Defeats Sedol Again, With 'Near Perfect Game' (<http://www.tomshardware.com/news/alphago-defeats-sedol-second-time,31377.html>)

**Ví dụ 1:** AlphaGo gần đây nổi tiếng với việc chơi cờ vây thắng cả con người (<https://gogameguru.com/tag/deepmind-alphago-lee-sedol/>). Cờ vây được xem là có độ phức tạp cực kỳ cao (<https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/>) với tổng số nước đi là  $xấp xỉ 10^{761}$ , so với cờ vua là  $10^{120}$  và tổng số nguyên tử trong toàn vũ trụ là khoảng  $10^{80}$ !! Vì vậy, thuật toán phải chọn ra 1 nước đi tối ưu trong số hàng nghìn tỉ lựa chọn, và tất nhiên, không thể áp dụng thuật toán tương tự như IBM Deep Blue ([https://en.wikipedia.org/wiki/Deep\\_Blue\\_\(chess\\_computer\)](https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer))) (IBM Deep Blue đã thắng con người trong môn cờ vua 20 năm trước). Về cơ bản, AlphaGo bao gồm các thuật toán thuộc cả Supervised learning và Reinforcement learning. Trong phần Supervised learning, dữ liệu từ các ván cờ do con người chơi với nhau được đưa vào để huấn luyện. Tuy nhiên, mục đích cuối cùng của AlphaGo không phải là chơi như con người mà phải thậm chí thắng cả con người. Vì vậy, sau khi học xong các ván cờ của con người, AlphaGo tự chơi với chính nó với hàng triệu ván chơi để tìm ra các nước đi mới tối ưu hơn. Thuật toán trong phần tự chơi này được xếp vào loại Reinforcement learning. (Xem thêm tại Google DeepMind's AlphaGo: How it works (<https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/>)).

**Ví dụ 2:** Huấn luyện cho máy tính chơi game Mario (<https://www.youtube.com/watch?v=qv6UVVOQ0F44>). Đây là một chương trình thú vị dạy máy tính chơi game Mario. Game này đơn giản hơn cờ vây vì tại một thời điểm, người chơi chỉ phải bấm một số lượng nhỏ các nút (di chuyển, nhảy, bắn đạn) hoặc không cần bấm nút nào. Đồng thời, phản ứng của máy cũng đơn giản hơn và lặp lại ở mỗi lần chơi (tại thời điểm cụ thể sẽ xuất hiện một chướng ngại vật cố định ở một vị trí cố định). Đầu vào của thuật toán là sơ đồ của màn hình tại thời điểm hiện tại, nhiệm vụ của thuật toán là với đầu vào đó, tổ hợp phím nào nên được bấm. Việc huấn luyện này được dựa trên điểm số cho việc di chuyển được bao xa trong thời gian bao lâu trong game, càng xa và càng nhanh thì được điểm thường càng cao (điểm thường này không phải là điểm của trò chơi mà là điểm do chính người lập trình tạo ra). Thông qua huấn luyện, thuật toán sẽ tìm ra một cách tối ưu để tối đa số điểm trên, qua đó đạt được mục đích cuối cùng là cứu công chúa.

### Marl/O - Machine Learning for Video Games



Huấn luyện cho máy tính chơi game Mario

## 2. Phân nhóm dựa trên chức năng

Có một cách phân nhóm thứ hai dựa trên chức năng của các thuật toán. Trong phần này, tôi xin chỉ liệt kê các thuật toán. Thông tin cụ thể sẽ được trình bày trong các bài viết khác tại blog này. Trong quá trình viết, tôi có thể sẽ thêm bớt một số thuật toán.

### Regression Algorithms

1. Linear Regression ([/2016/12/28/linearregression/](#))
2. Logistic Regression ([/2017/01/27/logisticregression/#sigmoid-function](#))
3. Stepwise Regression

### Classification Algorithms

1. Linear Classifier
2. Support Vector Machine (SVM) (<https://machinelearningcoban.com/2017/04/09/sm/>)
3. Kernel SVM (<https://machinelearningcoban.com/2017/04/22/kernelsmv/>)
4. Sparse Representation-based classification (SRC)

### Instance-based Algorithms

1. k-Nearest Neighbor (kNN) ([/2017/01/08/knn/](#))
2. Learning Vector Quantization (LVQ)

### Regularization Algorithms

1. Ridge Regression
2. Least Absolute Shrinkage and Selection Operator (LASSO)
3. Least-Angle Regression (LARS)

### Bayesian Algorithms

1. Naive Bayes
2. Gaussian Naive Bayes

### Clustering Algorithms

1. k-Means clustering ([/2017/01/01/kmeans/](#))
2. k-Medians
3. Expectation Maximization (EM)

### Artificial Neural Network Algorithms

1. Perceptron ([/2017/01/21/perceptron/](#))
2. Softmax Regression ([/2017/02/17/softmax/](#))
3. Multi-layer Perceptron ([/2017/02/24/mlp/](#))

#### 4. Back-Propagation (/2017/02/24/mlp/#-backpropagation)

## Dimensionality Reduction Algorithms

1. Principal Component Analysis (PCA) (<https://machinelearningcoban.com/2017/06/15/pca/>)
2. Linear Discriminant Analysis (LDA) (<https://machinelearningcoban.com/2017/06/30/lda/>)

## Ensemble Algorithms

1. Boosting
2. AdaBoost
3. Random Forest

Và còn rất nhiều các thuật toán khác.

## 3. Tài liệu tham khảo

1. A Tour of Machine Learning Algorithms (<http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/>)
2. Điểm qua các thuật toán Machine Learning hiện đại (<https://ongxuanhong.wordpress.com/2015/10/22/diem-quacac-thuật-toán-machine-learning-hien-dai/>)

Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (<https://www.facebook.com/groups/257768141347267/>) để nhận được câu trả lời sớm hơn.

Bạn đọc có thể ủng hộ blog qua 'Buy me a coffee' (/buymeacoffee/) ở góc bên trái của blog.

Tôi đang trong quá trình viết cuốn sách 'Machine Learning cơ bản I', các bạn có thể đặt trước tại đây (/ebook/). Cảm ơn bạn.

« Bài 1: Giới thiệu về Machine Learning (/2016/12/26/introduce/)

Bài 3: Linear Regression » (/2016/12/28/linearregression/)

24 Comments tiepvu

 Login ▾

 Recommend 4  Share

Sort by Best ▾



Join the discussion...

LOG IN WITH OR SIGN UP WITH DISQUS [?](#)

Name

 **Dương Đỗ** • 15 days ago  
@Tiep Vu Huu anh viết hay, dễ hiểu. Nhưng em nghĩ mình nên thêm vào blog chức năng highlight (tương tự Medium) hoặc mang hẳn blog lên Medium để mọi người thuận tiện hơn trong việc đọc...

1 ^ | v • Reply • Share >

 **Khánh Hòa Nguyễn** • 16 days ago  
Hiện tại mình đang đảm nhiệm dự án về dự đoán dữ liệu lõi khi process 1 sản phẩm. Ví dụ để ra sản phẩm A, máy cần chạy recipe với power = 100W trong 1h. Trong 1h đó, hệ thống sẽ thu thập lại giá trị power biến đổi ntn (ví dụ 1s thu thập 1 lần). Nhiệm vụ của mình là thu thập dữ liệu này cho tất cả các lần chạy đúng, rồi dùng machine learning để phát hiện và thông báo cho người dùng khi phát hiện dữ liệu mới khác thường so với những lần chạy đúng này. Theo bạn thì mình nên sử dụng thuật toán nào cho dạng bài toán này? Đầu tiên mình nghĩ dùng loại Supervised là phù hợp, tuy nhiên người dùng ko chỉ chạy mỗi recipe với power = 100W trong 1h, hệ thống sẽ chạy nhiều loại recipe với power = 100/200/300... trong 15p/30p/1h/2h.... Nếu dùng Supervised thì mình phải handle nhiều quá, mà dùng Un-supervised hay Semi thì mình lại ko biết chọn thuật toán nào cho phù hợp. Mong bạn có thể chia sẻ kinh nghiệm

1 ^ | v • Reply • Share >

 **Tiep Vu Huu** Mod → Khánh Hòa Nguyễn • 16 days ago  
Bạn hỏi trên Forum nhé. Trên đó có nhiều người làm thực tế hơn, có thể họ biết câu trả lời.

^ | v • Reply • Share >

 **Khánh Hòa Nguyễn** → Tiep Vu Huu • 16 days ago  
Thanks bạn

^ | v • Reply • Share >



Rất dễ hiểu, cảm ơn bạn rất nhiều nhé!

[1 ^](#) [v](#) • Reply • Share >



Tường Nguyễn • 2 days ago

Chào anh, em muốn học ML và DL chuyên về xe tự lái thì nên theo hướng như thế nào ạ?

[^](#) [v](#) • Reply • Share >



Van Thang Cao • 3 days ago

Chào admin,

Các bài viết của admin về ML rất hay và dễ hiểu (mình mới đọc được 2 bài. hihi)

Mình đang tìm hiểu về Random Forest (RF) nhưng toàn tài liệu tiếng Anh, Admin có thể viết một bài hướng dẫn cách phân lớp với RF (với ví dụ cụ thể) để mọi người học hỏi được không?

Chân thành cảm ơn Admin.

[^](#) [v](#) • Reply • Share >



Khánh Hòa Nguyễn • 17 days ago

Cho mình hỏi là đối với loại Semi-Supervised Learning thì mình áp dụng thuật toán nào?

[^](#) [v](#) • Reply • Share >



NGUYỄN TRỌNG TOÀN • 18 days ago

thanks men ^^

[^](#) [v](#) • Reply • Share >



Võ Văn Huấn • 2 months ago

Anh cho em hỏi, tại sao dùng thuật ngữ: cặp(input, outcome) mà không phải là cặp (input, output) vậy anh?

[^](#) [v](#) • Reply • Share >



Tiep Vu Huu Mod → Võ Văn Huấn • 2 months ago

Không khác nhau nhiều đâu bạn. Có tài liệu dùng outcome, có tài liệu dùng output.

[^](#) [v](#) • Reply • Share >



anna → Võ Văn Huấn • 2 months ago

cùng thắc mắc

[^](#) [v](#) • Reply • Share >



Hoàng Tường • 5 months ago

anh cho em hỏi là laber trong này có nghĩa là gì ạ em hơi mơ hồ chỗ này ạ ^^

[^](#) [v](#) • Reply • Share >



Tiep Vu Huu Mod → Hoàng Tường • 4 months ago

Khi bạn làm bài toán phân loại chữ số viết tay thì label là "tên" của từng class, là 0, 1, 2, ..., 9.

Khi bạn làm bài toán phân loại email thường hay email rác, label là "thường", "rác".

Labels là tên của từng class.

[1 ^](#) [v](#) • Reply • Share >



Huyen Trinh • 5 months ago

Có bạn nào có tài liệu học Decision Tree clasification ko ạ. Cho mình xin dc ko?

[^](#) [v](#) • Reply • Share >



Nguyễn Văn Tiêm • 5 months ago

Mình đang nghiên cứu về Machine Learning Blocks nhưng mà đang rất mơ hồ. Liệu bạn có thể mở một chủ đề về nó để mọi người cho ý kiến được không ạ?

[^](#) [v](#) • Reply • Share >



Nam Vu • 8 months ago

Logistic Regression là cho Classification mà anh nhỉ?

[^](#) [v](#) • Reply • Share >



Tiep Vu Huu Mod → Nam Vu • 8 months ago

Cái ranh giới đó rất mong manh. Nó vẫn được giữ tên là Regression nên anh xem nó vào đó.

[^](#) [v](#) • Reply • Share >



Minh Trọc → Tiep Vu Huu • 5 months ago

Gs. Andrew Ng. cũng xếp nó vào Classification mà; ông nói nó có tên regression là do vấn đề lịch sử

[^](#) [v](#) • Reply • Share >



Dung • 10 months ago

Anh có thể viết một bài về thuật toán Expectation Maximisation (EM) dc ko ạ. EM đang học mà ko hiểu về nó mấy.

^ | v • Reply • Share ›



Tiep Vu Huu Mod → Dung • 10 months ago

Anh sẽ viết, nhưng anh cần viết bài k-means clustering trước đã.

1 ^ | v • Reply • Share ›



Phucnh • 10 months ago

Các công thức toán học bị lỗi math processing error, anh có thể xem lại không ạ.

^ | v • Reply • Share ›



Tiep Vu Huu Mod → Phucnh • 10 months ago

Chào em,

Em vào bằng trình duyệt nào về hệ điều hành gì? Em đã thử trên các trình duyệt khác chưa?

Anh kiểm tra trên Chrome, Firefox, Opera trên Ubuntu và Chrome trên Android thấy công thức toán không bị lỗi.

^ | v • Reply • Share ›



Phucnh → Tiep Vu Huu • 10 months ago

Dạ chào anh,  
 khi nãy em vào bằng safari ios thì gặp lỗi đó, giờ thì xem bình thường rồi ạ. Có lẽ là lỗi tạm thời.  
 Cảm ơn anh.

1 ^ | v • Reply • Share ›

## ALSO ON TIEPVU

**Machine Learning cơ bản**

11 comments • 8 months ago

CTVR — Em cũng thấy yêu cầu 64 bit trên win. Chắc em chạy với linux vậy.

P/s: Giờ em mới đọc xong loạt bài về convex trên file pdf. Hồi mới vào ...

**ebook Machine Learning cơ bản**

26 comments • 3 months ago

Xuân Xuân — Anh ơi, cái email trường em cấp ý, em test nó không cho các mail ngoài gửi vào:"Hi. This is the qmail-send program at vnu.edu.vn.I'm ...

[✉ Subscribe](#)  [ⓘ Add Disqus to your site](#) [Add](#) [🔒 Privacy](#)

Total visits:

## Bài 3: Linear Regression

Linear-models (/tags#Linear-models)    Regression (/tags#Regression)    Supervised-learning (/tags#Supervised-learning)

Dec 28, 2016

Trong bài này, tôi sẽ giới thiệu một trong những thuật toán cơ bản nhất (và đơn giản nhất) của Machine Learning. Đây là một thuật toán *Supervised learning* có tên **Linear Regression** (Hồi Quy Tuyến Tính). Bài toán này đôi khi được gọi là *Linear Fitting* (trong thống kê) hoặc *Linear Least Square*.

**Trong trang này:**

- 1. Giới thiệu
- 2. Phân tích toán học
  - 2.1. Dạng của Linear Regression
  - 2.2. Sai số dự đoán
  - 2.3. Hàm mất mát
  - 2.4. Nghiệm cho bài toán Linear Regression
- 3. Ví dụ trên Python
  - 3.1. Bài toán
  - 3.2. Hiển thị dữ liệu trên đồ thị
  - 3.3. Nghiệm theo công thức
  - 3.4. Nghiệm theo thư viện scikit-learn
- 4. Thảo luận
  - 4.1. Các bài toán có thể giải bằng Linear Regression
  - 4.2. Hạn chế của Linear Regression
  - 4.3. Các phương pháp tối ưu
- 5. Tài liệu tham khảo

### 1. Giới thiệu

Quay lại ví dụ đơn giản được nêu trong bài trước (/2016/12/27/categories/#regression-hoi-quy): một căn nhà rộng  $x_1 \text{ m}^2$ , có  $x_2$  phòng ngủ và cách trung tâm thành phố  $x_3 \text{ km}$  có giá là bao nhiêu. Giả sử chúng ta đã có số liệu thống kê từ 1000 căn nhà trong thành phố đó, liệu rằng khi có một căn nhà mới với các thông số về diện tích, số phòng ngủ và khoảng cách tới trung tâm, chúng ta có thể dự đoán được giá của căn nhà đó không? Nếu có thì hàm dự đoán  $y = f(\mathbf{x})$  sẽ có dạng như thế nào. Ở đây  $\mathbf{x} = [x_1, x_2, x_3]$  là một vector hàng chứa thông tin *input*,  $y$  là một số vô hướng (scalar) biểu diễn *output* (tức giá của căn nhà trong ví dụ này).

**Lưu ý về ký hiệu toán học:** trong các bài viết của tôi, các số vô hướng được biểu diễn bởi các chữ cái viết ở dạng không in đậm, có thể viết hoa, ví dụ  $x_1, N, y, k$ . Các vector được biểu diễn bằng các chữ cái thường in đậm, ví dụ  $\mathbf{y}, \mathbf{x}_1$ . Các ma trận được biểu diễn bởi các chữ viết hoa in đậm, ví dụ  $\mathbf{X}, \mathbf{Y}, \mathbf{W}$ .

Một cách đơn giản nhất, chúng ta có thể thấy rằng: i) diện tích nhà càng lớn thì giá nhà càng cao; ii) số lượng phòng ngủ càng lớn thì giá nhà càng cao; iii) càng xa trung tâm thì giá nhà càng giảm. Một hàm số đơn giản nhất có thể mô tả mối quan hệ giữa giá nhà và 3 đại lượng đầu vào là:

$$\begin{aligned} y &\approx f(\mathbf{x}) = \hat{y} \\ f(\mathbf{x}) &= w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0 \quad (1) \end{aligned}$$

trong đó,  $w_1, w_2, w_3, w_0$  là các hằng số,  $w_0$  còn được gọi là bias. Mỗi quan hệ  $y \approx f(\mathbf{x})$  bên trên là một mối quan hệ tuyến tính (linear). Bài toán chúng ta đang làm là một bài toán thuộc loại regression. Bài toán đi tìm các hệ số tối ưu  $\{w_1, w_2, w_3, w_0\}$  chính vì vậy được gọi là bài toán Linear Regression.

**Chú ý 1:**  $y$  là giá trị thực của *outcome* (dựa trên số liệu thống kê chúng ta có trong tập *training data*), trong khi  $\hat{y}$  là giá trị mà mô hình Linear Regression dự đoán được. Nhìn chung,  $y$  và  $\hat{y}$  là hai giá trị khác nhau do có sai số mô hình, tuy nhiên, chúng ta mong muốn rằng sự khác nhau này rất nhỏ.

**Chú ý 2:** *Linear* hay *tuyến tính* hiểu một cách đơn giản là *thẳng, phẳng*. Trong không gian hai chiều, một hàm số được gọi là *tuyến tính* nếu đồ thị của nó có dạng một *đường thẳng*. Trong không gian ba chiều, một hàm số được gọi là *tuyến tính* nếu đồ thị của nó có dạng một *mặt phẳng*. Trong không gian nhiều hơn 3 chiều, khái niệm *mặt phẳng* không còn phù hợp nữa, thay vào đó, một khái niệm khác ra đời được gọi là *siêu mặt phẳng (hyperplane)*. Các hàm số tuyến tính là các hàm đơn giản nhất, vì chúng thuận tiện trong việc hình dung và tính toán. Chúng ta sẽ được thấy trong các bài viết sau, *tuyến tính* rất quan trọng và hữu ích trong các bài toán Machine Learning. Kinh nghiệm cá nhân tôi cho thấy, trước khi hiểu được các thuật toán *phi tuyến tính* (non-linear, không phẳng), chúng ta cần nắm vững các kỹ thuật cho các mô hình *tuyến tính*.

### 2. Phân tích toán học

## 2.1. Dạng của Linear Regression

Trong phương trình (1) phía trên, nếu chúng ta đặt  $\mathbf{w} = [w_0, w_1, w_2, w_3]^T$  là vector (cột) hệ số cần phải tối ưu và  $\bar{\mathbf{x}} = [1, x_1, x_2, x_3]$  (đọc là *x bar* trong tiếng Anh) là vector (hàng) dữ liệu đầu vào *mở rộng*. Số 1 ở đầu được thêm vào để phép tính đơn giản hơn và thuận tiện cho việc tính toán. Khi đó, phương trình (1) có thể được viết lại dưới dạng:

$$y \approx \bar{\mathbf{x}}\mathbf{w} = \hat{y}$$

Chú ý rằng  $\bar{\mathbf{x}}$  là một vector hàng. (Xem thêm về ký hiệu vector hàng và cột tại đây (/math/#luu-y-ve-ky-hieu))

## 2.2. Sai số dự đoán

Chúng ta mong muốn rằng sự sai khác  $e$  giữa giá trị thực  $y$  và giá trị dự đoán  $\hat{y}$  (đọc là *y hat* trong tiếng Anh) là nhỏ nhất. Nói cách khác, chúng ta muốn giá trị sau đây càng nhỏ càng tốt:

$$\frac{1}{2}e^2 = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(y - \bar{\mathbf{x}}\mathbf{w})^2$$

trong đó hệ số  $\frac{1}{2}$  (*lại*) là để thuận tiện cho việc tính toán (khi tính đạo hàm thì số  $\frac{1}{2}$  sẽ bị triệt tiêu). Chúng ta cần  $e^2$  vì  $e = y - \hat{y}$  có thể là một số âm, việc nói  $e$  nhỏ nhất sẽ không đúng vì khi  $e = -\infty$  là rất nhỏ nhưng sự sai lệch là rất lớn. Bạn đọc có thể tự đặt câu hỏi: **tại sao không dùng trị tuyệt đối  $|e|$  mà lại dùng bình phương  $e^2$  ở đây?** Câu trả lời sẽ có ở phần sau.

## 2.3. Hàm mất mát

Điều tương tự xảy ra với tất cả các cặp (*input, outcome*)  $(\mathbf{x}_i, y_i)$ ,  $i = 1, 2, \dots, N$ , với  $N$  là số lượng dữ liệu quan sát được. Điều chúng ta muốn, tổng sai số là nhỏ nhất, tương đương với việc tìm  $\mathbf{w}$  để hàm số sau đạt giá trị nhỏ nhất:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N (y_i - \bar{\mathbf{x}}_i \mathbf{w})^2 \quad (2)$$

Hàm số  $\mathcal{L}(\mathbf{w})$  được gọi là **hàm mất mát** (loss function) của bài toán Linear Regression. Chúng ta luôn mong muốn rằng sự mất mát (sai số) là nhỏ nhất, điều đó đồng nghĩa với việc tìm vector hệ số  $\mathbf{w}$  sao cho giá trị của hàm mất mát này càng nhỏ càng tốt. Giá trị của  $\mathbf{w}$  làm cho hàm mất mát đạt giá trị nhỏ nhất được gọi là **điểm tối ưu** (optimal point), ký hiệu:

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

Trước khi đi tìm lời giải, chúng ta đơn giản hóa phép toán trong phương trình hàm mất mát (2). Đặt  $\mathbf{y} = [y_1; y_2; \dots; y_N]$  là một vector cột chứa tất cả các *output* của *training data*;  $\bar{\mathbf{X}} = [\bar{\mathbf{x}}_1; \bar{\mathbf{x}}_2; \dots; \bar{\mathbf{x}}_N]$  là ma trận dữ liệu đầu vào (mở rộng) mà mỗi hàng của nó là một điểm dữ liệu. Khi đó hàm số mất mát  $\mathcal{L}(\mathbf{w})$  được viết dưới dạng ma trận đơn giản hơn:

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &= \frac{1}{2} \sum_{i=1}^N (y_i - \bar{\mathbf{x}}_i \mathbf{w})^2 \\ &= \frac{1}{2} \|\mathbf{y} - \bar{\mathbf{X}}\mathbf{w}\|_2^2 \end{aligned} \quad (3)$$

với  $\|\mathbf{z}\|_2$  là Euclidean norm (chuẩn Euclid, hay khoảng cách Euclid), nói cách khác  $\|\mathbf{z}\|_2^2$  là tổng của bình phương mỗi phần tử của vector  $\mathbf{z}$ . Tới đây, ta đã có một dạng đơn giản của hàm mất mát được viết như phương trình (3).

## 2.4. Nghiệm cho bài toán Linear Regression

**Cách phổ biến nhất để tìm nghiệm cho một bài toán tối ưu (chúng ta đã biết từ khi học cấp 3) là giải phương trình đạo hàm (gradient) bằng 0!** Tất nhiên đó là khi việc tính đạo hàm và việc giải phương trình đạo hàm bằng 0 không quá phức tạp. Thật may mắn, với các mô hình tuyến tính, hai việc này là khả thi.

Đạo hàm theo  $\mathbf{w}$  của hàm mất mát là:

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \bar{\mathbf{X}}^T (\bar{\mathbf{X}}\mathbf{w} - \mathbf{y})$$

Các bạn có thể tham khảo bảng đạo hàm theo vector hoặc ma trận của một hàm số trong mục D.2 của tài liệu này (<https://ccrma.stanford.edu/~dattorro/matrixcalc.pdf>). **Đến đây tôi xin quay lại câu hỏi ở phần Sai số dự đoán phía trên về việc tại sao không dùng trị tuyệt đối mà lại dùng bình phương.** Câu trả lời là **hàm bình phương có đạo hàm tại mọi nơi, trong khi hàm trị tuyệt đối thì không (đạo hàm không xác định tại 0)**.

Phương trình đạo hàm bằng 0 tương đương với:

$$\bar{\mathbf{X}}^T \bar{\mathbf{X}}\mathbf{w} = \bar{\mathbf{X}}^T \mathbf{y} \triangleq \mathbf{b} \quad (4)$$

(ký hiệu  $\bar{\mathbf{X}}^T \mathbf{y} \triangleq \mathbf{b}$  nghĩa là đặt  $\bar{\mathbf{X}}^T \mathbf{y}$  bằng  $\mathbf{b}$  ).

Nếu ma trận vuông  $\mathbf{A} \triangleq \bar{\mathbf{X}}^T \bar{\mathbf{X}}$  khả nghịch (non-singular hay invertible) thì phương trình (4) có nghiệm duy nhất:  $\mathbf{w} = \mathbf{A}^{-1} \mathbf{b}$ .

Vậy nếu ma trận  $\mathbf{A}$  không khả nghịch (có định thức bằng 0) thì sao? Nếu các bạn vẫn nhớ các kiến thức về hệ phương trình tuyến tính, trong trường hợp này thì hoặc phương trình (4) vô nghiệm, hoặc là nó có vô số nghiệm. Khi đó, chúng ta sử dụng khái niệm *giả nghịch đảo* ([https://vi.wikipedia.org/wiki/Giả\\_nghịch\\_đảo\\_Moore-Penrose](https://vi.wikipedia.org/wiki/Giả_nghịch_đảo_Moore-Penrose))  $\mathbf{A}^\dagger$  (đọc là *A dagger* trong tiếng Anh). (*Giả nghịch đảo (pseudo inverse)* là trường hợp tổng quát của *nghịch đảo* khi ma trận không khả nghịch hoặc thậm chí không vuông. Trong khuôn khổ bài viết này, tôi xin phép được lược bỏ phần này, nếu các bạn thực sự quan tâm, tôi sẽ viết một bài khác chỉ nói về giả nghịch đảo. Xem thêm: *Least Squares, Pseudo-Inverses, PCA & SVD* (<http://www.sci.utah.edu/~gerig/CS6640-F2012/Materials/pseudoinverse-cis61009sl10.pdf>).

Với khái niệm giả nghịch đảo, điểm tối ưu của bài toán Linear Regression có dạng:

$$\mathbf{w} = \mathbf{A}^\dagger \mathbf{b} = (\bar{\mathbf{X}}^T \bar{\mathbf{X}})^\dagger \bar{\mathbf{X}}^T \mathbf{y} \quad (5)$$

### 3. Ví dụ trên Python

#### 3.1. Bài toán

Trong phần này, tôi sẽ chọn một ví dụ đơn giản về việc giải bài toán Linear Regression trong Python. Tôi cũng sẽ so sánh nghiệm của bài toán khi giải theo phương trình (5) và nghiệm tìm được khi dùng thư viện scikit-learn (<http://scikit-learn.org/stable/>) của Python. (Đây là thư viện Machine Learning được sử dụng rộng rãi trong Python). Trong ví dụ này, dữ liệu đầu vào chỉ có 1 giá trị (1 chiều) để thuận tiện cho việc minh họa trong mặt phẳng.

Chúng ta có 1 bảng dữ liệu về chiều cao và cân nặng của 15 người như dưới đây:

Chiều cao (cm)	Cân nặng (kg)	Chiều cao (cm)	Cân nặng (kg)
147	49	168	60
150	50	170	72
153	51	173	63
155	52	175	64
158	54	178	66
160	56	180	67
163	58	183	68
165	59		

Bài toán đặt ra là: liệu có thể dự đoán cân nặng của một người dựa vào chiều cao của họ không? (Trên thực tế, tất nhiên là không, vì cân nặng còn phụ thuộc vào nhiều yếu tố khác nữa, thể tích chẳng hạn). Vì blog này nói về các thuật toán Machine Learning đơn giản nên tôi sẽ giả sử rằng chúng ta có thể dự đoán được.

Chúng ta có thể thấy là cân nặng sẽ tỉ lệ thuận với chiều cao (càng cao càng nặng), nên có thể sử dụng Linear Regression model cho việc dự đoán này. Để kiểm tra độ chính xác của model tìm được, chúng ta sẽ giữ lại cột 155 và 160 cm để kiểm thử, các cột còn lại được sử dụng để huấn luyện (train) model.

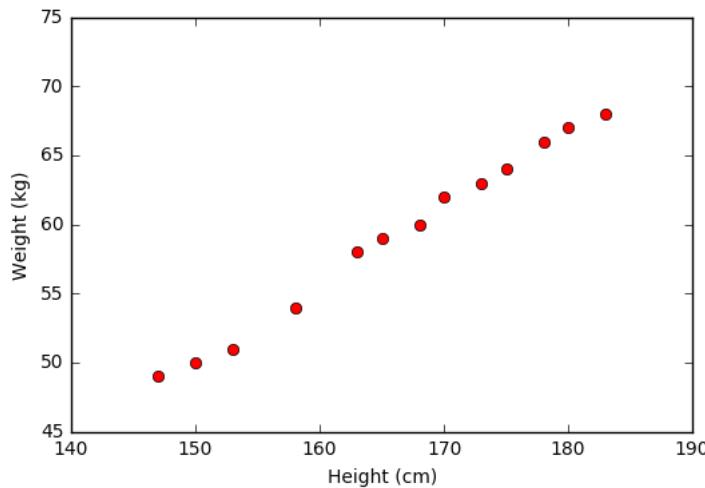
#### 3.2. Hiển thị dữ liệu trên đồ thị

Trước tiên, chúng ta cần có hai thư viện numpy (<http://www.numpy.org/>) cho đại số tuyến tính và matplotlib (<http://matplotlib.org/>) cho việc vẽ hình.

```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals
import numpy as np
import matplotlib.pyplot as plt
```

Tiếp theo, chúng ta khai báo và biểu diễn dữ liệu trên một đồ thị.

```
# height (cm)
X = np.array([[147, 150, 153, 158, 163, 165, 168, 170, 173, 175, 178, 180, 183]]).T
# weight (kg)
y = np.array([[49, 50, 51, 54, 58, 59, 60, 62, 63, 64, 66, 67, 68]]).T
# Visualize data
plt.plot(X, y, 'ro')
plt.axis([140, 190, 45, 75])
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.show()
```



Từ đồ thị này ta thấy rằng dữ liệu được sắp xếp gần như theo 1 đường thẳng, vậy mô hình Linear Regression nhiều khả năng sẽ cho kết quả tốt:

$$(cân nặng) = w_1 \cdot (\chiều cao) + w_0$$

### 3.3. Nghiệm theo công thức

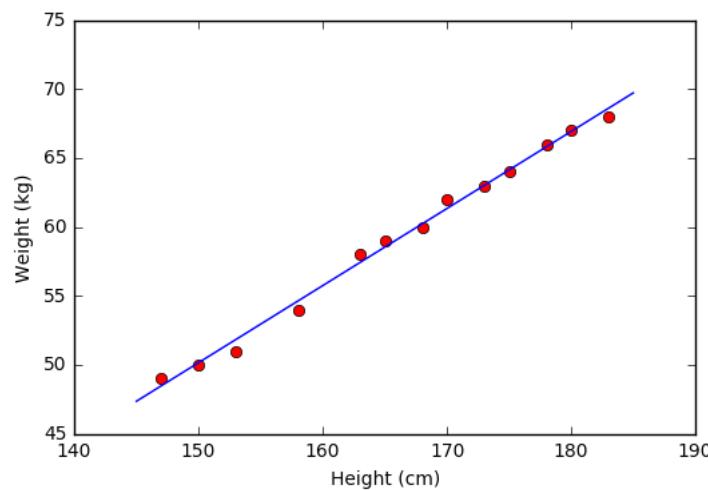
Tiếp theo, chúng ta sẽ tính toán các hệ số  $w_1$  và  $w_0$  dựa vào công thức (5). Chú ý: giả nghịch đảo của một ma trận A trong Python sẽ được tính bằng `numpy.linalg.pinv(A)`, `pinv` là từ viết tắt của *pseudo inverse*.

```
# Building Xbar
one = np.ones((X.shape[0], 1))
Xbar = np.concatenate((one, X), axis = 1)

# Calculating weights of the fitting line
A = np.dot(Xbar.T, Xbar)
b = np.dot(Xbar.T, y)
w = np.dot(np.linalg.pinv(A), b)
print('w = ', w)
# Preparing the fitting line
w_0 = w[0][0]
w_1 = w[1][0]
x0 = np.linspace(145, 185, 2)
y0 = w_0 + w_1*x0

# Drawing the fitting line
plt.plot(X.T, y.T, 'ro')      # data
plt.plot(x0, y0)              # the fitting line
plt.axis([140, 190, 45, 75])
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.show()
```

```
w = [[-33.73541021]
 [ 0.55920496]]
```



Từ đồ thị bên trên ta thấy rằng các điểm dữ liệu màu đỏ nằm khá gần đường thẳng dự đoán màu xanh. Vậy mô hình Linear Regression hoạt động tốt với tập dữ liệu *training*. Jetzt, chúng ta sử dụng mô hình này để dự đoán cân nặng của hai người có chiều cao 155 và 160 cm mà chúng ta đã không dùng khi tính toán nghiệm.

```
y1 = w_1*155 + w_0
y2 = w_1*160 + w_0

print( u'Predict weight of person with height 155 cm: %.2f (kg), real number: %d (kg)' % (y1) )
print( u'Predict weight of person with height 160 cm: %.2f (kg), real number: %d (kg)' % (y2) )
```

```
Predict weight of person with height 155 cm: 52.94 (kg), real number: 52 (kg)
Predict weight of person with height 160 cm: 55.74 (kg), real number: 56 (kg)
```

Chúng ta thấy rằng kết quả dự đoán khá gần với số liệu thực tế.

### 3.4. Nghiệm theo thư viện scikit-learn

Tiếp theo, chúng ta sẽ sử dụng thư viện scikit-learn của Python để tìm nghiệm.

```
from sklearn import datasets, linear_model

# fit the model by Linear Regression
regr = linear_model.LinearRegression(fit_intercept=False) # fit_intercept = False for calculating the bias
regr.fit(Xbar, y)

# Compare two results
print('Solution found by scikit-learn : ', regr.coef_)
print('Solution found by (5): ', w.T)
```

```
Solution found by scikit-learn : [[ -33.73541021  0.55920496]]
Solution found by (5): [[ -33.73541021  0.55920496]]
```

Chúng ta thấy rằng hai kết quả thu được như nhau! (Nghĩa là tôi đã không mắc lỗi nào trong cách tìm nghiệm ở phần trên)

Source code Jupyter Notebook cho bài này. (<https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/LR/LR.ipynb>)

## 4. Thảo luận

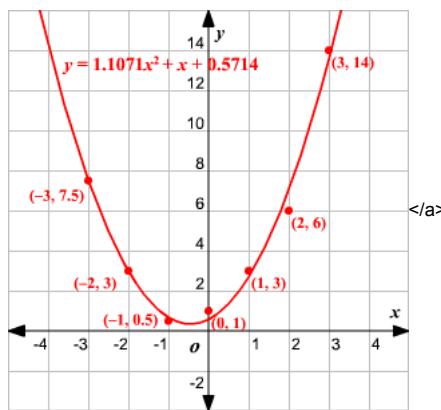
### 4.1. Các bài toán có thể giải bằng Linear Regression

Hàm số  $y \approx f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  là một hàm tuyến tính theo cả  $\mathbf{w}$  và  $\mathbf{x}$ . Trên thực tế, Linear Regression có thể áp dụng cho các mô hình chỉ cần tuyến tính theo  $\mathbf{w}$ . Ví dụ:

$$\begin{aligned} y \approx & w_1 x_1 + w_2 x_2 + w_3 x_1^2 + \\ & + w_4 \sin(x_2) + w_5 x_1 x_2 + w_0 \end{aligned}$$

là một hàm tuyến tính theo  $\mathbf{w}$  và vì vậy cũng có thể được giải bằng Linear Regression. Với mỗi dữ liệu đầu vào  $\mathbf{x} = [x_1; x_2]$ , chúng ta tính toán dữ liệu mới  $\tilde{\mathbf{x}} = [x_1, x_2, x_1^2, \sin(x_2), x_1 x_2]$  (đọc là  $x \tilde{}$  trong tiếng Anh) rồi áp dụng Linear Regression với dữ liệu mới này.

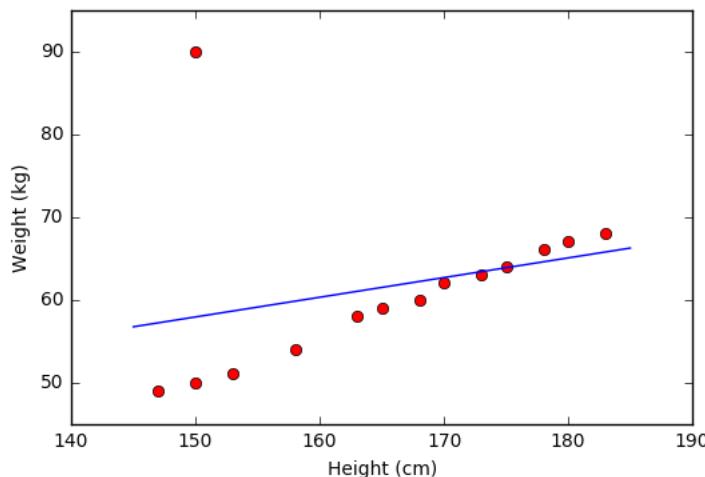
Xem thêm ví dụ về Quadratic Regression ([http://www.varsitytutors.com/hotmath/hotmath\\_help/topics/quadratic-regression](http://www.varsitytutors.com/hotmath/hotmath_help/topics/quadratic-regression)) (Hồi Quy Bậc Hai).



Quadratic Regression (Nguồn: Quadratic Regression ([http://www.varsitytutors.com/hotmath/hotmath\\_help/topics/quadratic-regression](http://www.varsitytutors.com/hotmath/hotmath_help/topics/quadratic-regression)))

### 4.2. Hạn chế của Linear Regression

Hạn chế đầu tiên của Linear Regression là nó rất **nhạy cảm với nhiễu** (sensitive to noise). Trong ví dụ về mối quan hệ giữa chiều cao và cân nặng bên trên, nếu có chỉ một cặp dữ liệu *nhiễu* (150 cm, 90kg) thì kết quả sẽ sai khác đi rất nhiều. Xem hình dưới đây:



Vì vậy, trước khi thực hiện Linear Regression, các nhiễu (*outlier*) cần phải được loại bỏ. Bước này được gọi là tiền xử lý (pre-processing).

Hạn chế thứ hai của Linear Regression là nó **không biểu diễn được các mô hình phức tạp**. Mặc dù trong phần trên, chúng ta thấy rằng phương pháp này có thể được áp dụng nếu quan hệ giữa *outcome* và *input* không nhất thiết phải là tuyến tính, nhưng mối quan hệ này vẫn đơn giản nhiều so với các mô hình thực tế. Hơn nữa, chúng ta sẽ tự hỏi: làm thế nào để xác định được các hàm  $x_1^2, \sin(x_2), x_1x_2$  như ở trên?!

### 4.3. Các phương pháp tối ưu

Linear Regression là một mô hình đơn giản, lời giải cho phương trình đạo hàm bằng 0 cũng khá đơn giản. *Trong hầu hết các trường hợp, chúng ta không thể giải được phương trình đạo hàm bằng 0.*

Nhưng có một điều chúng ta nên nhớ, **còn tính được đạo hàm là còn có hy vọng**.

## 5. Tài liệu tham khảo

1. Linear Regression - Wikipedia ([https://en.wikipedia.org/wiki/Linear\\_regression](https://en.wikipedia.org/wiki/Linear_regression))
2. Simple Linear Regression Tutorial for Machine Learning (<http://machinelearningmastery.com/simple-linear-regression-tutorial-for-machine-learning/>)
3. Least Squares, Pseudo-Inverses, PCA & SVD (<http://www.sci.utah.edu/~gerig/CS6640-F2012/Materials/pseudoinverse-cis61009sl10.pdf>)

Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (<https://www.facebook.com/groups/257768141347267/>) để nhận được câu trả lời sớm hơn.

Bạn đọc có thể ủng hộ blog qua 'Buy me a coffee' (/buymeacoffee/) ở góc trên bên trái của blog.

Tôi đang trong quá trình viết cuốn sách 'Machine Learning cơ bản I', các bạn có thể đặt trước tại đây (/ebook/). Cảm ơn bạn.

« Bài 2: Phân nhóm các thuật toán Machine Learning (/2016/12/27/categories/)

Bài 4: K-means Clustering » (/2017/01/01/kmeans/)

66 Comments tiepvu

1 Login ▾

Recommend 14 Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS

Name



đức nam đoàn • 7 months ago

Tác giả viết bài rất có tâm, em cũng tham khảo nhiều blog machine learning cả tiếng anh, và tiếng việt, thì thấy blog rất hay và dễ hiểu ngay cả đối với người mới. Cảm ơn anh!

7 ^ | v • Reply • Share ›



Thien Cong Nguyen • 3 months ago

Anh viết thêm nhiều chuyên mục lên, vì ở Việt Nam ở đâu cũng share bài của anh hết :)

2 ^ | v • Reply • Share &gt;



Lê Hải Sơn • 4 months ago

Bài viết rất hay , xem vid của andrew Ng thì có mấy đoạn ko nghe ra là gì thì tác giả đã giải thích rồi

2 ^ | v • Reply • Share &gt;



Düng Ict • 2 months ago

Bài viết rất hay a . Tks a nhé

1 ^ | v • Reply • Share &gt;



Hoàng Phạm • 3 months ago

Chao anh. Em doc blog va phat hien a typo: ma tran kha nghich la "invertible" matrix, not "inversable".

1 ^ | v • Reply • Share &gt;



Tiep Vu Huu Mod → Hoàng Phạm • 3 months ago

Cảm ơn em. Anh đã fix lại lỗi đó.

^ | v • Reply • Share &gt;



Chu\_Chính • 4 months ago

RẤT CÁM ƠN ANH ĐÃ CHIA SẼ KIẾN THỨC, CHÚC ANH SỨC KHỎE VÀ THÀNH CÔNG HƠN NỮA

1 ^ | v • Reply • Share &gt;



Đặng Thái Hòa • 5 months ago

Cảm ơn tác giả nhiều nhiều chúc tác giả ra thêm nhiều tác phẩm hay nữa

1 ^ | v • Reply • Share &gt;



Minh Trọc • 6 months ago

Trong bài giảng của GS. Andrew Ng nói đến cost function giống gióng hàm mất mát mà bạn đề cập nhưng là (1/2N) <https://www.coursera.org/learn/machine-learning/lecture/1000/cost-function-and-gradients>

Xin hỏi có phải 2 hàm này là một? Nếu là một thì công thức nào đúng?

1 ^ | v • Reply • Share &gt;



Tiep Vu Huu Mod → Minh Trọc • 6 months ago

Như nhau thôi bạn. Thêm số 2 ở mẫu số để lúc tính đạo hàm bị triệt tiêu.

Còn việc tìm biến số w để  $f(w)$ ,  $10f(w)$  hay  $100f(w)$  đạt giá trị nhỏ nhất thì kết quả cũng như nhau thôi.

1 ^ | v • Reply • Share &gt;



Thanh Nguyen • 10 months ago

1 comment nhỏ ở phần Nghiêm... Đạo hàm theo w thiếu mất dấu -. Ngoài ra cũng trong phần đó, nghiêm sử dụng pseudo-inverse là tối ưu theo Least square sense. Nếu có thể thì cậu viết 1 chút về cách tính A dagger từ SVD. Điều đó có thể đặt ra câu hỏi về phức tạp tính toán nếu số chiều và training size lớn và dẫn đến những cách tối ưu hiệu quả hơn

1 ^ | v • Reply • Share &gt;



Tiep Vu Huu Mod → Thanh Nguyen • 10 months ago

Cảm ơn bạn. Mình đã sửa phần đấy.

Về giả nghịch đảo, nếu có dịp mình sẽ nói sau. Vì là bài đầu tiên, và cũng đã đủ dài nên mình không muốn làm người đọc cảm thấy ngợp.

1 ^ | v • Reply • Share &gt;



Tuyền Chim • 4 days ago

đọc bài này 2 lần rồi, khá thầm, mỗi tội mình thích kiểu code thuần hết không dùng thư viện hon, như vậy cảm thấy thầm hơn chứ dùng thư viện lâu lâu lại quyên mất :D

^ | v • Reply • Share &gt;



Hung Nguyen • 16 days ago

Cho em hỏi với ạ. Theo em biết thì đạo hàm của hàm hợp  $[f(g(x))]'$  =  $f'(g).g'(x)$ .Vậy tại sao lại để  $Xbar^T$  ở đằng trước hàm?

Em cảm ơn!

^ | v • Reply • Share &gt;



Khánh Hòa Nguyễn • 18 days ago

Cho mình hỏi cách tìm nghiệm của bạn trên bài viết này là dùng phương pháp OLS (Ordinary Least Squares) ko? Nếu phải thì tại sao khi mình dùng thư viện statsmodels.api, gọi hàm .OLS thì lại ra kết quả ko giống với kết quả của bạn vậy?

[^](#) [v](#) [• Reply](#) [Share](#)



Tiep Vu Huu Mod → Khánh Hòa Nguyễn • 18 days ago

Bạn có thể chia sẻ code được không. Có code thì dễ thảo luận hơn.

[^](#) [v](#) [• Reply](#) [Share](#)



Khánh Hòa Nguyễn → Tiep Vu Huu • 17 days ago

Hôm qua mình có sai sót trong việc truyền dữ liệu đầu vào. Giờ thì kết quả đã ra đúng rồi bạn. Thanks bạn nhiều

[^](#) [v](#) [• Reply](#) [Share](#)



Tạ Anh Tú • 23 days ago

A có bài nào về hướng dẫn cài Python, IDE để code và Python cơ bản ko ạ?

[^](#) [v](#) [• Reply](#) [Share](#)



Tiep Vu Huu Mod → Tạ Anh Tú • 23 days ago

Đây em:

<https://machinelearningcoba...>

[^](#) [v](#) [• Reply](#) [Share](#)



linh tran • a month ago

Bài viết rất rõ ràng và dễ hiểu. Cám ơn anh đã chia sẻ!

[^](#) [v](#) [• Reply](#) [Share](#)



Nguyen Thanh Truong • 2 months ago

Giống khớp hàm trong xử lý số liệu

[^](#) [v](#) [• Reply](#) [Share](#)



Phạm Quốc Huy • 3 months ago

Chào anh, em mới bắt đầu học về Learning machine. Em thấy có rất nhiều phần toán trong bài viết, anh có thể chia sẻ anh đã học các kiến thức đó ở đâu không, em muốn học đầy đủ từ nền tảng toán (nếu có sách anh giới thiệu thì càng tốt :)). Cám ơn anh.

[^](#) [v](#) [• Reply](#) [Share](#)



Tiep Vu Huu Mod → Phạm Quốc Huy • 3 months ago

Chào em. Phần toán thi em nên học Đại Số Tuyến Tính và Xác Suất Thống Kê thật kỹ, thêm một chút về hàm nhiều biến nữa. Các kiến thức này ban đầu anh đều được học ở trường đại học ở VN. Em tìm hai cuốn Toán cao cấp phần đại số của Nguyễn Đình Trí và Xác Suất Thống Kê của Tổng Đinh Quy. Anh không nhớ chính xác tên nhưng tác giả chính xác là hai thầy này.

[1](#) [^](#) [v](#) [• Reply](#) [Share](#)



Phạm Quốc Huy → Tiep Vu Huu • 3 months ago

Cám ơn anh nhiều, em sẽ cố gắng theo đuổi ngành này. Chúc anh có nhiều sức khỏe và có thêm nhiều bài viết hay :)

[1](#) [^](#) [v](#) [• Reply](#) [Share](#)



Thành Công Nguyễn • 3 months ago

Hình như cái gradient là vector-valued function anh ơi. Trong cái tóm tắt toán á.

[^](#) [v](#) [• Reply](#) [Share](#)



Tiep Vu Huu Mod → Thành Công Nguyễn • 3 months ago

Chỗ nào em nhỉ. Anh chưa hiểu ý em lắm.

[^](#) [v](#) [• Reply](#) [Share](#)



Đông Iv • 5 months ago

mọi người cho em hỏi phần code  $w_0 = w[0][0]$ ,  $w_1 = w[1][0]$  là như thế nào ạ, sao lại lấy  $w_0$  lại là  $w[0][0]$ , tương tự như thế với  $w_1$ , mình lấy giá trị khác được không ạ, giả dụ  $w_0 = w[0]$  chẳng hạn ạ, em xin chân thành cảm ơn!

[^](#) [v](#) [• Reply](#) [Share](#)



Thanh Pham Chi → Đông Iv • 4 months ago

Ma trận  $w$  là ma trận  $[2 \times 1]$  có nghĩa là 2 hàng và 1 cột. Vì vậy khi lấy giá trị của một ma trận thì bạn phải có vị trí của hàng và của cột mới lấy được đúng không. Trong trường hợp này ma trận này giống với vector cột chỉ có 1 cột duy nhất và 2 hàng thì bạn có thể dùng  $w_0 = w[0]$  được vẫn đảm bảo độ chính xác. Nhưng nếu lấy ví dụ là ma trận  $2 \times 2$  thì khi bạn viết  $w[0]$  thì nó sẽ lấy nguyên cả cái hàng đầu tiên ra. Hy vọng sẽ giúp được bạn vì hỏi hồn thảng rồi :D

[1](#) [^](#) [v](#) [• Reply](#) [Share](#)



Đông Iv → Thanh Pham Chi • 4 months ago

cảm ơn bạn nhiều nhá, he.

^ | v • Reply • Share •

 **Thanh Pham Chi** → Đông lv • 4 months ago

Ok có gì không hiểu cứ hỏi ở đây nhé, đừng để mất hứng thú học là được :v

^ | v • Reply • Share •

 **Đông lv** → Thanh Pham Chi • 4 months ago

uh, cảm ơn bạn, mình đang làm với tập MNIST trên tensorflow, kích thước ảnh của nó là 28\*28, giờ mình muốn chuyển về 22\*22, mà loay hoay mãi chưa làm được, tìm trên mạng cũng không thấy, bạn biết cách làm không?

^ | v • Reply • Share •

 **Thanh Pham Chi** → Đông lv • 4 months ago

Thử câu lệnh này xem

```
new_images=tf.image.resize_images(old_images, [22, 22])
```

^ | v • Reply • Share •

 **Đông lv** → Thanh Pham Chi • 4 months ago

Mình cũng thử rồi nhưng bị lỗi bạn ạ, để có gì hỏi mọi người tiếp chứ mình cũng chịu rồi, cảm ơn bạn rất nhiều,

^ | v • Reply • Share •

 **Duy Thong** • 7 months ago

Bạn ơi cho mình hỏi, khi mình chạy code với cách dùng thư viện scikit-learn, mình không dùng hai câu lệnh này:

```
#regr=linear_model.LinearRegression(fit_intercept=False)
```

```
#regr.fit(Xbar,y)
```

Kết quả là mình vẫn in được kết quả của regr.coef\_

```
print('Solution found by scikit-learn : ', regr.coef_ )
```

```
## Solution found by scikit-learn : [[-33.73541021 0.55920496]]
```

Mình mới bắt đầu code python nên chưa rõ lắm. Mong bạn giải thích dùm mình với. Cảm ơn bạn

^ | v • Reply • Share •

 **Tiep Vu Huu** Mod → Duy Thong • 7 months ago

Cách tốt nhất là bạn tìm mô tả của hàm linear\_model.LinearRegression trên sklearn (Google sẽ thấy). Về câu lệnh, nếu không rõ thì cứ Google thôi.

^ | v • Reply • Share •

 **Hoa Huang Duong** • 7 months ago

Hi bạn, mình có 1 chút mơ hồ với Linear Regression (LR). Mình học về xác suất thống kê, cũng có lý thuyết về LR, cách fitting cũng là minimize "error sum of squares" giữa dữ liệu quan sát được (observed data), và dữ liệu mô hình (fitted data). Khi mình đọc bài viết này, thì mình thấy ML cũng có bài toán LR mà cách làm thì như bên thống kê. Tuy nhiên, về bên thống kê, mình chỉ fitting được với dạng hàm linear. Còn nếu mối liên hệ giữa output y và input x ko fai linear, nếu dùng matlab mình có thể fitting cho hàm số đa thức (polynomial) hoặc hàm exp(). Với cách làm này, người dùng vô tình "ép" mối liên hệ của y và x dưới 2 dạng: polynomial hoặc exp. Theo Stone–Weierstrass theorem, thì bất kỳ 1 function liên tục nào cũng đều biểu diễn được dưới dạng polynomial. Tuy nhiên, trên thực tế, khi fitting cho những đường cong như vậy, sai số là lớn. Mình có 1 tập hợp (x,y) khi vẽ ra mình thấy nó có đường cong giống dạng biểu diễn mũ exp và liên tục. Tuy nhiên, khi mình fitting (sử dụng matlab cho mô hình exp) thì mình thu được sai số residual lớn. Mình thắc mắc, liệu trong ML, nếu có bộ dữ liệu quan sát (x,y), và mình không chỉ định dạng của model \hat{y} là dạng gì hết. Liệu ML có thể tìm được dạng biểu diễn \hat{y} phù hợp với các hệ số tối ưu nhất, thể hiện được liên hệ của y và x?

^ | v • Reply • Share •

 **Hoàn DK** → Hoa Huang Duong • 2 months ago

chào bạn, về việc 2 variables có mối liên hệ không phải linear, trong R bạn có thể dùng lệnh lowess() để biểu thị mối quan hệ giữa 2 variable này. bằng cách làm này bạn cũng có thể dùng lệnh approx() trong R để ước lượng giá trị trung bình của y given giá trị của x. điểm yếu của cách làm này là bạn chỉ có thể ước lượng y trong khoảng data range mà bạn thu thập được.

ngoài ra khi 2 variable ko có mhq linear. bạn có thể dùng sqrt(x) sqrt(y) x^ y^ log(x) log(y) (log natural trong R) để đưa mhq về linear. bởi nếu các assumption của linear model ko dc đảm bảo model của bạn sẽ ko thể sử dụng được.

về tài liệu, bạn có thể tìm đọc cuốn này <https://www.amazon.com/Stat...>

^ | v • Reply • Share •

 **Tiep Vu Huu** Mod → Hoa Huang Duong • 7 months ago

Chào bạn. Hiện chưa có mô hình nào tự động làm được việc đó cả.

Về polynomial regression, bạn có thể đọc thêm Bài 15 để có thêm một góc nhìn quan trọng khác:

<http://machinelearningcaban...>

^ | v • Reply • Share •

 **Hoa Huang Duong** → Tiep Vu Huu • 5 months ago

Mình đã đọc bài này của bạn khi nhận được trả lời từ bạn nhưng đến bây giờ mình mới nghiệm ra điều bạn nói về overfitting. Mình học hơi chậm, nhưng các bài viết của bạn giúp mình tự học và củng cố kiến thức rất nhiều.

1 ^ | v • Reply • Share •



**Tiep Vu Huu** Mod → Hoa Huong Duong • 5 months ago  
Cảm ơn bạn. Rất vui vì blog hữu ích với bạn.

Học Machine Learning không vội được, mọi thứ sẽ được ngấm dần dần. Rất cần sự kiên nhẫn và niềm đam mê để theo đuổi. Chúc bạn thành công.

^ | v · Reply · Share >



**Andy Cole** • 7 months ago

cho mình hỏi cái là trong code python tại sao lại có thêm khúc

```
# Building Xbar
one = np.ones((X.shape[0], 1))
Xbar = np.concatenate((one, X), axis = 1)
```

theo mình hiểu code thì là insert thêm 1 cột với giá trị là 1 vào matrix X.

Trong mấy công thức toán ở trên thì đâu có nói cái này, sao code lại có?

Và tại sao lại thêm 1 cột (không phải là 2 cột hay 4 cột)?

Và tại sao giá trị là 1 (sao không dùng giá trị khác như 2 hay 8)?

Mình thử không chạy code này thì thấy w chỉ có 1 giá trị, tại sao nhỉ?

Mình mới học nên nhiều thắc mắc, mong được bạn giúp đỡ

^ | v · Reply · Share >



**Thanh Nguyen** → Andy Cole • 7 months ago

mình có thể giúp bạn trả lời thắc mắc (vì trc đây mình cũng có thắc mắc giống bạn và tự tìm câu trả lời)

1. việc thêm giá trị 1 vào vi để có tương ứng với hệ số w0 trong công thức (1) trong bài. nếu là giá trị 2 hay 8 thì công thức sẽ trở thành 2w0 hay 8w0. để đơn giản, ko mất tính tổng quát, ta lấy giá trị 1

2. thêm 1 cột mà ko phải nhiều hơn vi để tương ứng với các kích thước dimension của matrix. bạn thử ngồi viết các dimension của matrix trong công thức (5) trong bài thì bạn sẽ rõ

2 ^ | v · Reply · Share >



**Andy Cole** → Thanh Nguyen • 7 months ago

à, thanks bạn nhiều

^ | v · Reply · Share >



**Tiep Vu Huu** Mod → Andy Cole • 7 months ago

Tôi đã đề cập trong bài viết. Bạn đọc kỹ phần này sẽ thấy:

<http://machinelearningcoban...>

1 ^ | v · Reply · Share >



**Andy Cole** → Tiep Vu Huu • 7 months ago

ok bạn, mình đọc lướt cái đó nên k để, giờ thì mình đã clear, thanks.

^ | v · Reply · Share >



**Ky Giang Phan** → Andy Cole • 6 months ago

Bạn ơi phương thức concatenate dùng để làm gì vậy? Với X.shape[0] có ý nghĩa gì? Mong bạn giúp đỡ, mình tìm tài liệu mà không có.

^ | v · Reply · Share >



**Tiep Vu Huu** Mod → Ky Giang Phan • 6 months ago

Chào bạn,

np.concatenate là 'ghép' hai mảng nhiều chiều lại với nhau, ví dụ chồng hai ma trận lên nhau hoặc đặt chúng cạnh nhau.

Còn X.shape[0] là lấy chiều của X theo axis thứ nhất.

Đây là những thuộc tính và phương thức của numpy. Bạn có thể thấy trong Numpy Manual:

<https://docs.scipy.org/doc/...>

^ | v · Reply · Share >



**Nick Chung** • 8 months ago

Cái bộ lọc Kalman này thì sao ạ:

<https://tamgiang.wordpress....>

^ | v · Reply · Share >



**Tiep Vu Huu** Mod → Nick Chung • 8 months ago

Tôi không hiểu ý của bạn.

^ | v · Reply · Share >

**Nick Chung** → Tiep Vu Huu • 8 months ago

Anh ơi anh có thể viết nhiều ví dụ minh họa hơn nữa cho các bài 2,3,4 được không ạ...

^ | v · Reply · Share &gt;

[Load more comments](#)

ALSO ON TIEPVU

**Machine Learning cơ bản**

11 comments • 8 months ago

CTVR — Em cũng thấy yêu cầu 64 bit trên win. Chắc em chạy với linux vậy.

P/s: Giờ em mới đọc xong loạt bài về convex trên file pdf. Hồi mới vào ...

[Subscribe](#) [Add Disqus to your site](#) [Add Disqus](#) [Privacy](#)**ebook Machine Learning cơ bản**

26 comments • 3 months ago

Xuân Xuân — Anh ơi, cái email trường em cấp ý, em test nó không cho các mail ngoài gửi vào:"Hi. This is the qmail-send program at vnu.edu.vn.I'm ...

Total visits:

## Bài 4: K-means Clustering

Clustering (/tags#Clustering) Kmeans (/tags#Kmeans) Unsupervised-learning (/tags#Unsupervised-learning)

Jan 1, 2017

Trong trang này:

- 1. Giới thiệu
- 2. Phân tích toán học
  - Một số ký hiệu toán học
  - Hàm mất mát và bài toán tối ưu
  - Thuật toán tối ưu hàm mất mát
    - Cố định  $\mathbf{M}$ , tìm  $\mathbf{Y}$
    - Cố định  $\mathbf{Y}$ , tìm  $\mathbf{M}$
  - Tóm tắt thuật toán
- Ví dụ trên Python
  - Giới thiệu bài toán
  - Hiển thị dữ liệu trên đồ thị
  - Các hàm số cần thiết cho K-means clustering
  - Kết quả tìm được bằng thư viện scikit-learn
- 4. Thảo luận
  - Hạn chế
    - Chúng ta cần biết số lượng cluster cần clustering
    - Nghiệm cuối cùng phụ thuộc vào các centers được khởi tạo ban đầu
    - Các cluster cần có số lượng điểm gần bằng nhau
    - Các cluster cần có dạng hình tròn
    - Khi một cluster nằm phía trong 1 cluster khác
- 5. Tài liệu tham khảo

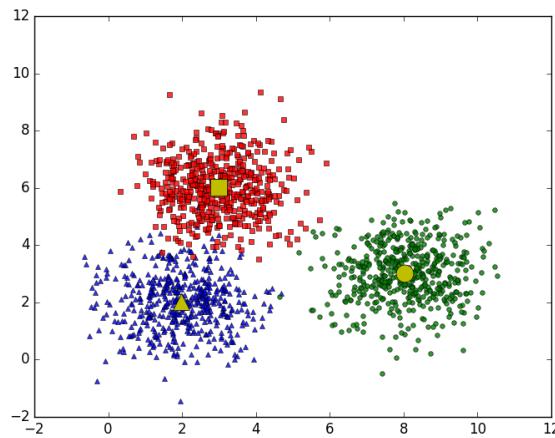
### 1. Giới thiệu

Trong bài trước (/2016/12/28/linearregression/), chúng ta đã làm quen với thuật toán Linear Regression - là thuật toán đơn giản nhất trong Supervised learning (/2016/12/27/categories/#supervised-learning-hoc-co-giam-sat). Bài này tôi sẽ giới thiệu một trong những thuật toán cơ bản nhất trong Unsupervised learning (/2016/12/27/categories/#unsupervised-learning-hoc-khong-giam-sat) - thuật toán K-means clustering (phân cụm K-means).

Trong thuật toán K-means clustering, chúng ta không biết nhãn (label) của từng điểm dữ liệu. Mục đích là làm thế nào để phân dữ liệu thành các cụm (cluster) khác nhau sao cho *dữ liệu trong cùng một cụm có tính chất giống nhau*.

**Ví dụ:** Một công ty muốn tạo ra những chính sách ưu đãi cho những nhóm khách hàng khác nhau dựa trên sự tương tác giữa mỗi khách hàng với công ty đó (số năm là khách hàng; số tiền khách hàng đã chi trả cho công ty; độ tuổi; giới tính; thành phố; nghề nghiệp; ...). Giả sử công ty đó có rất nhiều dữ liệu của rất nhiều khách hàng nhưng chưa có cách nào chia toàn bộ khách hàng đó thành một số nhóm/cụm khác nhau. Nếu một người biết Machine Learning được đặt câu hỏi này, phương pháp đầu tiên anh (chỉ) ta nghĩ đến sẽ là K-means Clustering. Vì nó là một trong những thuật toán đầu tiên mà anh ấy tìm được trong các cuốn sách, khóa học về Machine Learning. Và tôi cũng chắc rằng anh ấy đã đọc blog Machine Learning cơ bản (<https://tiepvupsu.github.io>). Sau khi đã phân ra được từng nhóm, nhân viên công ty đó có thể lựa chọn ra một vài khách hàng trong mỗi nhóm để quyết định xem mỗi nhóm tương ứng với nhóm khách hàng nào. Phần việc cuối cùng này cần sự can thiệp của con người, nhưng lượng công việc đã được rút gọn đi rất nhiều.

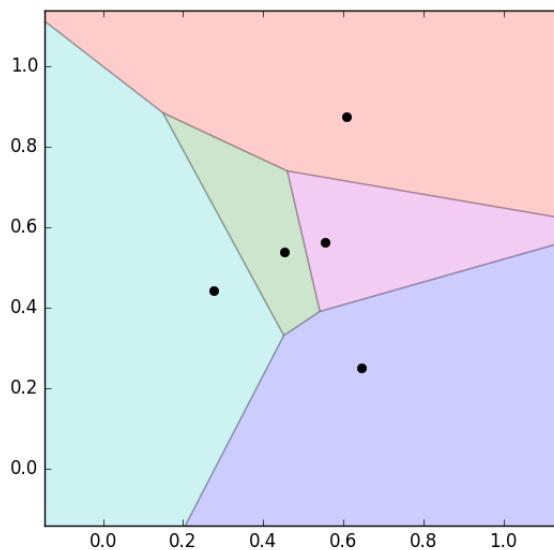
Ý tưởng đơn giản nhất về cluster (cụm) là tập hợp các điểm *ở gần nhau trong một không gian nào đó* (không gian này có thể có rất nhiều chiều trong trường hợp thông tin về một điểm dữ liệu là rất lớn). Hình bên dưới là một ví dụ về 3 cụm dữ liệu (từ giờ tôi sẽ viết gọn là *cluster*).



Bài toán với 3 clusters.

Giả sử mỗi cluster có một điểm đại diện (*center*) màu vàng. Và những điểm xung quanh mỗi center thuộc vào cùng nhóm với center đó. Một cách đơn giản nhất, xét một điểm bất kỳ, ta xét xem điểm đó gần với center nào nhất thì nó thuộc về cùng nhóm với center đó. Tới đây, chúng ta có một bài toán thú vị: *Trên một vùng biển hình vuông lớn có ba đảo hình vuông, tam giác, và tròn màu vàng như hình trên. Một điểm trên biển được gọi là thuộc lãnh hải của một đảo nếu nó nằm gần đảo này hơn so với hai đảo kia. Hãy xác định ranh giới lãnh hải của các đảo.*

Hình dưới đây là một hình minh họa cho việc phân chia lãnh hải nếu có 5 đảo khác nhau được biểu diễn bằng các hình tròn màu đen:



Phân vùng lãnh hải của mỗi đảo. Các vùng khác nhau có màu sắc khác nhau.

Chúng ta thấy rằng đường phân định giữa các lãnh hải là các đường thẳng (chính xác hơn thì chúng là các đường trung trực của các cặp điểm gần nhau). Vì vậy, lãnh hải của một đảo sẽ là một hình đa giác.

Cách phân chia này trong toán học được gọi là Voronoi Diagram ([https://en.wikipedia.org/wiki/Voronoi\\_diagram](https://en.wikipedia.org/wiki/Voronoi_diagram)).

Trong không gian ba chiều, lấy ví dụ là các hành tinh, thì (tạm gọi là) *lãnh không* của mỗi hành tinh sẽ là một đa diện. Trong không gian nhiều chiều hơn, chúng ta sẽ có những thứ (mà tôi gọi là) *siêu đa diện* (hyperpolygon).

Quay lại với bài toán phân nhóm và cụ thể là thuật toán K-means clustering, chúng ta cần một chút phân tích toán học trước khi đi tới phần tóm tắt thuật toán ở phần dưới. Nếu bạn không muốn đọc quá nhiều về toán, bạn có thể bỏ qua phần này. (*Tốt nhất là đừng bỏ qua, bạn sẽ tiếc đấy*).

## 2. Phân tích toán học

Mục đích cuối cùng của thuật toán phân nhóm này là: từ dữ liệu đầu vào và số lượng nhóm chúng ta muốn tìm, hãy chỉ ra center của mỗi nhóm và phân các điểm dữ liệu vào các nhóm tương ứng. Giả sử thêm rằng mỗi điểm dữ liệu chỉ thuộc vào đúng một nhóm.

### Một số ký hiệu toán học

Giả sử có  $N$  điểm dữ liệu là  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N] \in \mathbb{R}^{d \times N}$  và  $K < N$  là số cluster chúng ta muốn phân chia. Chúng ta cần tìm các center  $\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_K \in \mathbb{R}^{d \times 1}$  và label của mỗi điểm dữ liệu.

**Lưu ý về ký hiệu toán học:** trong các bài viết của tôi, các số vô hướng được biểu diễn bởi các chữ cái viết ở dạng không in đậm, ví dụ  $x_1, N, y, k$ . Các vector được biểu diễn bằng các chữ cái thường in đậm, ví dụ  $\mathbf{m}, \mathbf{x}_1$ . Các ma trận được biểu diễn bởi các chữ viết hoa in đậm, ví dụ  $\mathbf{X}, \mathbf{M}, \mathbf{Y}$ . Lưu ý này đã được nêu ở bài Linear Regression (/2016/12/28/linearregression/). Tôi xin được không nhắc lại trong các bài tiếp theo.

Với mỗi điểm dữ liệu  $\mathbf{x}_i$  đặt  $\mathbf{y}_i = [y_{i1}, y_{i2}, \dots, y_{iK}]$  là label vector của nó, trong đó nếu  $\mathbf{x}_i$  được phân vào cluster  $k$  thì  $y_{ik} = 1$  và  $y_{ij} = 0, \forall j \neq k$ . Điều này có nghĩa là có đúng một phần tử của vector  $\mathbf{y}_i$  là bằng 1 (tương ứng với cluster của  $\mathbf{x}_i$ ), các phần tử còn lại bằng 0. Ví dụ: nếu một điểm dữ liệu có label vector là  $[1, 0, 0, \dots, 0]$  thì nó thuộc vào cluster 1, là  $[0, 1, 0, \dots, 0]$  thì nó thuộc vào cluster 2, ... . Cách mã hóa label của dữ liệu như thế này được gọi là biểu diễn one-hot (<https://en.wikipedia.org/wiki/One-hot>). Chúng ta sẽ thấy cách biểu diễn one-hot này rất phổ biến trong Machine Learning ở các bài tiếp theo.

Ràng buộc của  $\mathbf{y}_i$  có thể viết dưới dạng toán học như sau:

$$y_{ik} \in \{0, 1\}, \quad \sum_{k=1}^K y_{ik} = 1 \quad (1)$$

## Hàm mất mát và bài toán tối ưu

Nếu ta coi center  $\mathbf{m}_k$  là center (hoặc representative) của mỗi cluster và ước lượng tất cả các điểm được phân vào cluster này bởi  $\mathbf{m}_k$ , thì một điểm dữ liệu  $\mathbf{x}_i$  được phân vào cluster  $k$  sẽ bị sai số là  $(\mathbf{x}_i - \mathbf{m}_k)$ . Chúng ta mong muốn sai số này có trị tuyệt đối nhỏ nhất nên (giống như trong bài Linear Regression (/2016/12/28/linearregression/#sai-so-du-doan)) ta sẽ tìm cách để đại lượng sau đây đạt giá trị nhỏ nhất:

$$\|\mathbf{x}_i - \mathbf{m}_k\|_2^2$$

Hơn nữa, vì  $\mathbf{x}_i$  được phân vào cluster  $k$  nên  $y_{ik} = 1, y_{ij} = 0, \forall j \neq k$ . Khi đó, biểu thức bên trên sẽ được viết lại là:

$$y_{ik}\|\mathbf{x}_i - \mathbf{m}_k\|_2^2 = \sum_{j=1}^K y_{ij}\|\mathbf{x}_i - \mathbf{m}_j\|_2^2$$

(*Hy vọng chọn này không quá khó hiểu*)

Sai số cho toàn bộ dữ liệu sẽ là:

$$\mathcal{L}(\mathbf{Y}, \mathbf{M}) = \sum_{i=1}^N \sum_{j=1}^K y_{ij}\|\mathbf{x}_i - \mathbf{m}_j\|_2^2$$

Trong đó  $\mathbf{Y} = [\mathbf{y}_1; \mathbf{y}_2; \dots; \mathbf{y}_N], \mathbf{M} = [\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_K]$  lần lượt là các ma trận được tạo bởi label vector của mỗi điểm dữ liệu và center của mỗi cluster. Hàm số mất mát trong bài toán K-means clustering của chúng ta là hàm  $\mathcal{L}(\mathbf{Y}, \mathbf{M})$  với ràng buộc như được nêu trong phương trình (1).

Tóm lại, chúng ta cần tối ưu bài toán sau:

$$\mathbf{Y}, \mathbf{M} = \arg \min_{\mathbf{Y}, \mathbf{M}} \sum_{i=1}^N \sum_{j=1}^K y_{ij}\|\mathbf{x}_i - \mathbf{m}_j\|_2^2 \quad (2)$$

$$\text{subject to: } y_{ij} \in \{0, 1\} \quad \forall i, j; \quad \sum_{j=1}^K y_{ij} = 1 \quad \forall i$$

(*subject to* nghĩa là *thỏa mãn điều kiện*).

**Nhắc lại khái niệm arg min:** Chúng ta biết ký hiệu min là *giá trị nhỏ nhất của hàm số*, arg min chính là *giá trị của biến số để hàm số đó đạt giá trị nhỏ nhất đó*. Nếu  $f(x) = x^2 - 2x + 1 = (x - 1)^2$  thì giá trị nhỏ nhất của hàm số này bằng 0, đạt được khi  $x = 1$ . Trong ví dụ này  $\min_x f(x) = 0$  và  $\arg \min_x f(x) = 1$ . Thêm ví dụ khác, nếu  $x_1 = 0, x_2 = 10, x_3 = 5$  thì ta nói  $\arg \min_i x_i = 1$  vì 1 là chỉ số để  $x_i$  đạt giá trị nhỏ nhất (bằng 0). Biến số viết bên dưới min là biến số chúng ta cần tối ưu. Trong các bài toán tối ưu, ta thường quan tâm tới arg min hơn là min.

## Thuật toán tối ưu hàm mất mát

Bài toán (2) là một bài toán khó tìm điểm tối ưu vì nó có thêm các điều kiện ràng buộc. *Bài toán này thuộc loại mix-integer programming (điều kiện biến là số nguyên) - là loại rất khó tìm nghiệm tối ưu toàn cục (global optimal point, tức nghiệm làm cho hàm mất mát đạt giá trị nhỏ nhất có thể)*. Tuy nhiên, trong một số trường hợp chúng ta vẫn có thể tìm được phương pháp để tìm được nghiệm gần đúng hoặc điểm cực tiểu. (*Nếu chúng ta vẫn nhớ chương trình toán ôn thi đại học thi điểm cực tiểu chưa chắc đã phải là điểm làm cho hàm số đạt giá trị nhỏ nhất*).

Một cách đơn giản để giải bài toán (2) là xen kẽ giải  $\mathbf{Y}$  và  $\mathbf{M}$  khi biến còn lại được cố định. Đây là một thuật toán lặp, cũng là kỹ thuật phổ biến khi giải bài toán tối ưu. Chúng ta sẽ lần lượt giải quyết hai bài toán sau đây:

## Cố định $\mathbf{M}$ , tìm $\mathbf{Y}$

**Giả sử đã tìm được các centers, hãy tìm các label vector để hàm mất mát đạt giá trị nhỏ nhất.** Điều này tương đương với việc tìm cluster cho mỗi điểm dữ liệu.

Khi các centers là cố định, bài toán tìm label vector cho toàn bộ dữ liệu có thể được chia nhỏ thành bài toán tìm label vector cho từng điểm dữ liệu  $\mathbf{x}_i$  như sau:

$$\mathbf{y}_i = \arg \min_{\mathbf{y}_i} \sum_{j=1}^K y_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|_2^2 \quad (3)$$

$$\text{subject to: } y_{ij} \in \{0, 1\} \quad \forall j; \quad \sum_{j=1}^K y_{ij} = 1$$

Vì chỉ có một phần tử của label vector  $\mathbf{y}_i$  bằng 1 nên bài toán (3) có thể tiếp tục được viết dưới dạng đơn giản hơn:

$$j = \arg \min_j \|\mathbf{x}_i - \mathbf{m}_j\|_2^2$$

Vì  $\|\mathbf{x}_i - \mathbf{m}_j\|_2^2$  chính là bình phương khoảng cách tính từ điểm  $\mathbf{x}_i$  tới center  $\mathbf{m}_j$ , ta có thể kết luận rằng **mỗi điểm  $\mathbf{x}_i$  thuộc vào cluster có center gần nó nhất!** Từ đó ta có thể dễ dàng suy ra label vector của từng điểm dữ liệu.

## Cố định $\mathbf{Y}$ , tìm $\mathbf{M}$

**Giả sử đã tìm được cluster cho từng điểm, hãy tìm center mới cho mỗi cluster để hàm mất mát đạt giá trị nhỏ nhất.**

Một khi chúng ta đã xác định được label vector cho từng điểm dữ liệu, bài toán tìm center cho mỗi cluster được rút gọn thành:

$$\mathbf{m}_j = \arg \min_{\mathbf{m}_j} \sum_{i=1}^N y_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|_2^2$$

Tới đây, ta có thể tìm nghiệm bằng phương pháp giải đạo hàm bằng 0, vì hàm cần tối ưu là một hàm liên tục và có đạo hàm xác định tại mọi điểm. **Và quan trọng hơn, hàm này là hàm convex (lồi) theo  $\mathbf{m}_j$  nên chúng ta sẽ tìm được giá trị nhỏ nhất và điểm tối ưu tương ứng.** Sau này nếu có dịp, tôi sẽ nói thêm về **tối ưu lồi** (*convex optimization*) - một mảng cực kỳ quan trọng trong toán tối ưu.

Đặt  $l(\mathbf{m}_j)$  là hàm bên trong dấu arg min, ta có đạo hàm:

$$\frac{\partial l(\mathbf{m}_j)}{\partial \mathbf{m}_j} = 2 \sum_{i=1}^N y_{ij} (\mathbf{m}_j - \mathbf{x}_i)$$

Giải phương trình đạo hàm bằng 0 ta có:

$$\begin{aligned} \mathbf{m}_j \sum_{i=1}^N y_{ij} &= \sum_{i=1}^N y_{ij} \mathbf{x}_i \\ \Rightarrow \mathbf{m}_j &= \frac{\sum_{i=1}^N y_{ij} \mathbf{x}_i}{\sum_{i=1}^N y_{ij}} \end{aligned}$$

Nếu để ý một chút, chúng ta sẽ thấy rằng mẫu số chính là phép đếm số lượng các điểm dữ liệu trong cluster  $j$  (*Bạn có nhận ra không?*). Còn tử số chính là **tổng các điểm dữ liệu** trong cluster  $j$ . (*Nếu bạn đọc vẫn nhớ điều kiện ràng buộc của các  $y_{ij}$  thì sẽ có thể nhanh chóng nhìn ra điều này*).

Hay nói một cách đơn giản hơn nhiều:  $\mathbf{m}_j$  là trung bình cộng của các điểm trong cluster  $j$ .

Tên gọi *K-means clustering* cũng xuất phát từ đây.

## Tóm tắt thuật toán

Tới đây tôi xin được tóm tắt lại thuật toán (**đặc biệt quan trọng với các bạn bỏ qua phần toán học bên trên**) như sau:

**Đầu vào:** Dữ liệu  $\mathbf{X}$  và số lượng cluster cần tìm  $K$ .

**Đầu ra:** Các center  $\mathbf{M}$  và label vector cho từng điểm dữ liệu  $\mathbf{Y}$ .

1. Chọn  $K$  điểm bất kỳ làm các center ban đầu.
2. Phân mỗi điểm dữ liệu vào cluster có center gần nó nhất.
3. Nếu việc gán dữ liệu vào từng cluster ở bước 2 không thay đổi so với vòng lặp trước nó thì ta dừng thuật toán.
4. Cập nhật center cho từng cluster bằng cách lấy trung bình cộng của tất cả các điểm dữ liệu đã được gán vào cluster đó sau bước 2.
5. Quay lại bước 2.

Chúng ta có thể đảm bảo rằng thuật toán sẽ dừng lại sau một số hữu hạn vòng lặp. Thật vậy, vì hàm mất mát là một số dương và sau mỗi bước 2 hoặc 3, giá trị của hàm mất mát bị giảm đi. Theo kiến thức về dãy số trong chương trình cấp 3: *nếu một dãy số giảm và bị chặn dưới thì nó hội tụ!* Hơn nữa, số lượng cách phân nhóm cho toàn bộ dữ liệu là hữu hạn nên đến một lúc nào đó, hàm mất mát sẽ không thể thay đổi, và chúng ta có thể dừng thuật toán tại đây.

Chúng ta sẽ có một vài thảo luận về thuật toán này, về những hạn chế và một số phương pháp khắc phục. Nhưng trước hết, hãy xem nó thể hiện như thế nào trong một ví dụ cụ thể dưới đây.

## Ví dụ trên Python

### Giới thiệu bài toán

Để kiểm tra mức độ hiểu quả của một thuật toán, chúng ta sẽ làm một ví dụ đơn giản (thường được gọi là *toy example*). Trước hết, chúng ta chọn center cho từng cluster và tạo dữ liệu cho từng cluster bằng cách lấy mẫu theo phân phối chuẩn có kỳ vọng là center của cluster đó và ma trận hiệp phương sai (covariance matrix) là ma trận đơn vị.

Trước tiên, chúng ta cần khai báo các thư viện cần dùng. Chúng ta cần `numpy` và `matplotlib` như trong bài Linear Regression (/2016/12/28/linearregression/) cho việc tính toán ma trận và hiển thị dữ liệu. Chúng ta cũng cần thêm thư viện `scipy.spatial.distance` để tính khoảng cách giữa các cặp điểm trong hai tập hợp một cách hiệu quả.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(11)
```

Tiếp theo, ta tạo dữ liệu bằng cách lấy các điểm theo phân phối chuẩn có kỳ vọng tại các điểm có tọa độ (2, 2), (8, 3) và (3, 6), ma trận hiệp phương sai giống nhau và là ma trận đơn vị. Mỗi cluster có 500 điểm. (*Chú ý rằng mỗi điểm dữ liệu là một hàng của ma trận dữ liệu.*

```
means = [[2, 2], [8, 3], [3, 6]]
cov = [[1, 0], [0, 1]]
N = 500
X0 = np.random.multivariate_normal(means[0], cov, N)
X1 = np.random.multivariate_normal(means[1], cov, N)
X2 = np.random.multivariate_normal(means[2], cov, N)

X = np.concatenate((X0, X1, X2), axis = 0)
K = 3

original_label = np.asarray([0]*N + [1]*N + [2]*N).T
```

### Hiển thị dữ liệu trên đồ thị

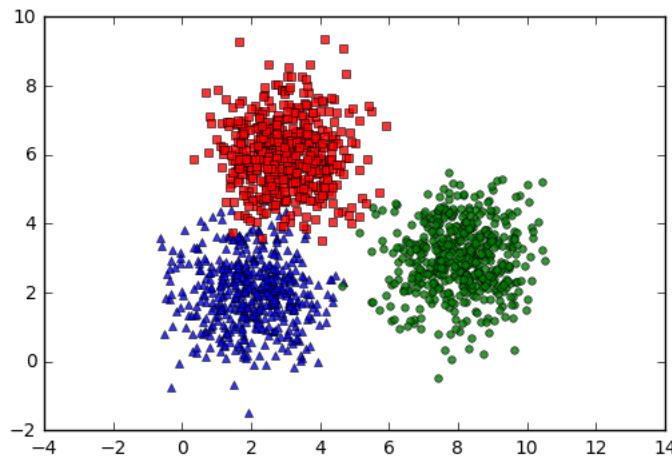
Chúng ta cần một hàm `kmeans_display` để hiển thị dữ liệu. Sau đó hiển thị dữ liệu theo nhãn ban đầu.

```
def kmeans_display(X, label):
    K = np.amax(label) + 1
    X0 = X[label == 0, :]
    X1 = X[label == 1, :]
    X2 = X[label == 2, :]

    plt.plot(X0[:, 0], X0[:, 1], 'b^', markersize = 4, alpha = .8)
    plt.plot(X1[:, 0], X1[:, 1], 'go', markersize = 4, alpha = .8)
    plt.plot(X2[:, 0], X2[:, 1], 'rs', markersize = 4, alpha = .8)

    plt.axis('equal')
    plt.plot()
    plt.show()

kmeans_display(X, original_label)
```



Trong đồ thị trên, mỗi cluster tương ứng với một màu. Có thể nhận thấy rằng có một vài điểm màu đỏ bị lắn sang phần cluster màu xanh.

## Các hàm số cần thiết cho K-means clustering

Viết các hàm:

1. kmeans\_init\_centers để khởi tạo các centers ban đầu.
2. kmeans\_assign\_labels để gán nhãn mới cho các điểm khi biết các centers.
3. kmeans\_update\_centers để cập nhật các centers mới dựa trên dữ liệu vừa được gán nhãn.
4. has\_converged để kiểm tra điều kiện dừng của thuật toán.

```
def kmeans_init_centers(X, k):
    # randomly pick k rows of X as initial centers
    return X[np.random.choice(X.shape[0], k, replace=False)]

def kmeans_assign_labels(X, centers):
    # calculate pairwise distances btw data and centers
    D = cdist(X, centers)
    # return index of the closest center
    return np.argmin(D, axis = 1)

def kmeans_update_centers(X, labels, K):
    centers = np.zeros((K, X.shape[1]))
    for k in range(K):
        # collect all points assigned to the k-th cluster
        Xk = X[labels == k, :]
        # take average
        centers[k,:] = np.mean(Xk, axis = 0)
    return centers

def has_converged(centers, new_centers):
    # return True if two sets of centers are the same
    return (set([tuple(a) for a in centers]) ==
            set([tuple(a) for a in new_centers]))
```

Phần chính của K-means clustering:

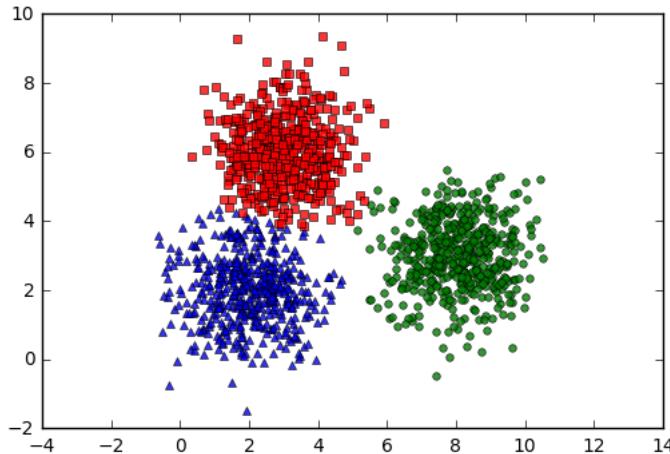
```
def kmeans(X, K):
    centers = [kmeans_init_centers(X, K)]
    labels = []
    it = 0
    while True:
        labels.append(kmeans_assign_labels(X, centers[-1]))
        new_centers = kmeans_update_centers(X, labels[-1], K)
        if has_converged(centers[-1], new_centers):
            break
        centers.append(new_centers)
        it += 1
    return (centers, labels, it)
```

Áp dụng thuật toán vừa viết vào dữ liệu ban đầu, hiển thị kết quả cuối cùng.

```
(centers, labels, it) = kmeans(X, K)
print 'Centers found by our algorithm:'
print centers[-1]

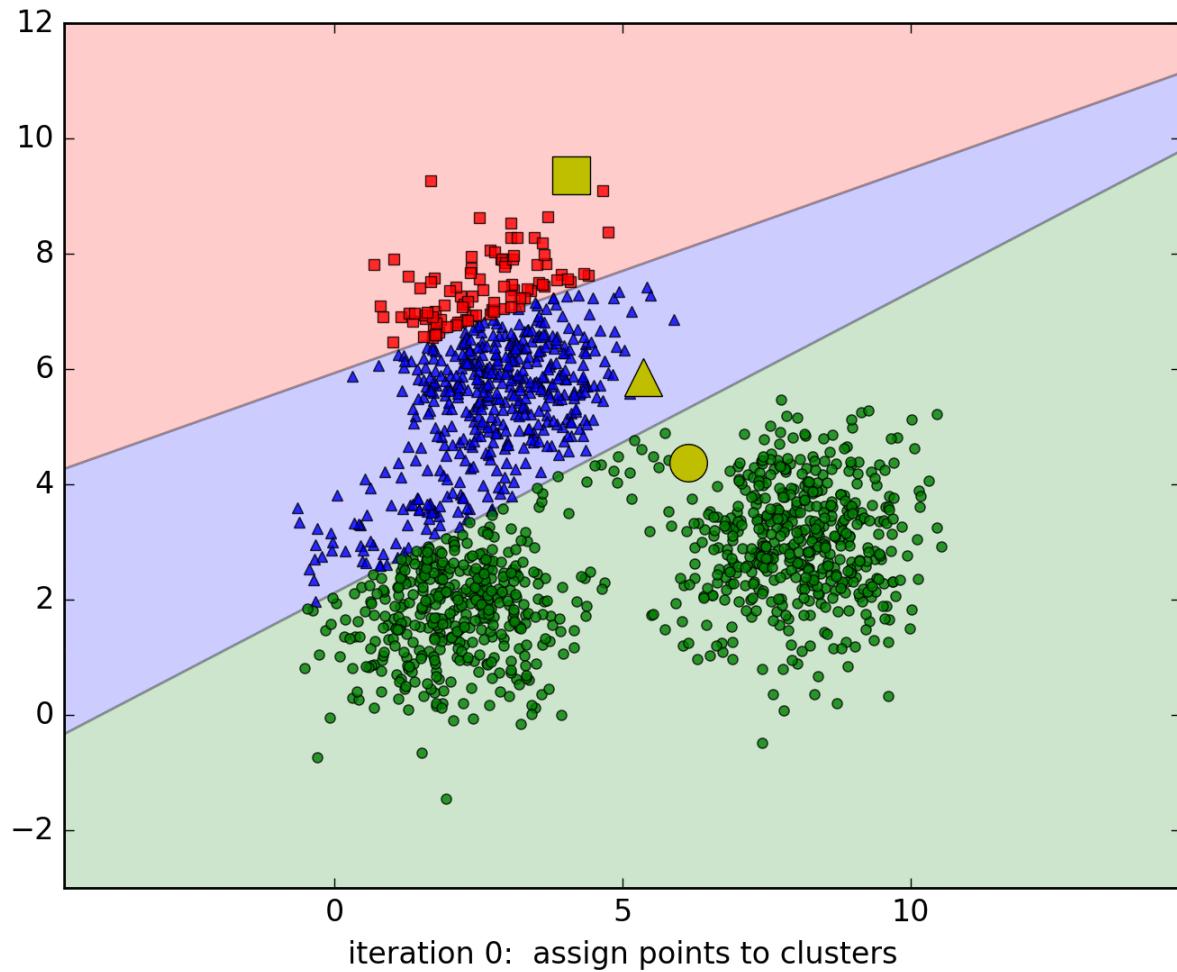
kmeans_display(X, labels[-1])
```

```
Centers found by our algorithm:
[[ 1.97563391  2.01568065]
 [ 8.03643517  3.02468432]
 [ 2.99084705  6.04196062]]
```



Từ kết quả này chúng ta thấy rằng thuật toán K-means clustering làm việc khá thành công, các centers tìm được khá gần với kỳ vọng ban đầu. Các điểm thuộc cùng một cluster hầu như được phân vào cùng một cluster (trừ một số điểm màu đỏ ban đầu đã bị phân nhầm vào cluster màu xanh da trời, nhưng tỉ lệ là nhỏ và có thể chấp nhận được).

Dưới đây là hình ảnh động minh họa thuật toán qua từng vòng lặp, chúng ta thấy rằng thuật toán trên hội tụ rất nhanh, chỉ cần 6 vòng lặp để có được kết quả cuối cùng:



Các bạn có thể xem thêm các trang web minh họa thuật toán K-means cluster tại:

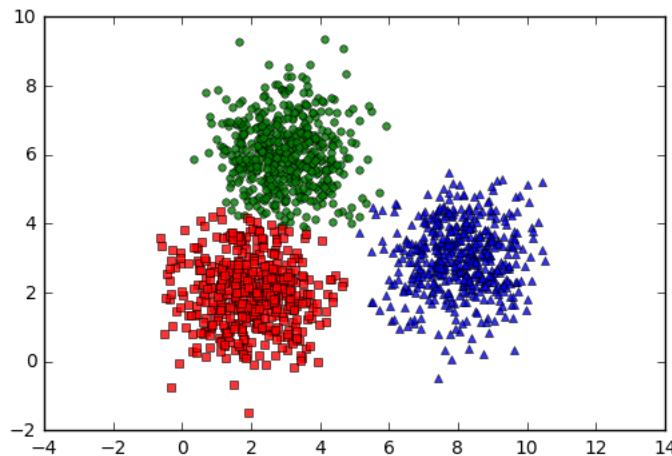
1. Visualizing K-Means Clustering (<https://www.naftaliharris.com/blog/visualizing-k-means-clustering/>)
2. Visualizing K-Means Clustering - Standford (<http://stanford.edu/class/ee103/visualizations/kmeans/kmeans.html>)

### Kết quả tìm được bằng thư viện scikit-learn

Để kiểm tra thêm, chúng ta hãy so sánh kết quả trên với kết quả thu được bằng cách sử dụng thư viện scikit-learn (<http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>).

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
print 'Centers found by scikit-learn:'
print kmeans.cluster_centers_
pred_label = kmeans.predict(X)
kmeans_display(X, pred_label)
```

```
Centers found by scikit-learn:
[[ 8.0410628  3.02094748]
 [ 2.99357611  6.03605255]
 [ 1.97634981  2.01123694]]
```



Thật may mắn (*cho tôi*), hai thuật toán cho cùng một đáp số! Với cách thứ nhất, tôi mong muốn các bạn hiểu rõ được thuật toán K-means clustering làm việc như thế nào. Với cách thứ hai, tôi hy vọng các bạn biết áp dụng thư viện sẵn có như thế nào.

## 4. Thảo luận

### Hạn chế

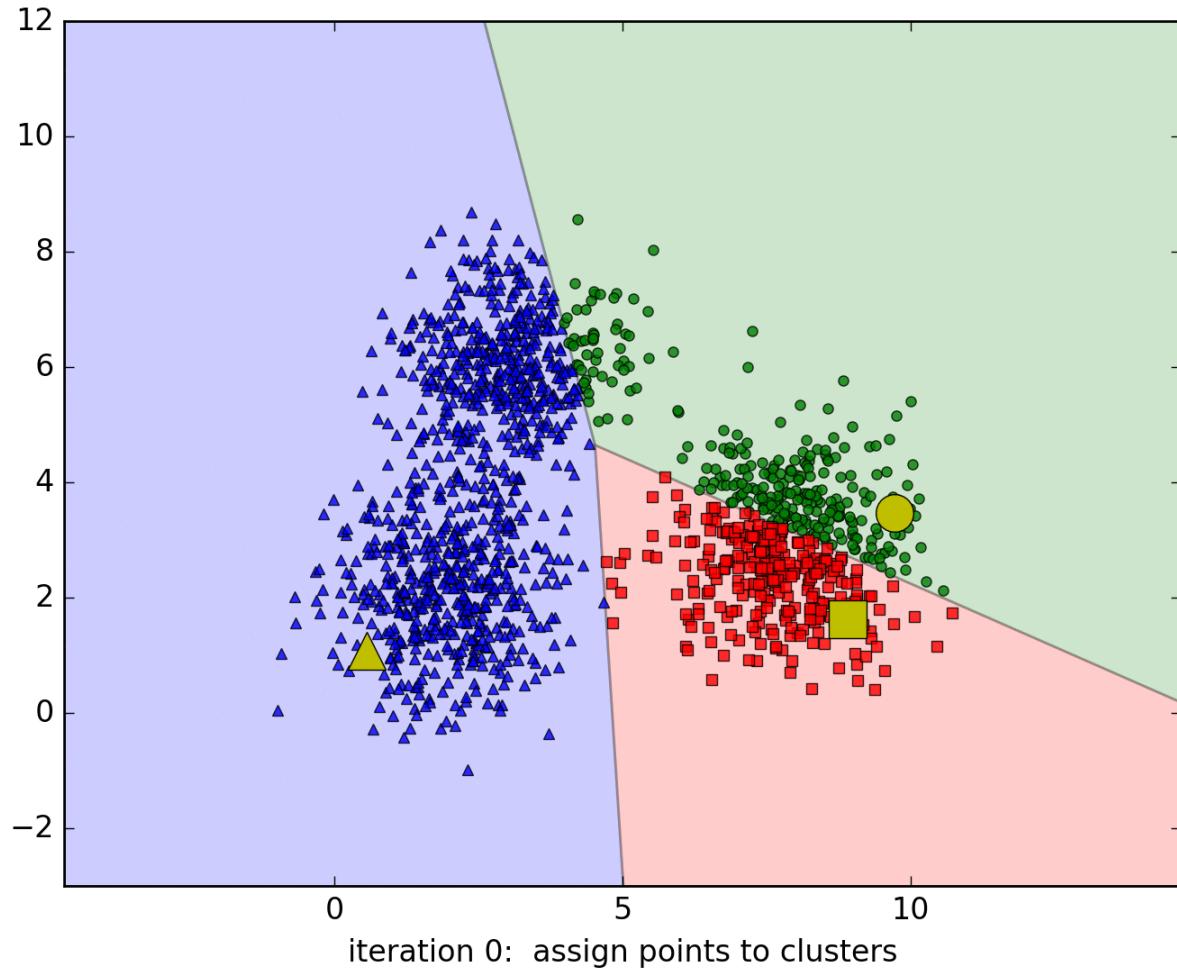
Có một vài hạn chế của thuật toán K-means clustering:

#### Chúng ta cần biết số lượng cluster cần clustering

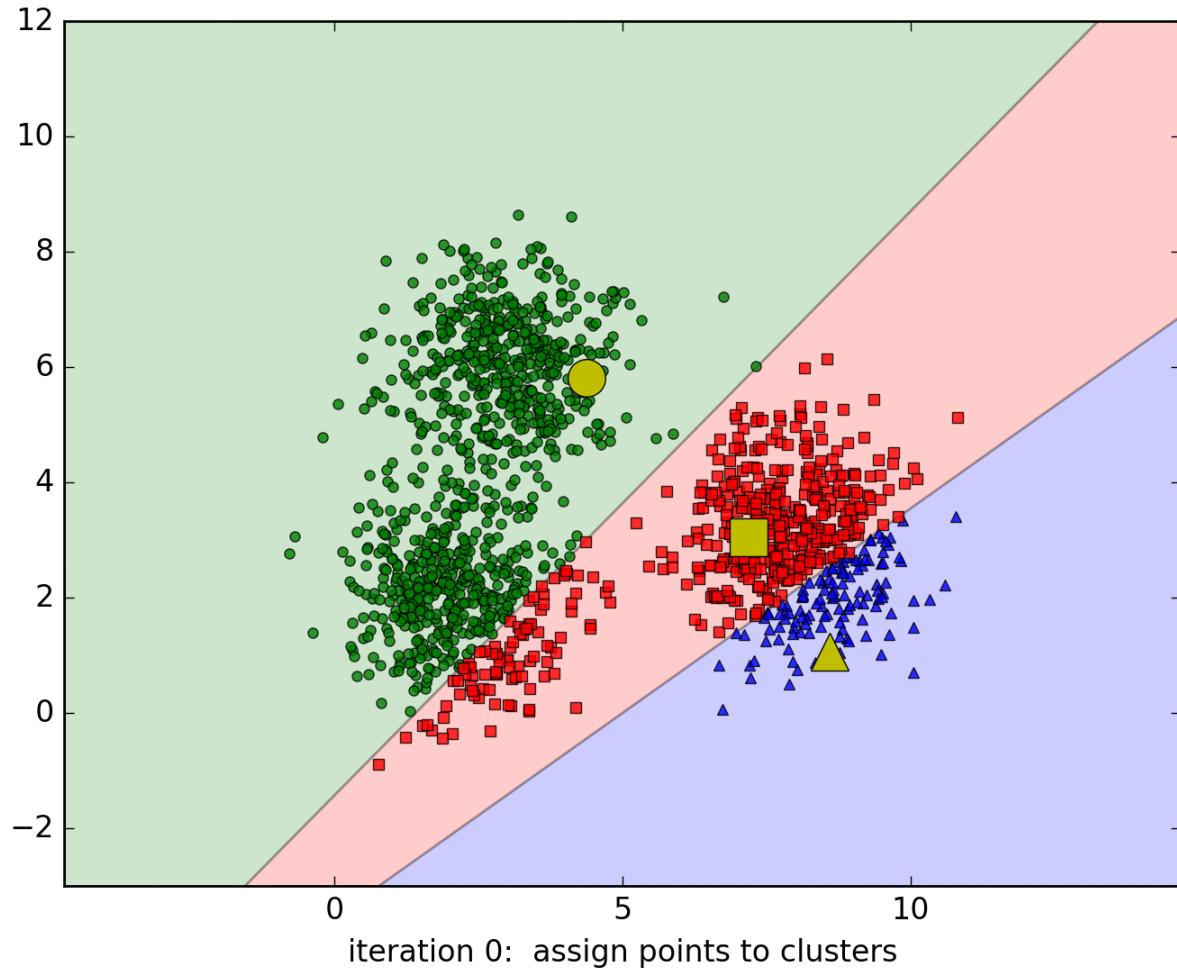
Để ý thấy rằng trong thuật toán nêu trên, chúng ta cần biết đại lượng  $K$  là số lượng clusters. Trong thực tế, nhiều trường hợp chúng ta không xác định được giá trị này. Có một số phương pháp giúp xác định số lượng clusters, tôi sẽ dành thời gian nói về chúng sau nếu có dịp. Bạn đọc có thể tham khảo Elbow method - Determining the number of clusters in a data set ([https://en.wikipedia.org/wiki/Determining\\_the\\_number\\_of\\_clusters\\_in\\_a\\_data\\_set](https://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set)).

#### Nghiệm cuối cùng phụ thuộc vào các centers được khởi tạo ban đầu

Tùy vào các center ban đầu mà thuật toán có thể có tốc độ hội tụ rất chậm, ví dụ:



hoặc thậm chí cho chúng ta nghiệm không chính xác (chỉ là local minimum - điểm cực tiểu - mà không phải giá trị nhỏ nhất):

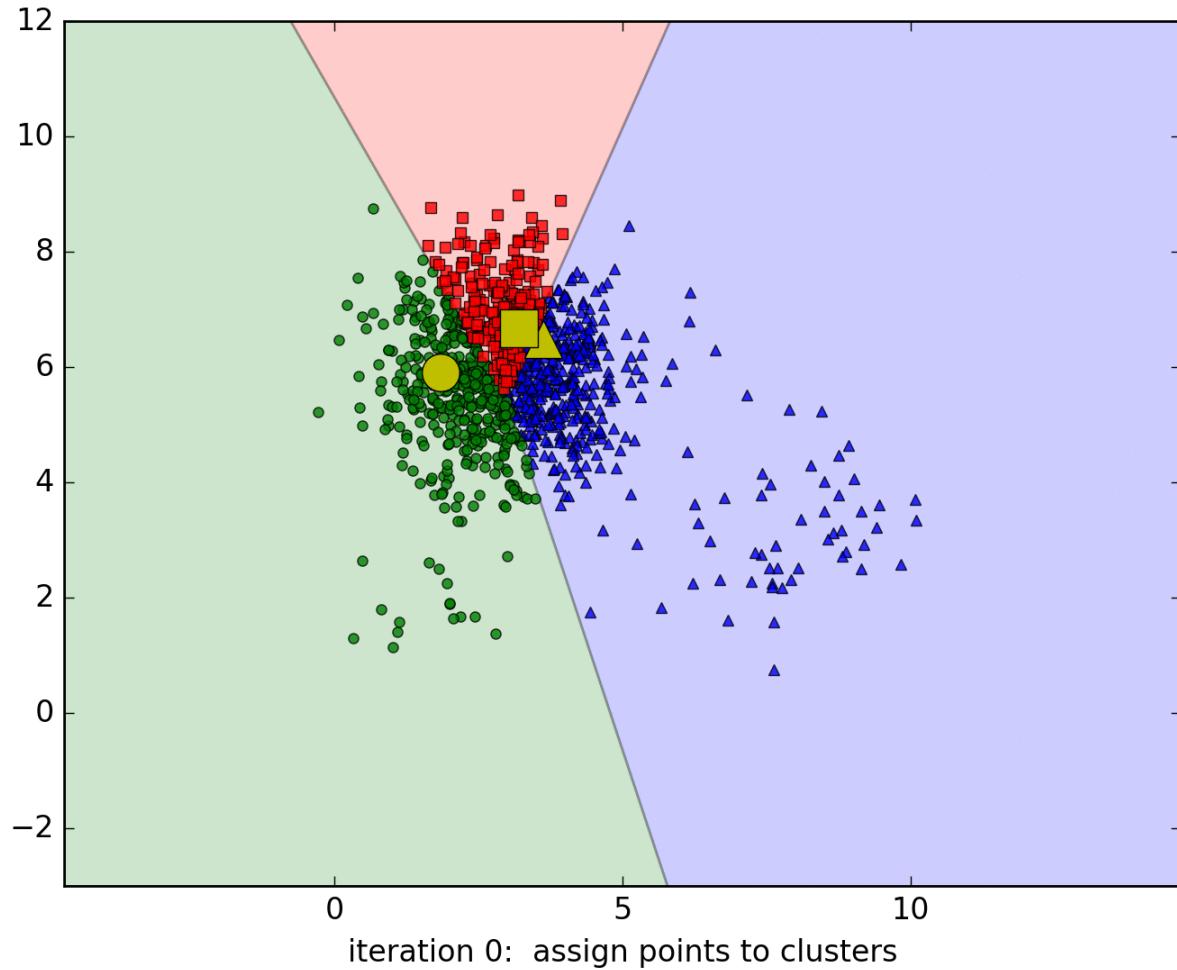


Có một vài cách khắc phục đó là:

- Chạy K-means clustering nhiều lần với các center ban đầu khác nhau rồi chọn cách có hàm mất mát cuối cùng đạt giá trị nhỏ nhất.
- K-means++ - Improve initialization algorithm - wiki ([https://en.wikipedia.org/wiki/K-means%2B%2B#Improved\\_INITIALIZATION\\_ALGORITHM](https://en.wikipedia.org/wiki/K-means%2B%2B#Improved_INITIALIZATION_ALGORITHM)).
- Bạn nào muốn tìm hiểu sâu hơn có thể xem bài báo khoa học Cluster center initialization algorithm for K-means clustering (<http://www.sciencedirect.com/science/article/pii/S0167865504000996>).

Các cluster cần có số lượng điểm gần bằng nhau

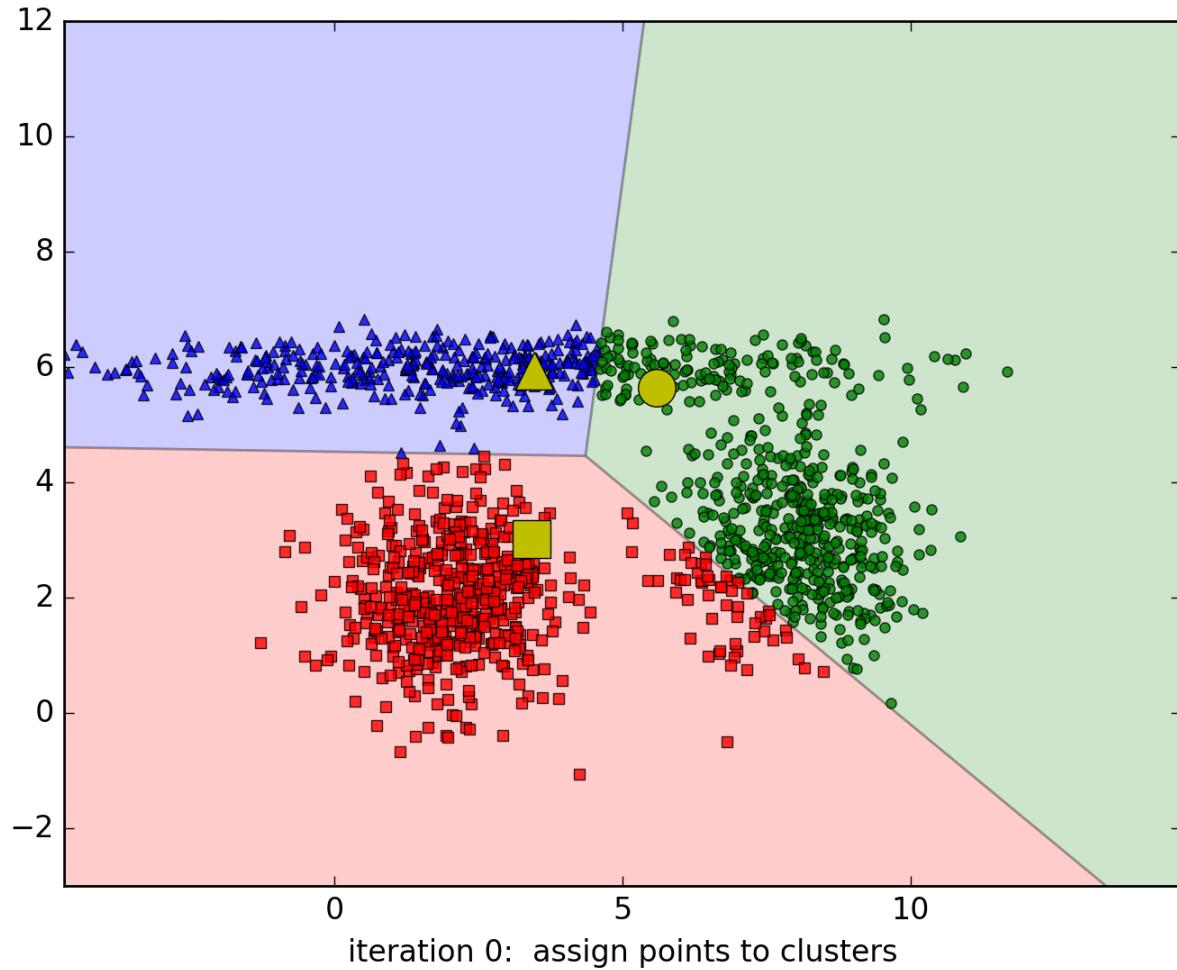
Dưới đây là một ví dụ với 3 cluster với 20, 50, và 1000 điểm. Kết quả cuối cùng không chính xác.



Các cluster cần có dạng hình tròn

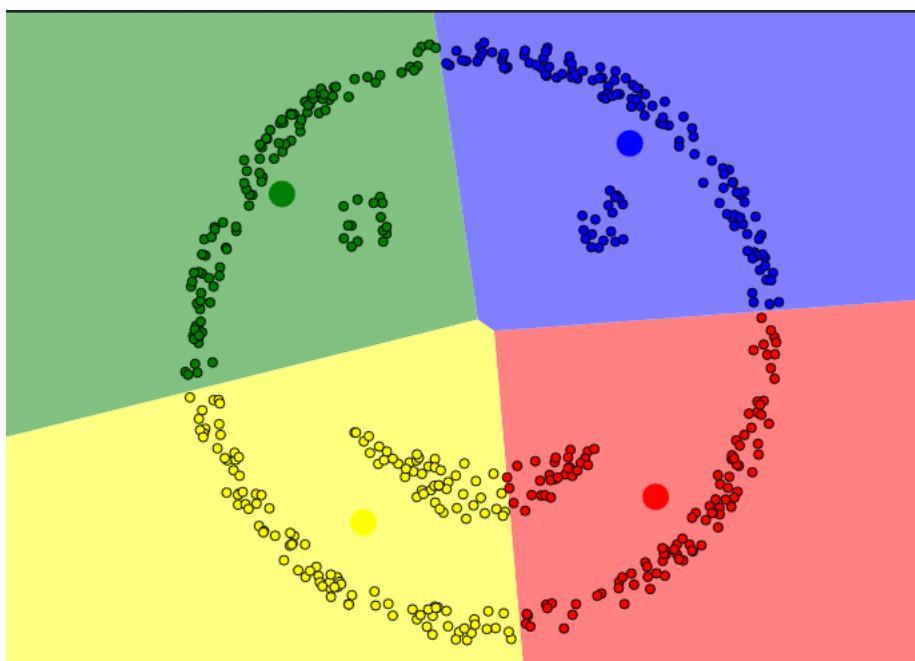
Tức các cluster tuân theo phân phối chuẩn và ma trận hiệp phương sai là ma trận đường chéo có các điểm trên đường chéo giống nhau.

Dưới đây là 1 ví dụ khi 1 cluster có dạng hình dẹt.



Khi một cluster nằm phía trong 1 cluster khác

Đây là ví dụ kinh điển về việc K-means clustering không thể phân cụm dữ liệu. Một cách tự nhiên, chúng ta sẽ phân ra thành 4 cụm: mắt trái, mắt phải, miệng, xung quanh mặt. Nhưng vì mắt và miệng nằm trong khuôn mặt nên K-means clustering không thực hiện được:



Mặc dù có những hạn chế, K-means clustering vẫn cực kỳ quan trọng trong Machine Learning và là nền tảng cho nhiều thuật toán phức tạp sau này. Chúng ta cần bắt đầu từ những thứ đơn giản. *Simple is best!*

## 5. Tài liệu tham khảo

Clustering documents using k-means ([http://scikit-learn.org/stable/auto\\_examples/text/document\\_clustering.html](http://scikit-learn.org/stable/auto_examples/text/document_clustering.html))

Voronoi Diagram - Wikipedia ([https://en.wikipedia.org/wiki/Voronoi\\_diagram](https://en.wikipedia.org/wiki/Voronoi_diagram))

Cluster center initialization algorithm for K-means clustering (<http://www.sciencedirect.com/science/article/pii/S0167865504000996>)

Visualizing K-Means Clustering (<https://www.naftaliharris.com/blog/visualizing-k-means-clustering/>)

Visualizing K-Means Clustering - Standford (<http://stanford.edu/class/ee103/visualizations/kmeans/kmeans.html>)

Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (<https://www.facebook.com/groups/257768141347267/>) để nhận được câu trả lời sớm hơn.

Bạn đọc có thể ủng hộ blog qua 'Buy me a coffee' (/buymeacoffee/) ở góc trên bên trái của blog.

Tôi đang trong quá trình viết cuốn sách 'Machine Learning cơ bản I', các bạn có thể đặt trước tại đây (/ebook/). Cảm ơn bạn.

« Bài 3: Linear Regression (/2016/12/28/linearregression/)

Bài 5: K-means Clustering: Simple Applications » (/2017/01/04/kmeans2/)

43 Comments tiepvu

 Login ▾

 Recommend 7

 Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name

Bùi Quý Bảo • 3 months ago

Anh ơi, anh cho em tài liệu, hay giải thích cụ thể về mặt toán học cho em về cách lấy mẫu được không ạ :

Lấy mẫu theo phân phối chuẩn có kỳ vọng là center của cluster và ma trận hiệp phương sai ?

Lý thuyết xác suất thống kê ở trường em chỉ biết về phân phối chuẩn, hàm mật độ xác suất, và kì vọng của nó. Biết covarian của 2 biến, chứ chưa biết ứng dụng thực tế như thế này ạ ? Anh giải thích cho em với ạ

1 ^ | v • Reply • Share >

Tiep Vu Huu Mod → Bùi Quý Bảo • 3 months ago

Chào em.

Anh đã post câu hỏi của em lên Forum, em lên đây xem mọi người thảo luận nhé:

<https://www.facebook.com/gr...>

^ | v • Reply • Share >

Chu Tiễn Khoa • 4 months ago

Mong anh Tiệp có bài giới thiệu về AffinityPropagation, Mean\_shift trong Unsupervised Learning. Em đọc tài liệu tiếng anh mà chưa hiểu lắm anh à

1 ^ | v • Reply • Share >

Tiep Vu Huu Mod → Chu Tiễn Khoa • 3 months ago

Về sau có thể anh sẽ viết Mean\_shift, còn cái đầu tiên thì anh cũng chưa đọc bao giờ.

^ | v • Reply • Share >

Nguyen Binh • 10 months ago

Em thấy trong bài có phần biểu diễn ma trận center của mỗi cluster là  $M=[m_1, m_2, \dots, m_N]$ , có phải chỗ đó nên ghi là  $M=[m_1, m_2, \dots, m_K]$  đúng ko ạ ?

1 ^ | v • Reply • Share >

Tiep Vu Huu Mod → Nguyen Binh • 10 months ago

Cảm ơn em.

Đúng rồi, anh đã sửa lại chỗ đó.

^ | v • Reply • Share >

Cuong Pham • 10 months ago

Thank a nhiều =D, Bài viết rất chi tiết và bổ ích với những người mới tìm hiểu về machine learning như em. Anh cho em hỏi chút ạ K-Mean cluster đòi hỏi các tập dữ liệu gần bằng nhau. Nếu trong các nhà mạng Viễn thông, em phân nhóm tập các khách hàng, như khách hàng cao cấp, khách hàng truyền

thông, khách hàng năng lực tiêu dùng thấp.... Nếu các nhóm này tỉ lệ khác nhau khá nhiều như 11%, 20%, 31%, 8%, 9%... thì kết quả có chính xác không.

[1 ^](#) | [v](#) • Reply • Share >

**Tiep Vu Huu** Mod → Cuong Pham • 10 months ago

Câu hỏi hay lắm.

Như em đọc trong bài thi kết quả của K-means clustering còn phụ thuộc vào nhiều yếu tố khác ngoài việc các nhóm cần có số dữ liệu gần bằng nhau. Như phân phối của từng nhóm chẳng hạn. K-means clustering yêu cầu chất về dữ liệu nhưng vẫn là thuật toán quan trọng, nó là các bước trung gian đơn giản mà hiệu quả trong nhiều thuật toán nâng cao sau này.

Nếu mới chỉ dùng K-means clustering mà đã có kết quả tốt, anh có thể dùng viết blog ở đây :)

Hẹn gặp em ở các bài Clustering sau, anh chưa biết là khi nào.

[^](#) | [v](#) • Reply • Share >

**Cuong Pham** → Tiep Vu Huu • 10 months ago

Em hiểu rồi à :) , mong sớm được đọc các bài tiếp theo của anh.

[^](#) | [v](#) • Reply • Share >

**Tiep Vu Huu** Mod → Cuong Pham • 10 months ago

Hãy giới thiệu với bạn bè để blog của anh đến được với nhiều người hơn nhé. Cảm ơn em.

[^](#) | [v](#) • Reply • Share >

**Cu Nguyen** • 10 months ago

Good Expression!

[1 ^](#) | [v](#) • Reply • Share >

**Nguyễn Thanh Hưng** • a month ago

Anh ơi!

Em là một sinh viên đang bắt đầu tìm hiểu về machine learning nhưng những kiến thức machine learning liên quan khá nhiều về toán học, phần MATH của anh thì vẫn không đủ, anh có thể gợi ý cho em về cách học cũng như là tài liệu về những kiến thức toán học mà anh bám sát trong suốt những bài viết của anh được không!

em cảm ơn ạ

[^](#) | [v](#) • Reply • Share >

**Nguyễn Việt Khôi** • 2 months ago

Anh cho em hỏi là R dxN có nghĩa là gì vậy ạ...

[^](#) | [v](#) • Reply • Share >

**Tiep Vu Huu** Mod → Nguyễn Việt Khôi • 2 months ago

Là tập hợp các ma trận có d hàng, N cột và các phần tử là số thực.

[^](#) | [v](#) • Reply • Share >

**Nguyễn Việt Khôi** → Tiep Vu Huu • 2 months ago

Cảm ơn anh, cho em hỏi thêm là hiện giờ em đang cảm thấy rất hứng thú với ML, em đã chủ động học python, xác xuất thống kê, đại số tuyến tính nhưng đọc bài giảng của anh em có hơi mơ hồ về một số phần, anh có cách nào để tự học tốt được không ạ.

[^](#) | [v](#) • Reply • Share >

**Hung Nguyen** • 2 months ago

Theo em hiểu là các Center m1, m2, ... mk thuộc vào  $R^{dxk}$ . Sao ở đây a có ghi là thuộc vào  $R^{dx1}$ . A có thể giải thích cho em được không ạ.



[^](#) | [v](#) • Reply • Share >

**Tiep Vu Huu** Mod → Hung Nguyen • 2 months ago

Mỗi center m\_1, ..., m\_K là 1 điểm dữ liệu, tức một vector, nên nó chỉ thuộc  $R^{\{dx1\}}$  được thôi.

[^](#) | [v](#) • Reply • Share >

**Hung Nguyen** → Tiep Vu Huu • 2 months ago

Ok anh em nhìn nhầm. Em tưởng là ma trận các center M thì sẽ có k cột. Ý a ở đây chỉ là 1 điểm m

[^](#) | [v](#) • Reply • Share >

**dung** • 2 months ago

có j a gửi thông tin qua mail cho e nha. minhdung1992@gmail.com

[^](#) | [v](#) • Reply • Share >

**Tiep Vu Huu** Mod → dũng • 2 months ago

Bạn vào Forum hỏi sẽ có người trả lời cho bạn.

^ | v • Reply • Share ›

dũng • 2 months ago

a ui e có ví dụ này. a jup e với ạ

Degree caprice topic lmt lpss pb

1 medium middle tourism yes yes yes

2 high left political yes yes yes

3 medium middle news yes yes yes

4 medium middle news yes yes yes

5 high left political yes yes yes

6 high right political yes no yes

7 high right political yes no no

8 medium right tourism yes no yes

9 medium right tourism yes yes yes

10 medium left news yes no yes

11 high left impression yes yes no

12 low right news no yes no

13 high left political yes yes yes

14 medium left impression yes yes yes

2) Thực hiện theo từng bước thuật toán kmean. Chọn 3 thuộc tính bất kỳ, 6 bản ghi, k=2

a xem hướng dẫn giúp e với ạ.

^ | v • Reply • Share ›

**Tiep Vu Huu** Mod → dũng • 2 months ago

Em đưa câu hỏi lên Forum nhé. Cảm ơn em.

<https://www.facebook.com/gr...>

^ | v • Reply • Share ›

**Trần Văn Dем** • 2 months ago

anh oi anh có thể viết về Latent Dirichlet allocation không ạ.

^ | v • Reply • Share ›

**Darwin** • 5 months ago

Anh oi trang này nhiều công thức bị Math Processing Error.Anh có thể sửa lại không ạ ?

^ | v • Reply • Share ›

**Tiep Vu Huu** Mod → Darwin • 4 months ago

Bạn refresh lại xem. Trên máy tôi chúng vẫn hoạt động bình thường.

^ | v • Reply • Share ›

**Xuan Tran** • 5 months ago

cho em hỏi với ạ, e cài các thư viện như numpy khoobg lỗi mà cài thư viện cdist lại không được ạ

^ | v • Reply • Share ›

**Len Vo** → Xuan Tran • 4 months ago

Chị cài spyder (Anaconda) luôn đi! Xài cho Windows tiện lắm!

Trước giờ em xài Pycharm. Và việc import thư viện khó khăn hơn. Nhờ bạn Pham Thanh Chi mà em biết được tool Spyder. Một lần nữa cảm ơn bạn Pham Thanh Chi nếu có thấy qua dòng reply này! :D

Hy vọng giúp được chị!

^ | v • Reply • Share ›

**Tiep Vu Huu** Mod → Xuan Tran • 4 months ago

cidst thì phải phải cài scipy nhé.

^ | v • Reply • Share ›

**Tiep Vu Huu** Mod → Xuan Tran • 5 months ago

Về lỗi cái các thư viện cho Python trên windows, bạn thử đọc phần này xem:

<http://machinelearningcaban...>

^ | v • Reply • Share ›

**Xuan Tran** • 5 months ago

anh oi em viết kmeans của anh vào chạy thử, em mới học python. nó báo lỗi thế này mà em không biết làm gì để sửa nó. lỗi ngay phần khai báo thư viện numpy

trong hàm em cũng gọi các thư viện như sau

## Bài 5: K-means Clustering: Simple Applications

Clustering (/tags#Clustering) Kmeans (/tags#Kmeans) Unsupervised-learning (/tags#Unsupervised-learning) MNIST (/tags#MNIST)

Jan 4, 2017

Trong bài này, tôi sẽ áp dụng thuật toán K-means clustering (/2017/01/01/kmeans/) vào ba bài toán xử lý ảnh thực tế hơn: i) Phân nhóm các chữ số viết tay, ii) Tách vật thể (image segmentation) và iii) Nén ảnh/dữ liệu (image compression). Qua đây, tôi cũng muốn độc giả làm quen với một số kỹ thuật đơn giản trong xử lý hình ảnh - một mảng quan trọng trong Machine Learning. Souce code cho các ví dụ trong trang này có thể được tìm thấy tại đây (<https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/kmeans/Kmeans2.ipynb>).

**Cảnh báo:** bài này không có nhiều toán.

Trong trang này:

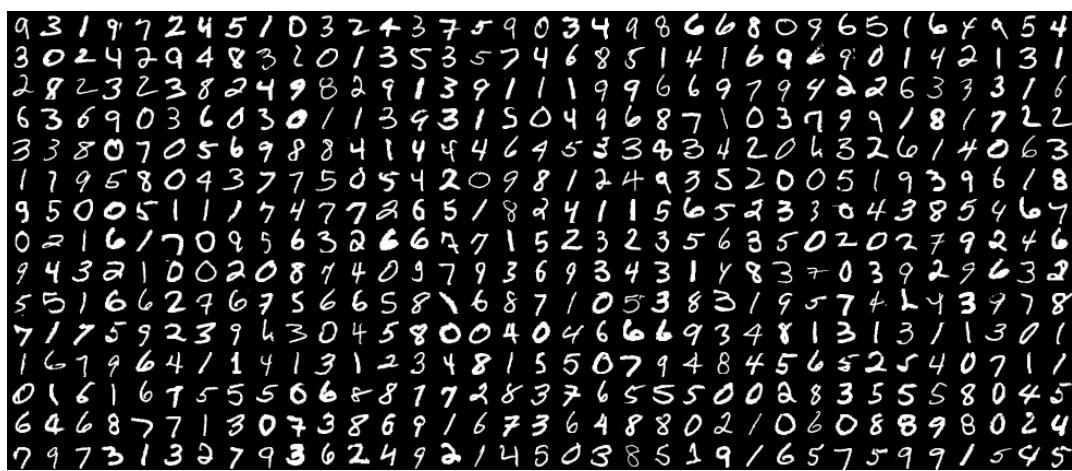
- 1. Phân nhóm chữ số viết tay
  - Bộ cơ sở dữ liệu MNIST
  - Bài toán phân nhóm giả định
  - Làm việc trên Python
- 2. Object Segmentation (tách vật thể trong ảnh)
  - Đặt vấn đề
  - Lên ý tưởng
  - Làm việc trên Python
- 3. Image Compression (nén ảnh và nén dữ liệu nói chung)
- 4. Thảo luận
- 5. Souce code
- 6. Tài liệu tham khảo

### 1. Phân nhóm chữ số viết tay

#### Bộ cơ sở dữ liệu MNIST

Bộ cơ sở dữ liệu MNIST (<http://yann.lecun.com/exdb/mnist/>) là bộ cơ sở dữ liệu lớn nhất về chữ số viết tay và được sử dụng trong hầu hết các thuật toán nhận dạng hình ảnh (Image Classification).

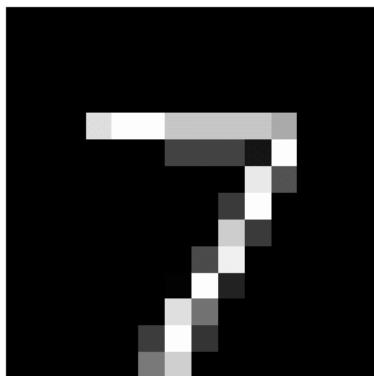
MNIST bao gồm hai tập con: tập dữ liệu huấn luyện (training set) có tổng cộng 60k ví dụ khác nhau về chữ số viết tay từ 0 đến 9, tập dữ liệu kiểm tra (test set) có 10k ví dụ khác nhau. Tất cả đều đã được gán nhãn. Hình dưới đây là ví dụ về một số hình ảnh được trích ra từ MNIST.



MNIST (<http://yann.lecun.com/exdb/mnist/>): bộ cơ sở dữ liệu của chữ số viết tay.

(Nguồn: Simple Neural Network implementation in Ruby) (<http://www.rubylab.io/2015/03/18/simple-neural-network-implementation-in-ruby/>)

Mỗi bức ảnh là một ảnh đen trắng (có 1 channel), có kích thước 28x28 pixel (tổng cộng 784 pixels). Mỗi pixel mang một giá trị là một số tự nhiên từ 0 đến 255. Các pixel màu đen có giá trị bằng 0, các pixel càng trắng thì có giá trị càng cao (nhưng không quá 255). Dưới đây là một ví dụ về chữ số 7 và giá trị các pixel của nó. (Vì mục đích hiển thị ma trận pixel ở bên phải, tôi đã resize bức ảnh về 14x14)



二

## Bài toán phân nhóm giả định

**Bài toán:** Giả sử rằng chúng ta không biết nhãn của các chữ số này, chúng ta muốn phân nhóm các bức ảnh gần giống nhau về một nhóm.

Lại thêm một giả sử nữa là chúng ta mới chỉ biết tới thuật toán phân nhóm K-means clustering (/2017/01/01/kmeans/) gần đây từ blog Machine Learning cơ bản (<https://tiepvpus.edu.vn>) (xin lỗi độc giả vì để các bài học có ý nghĩa hơn, chúng ta đổi khi cần những giả định không được thực tế cho lắm), chúng ta sẽ giải quyết bài toán này thế nào?

Trước khi áp dụng thuật toán K-means clustering (/2017/01/01/kmeans/), chúng ta cần coi mỗi bức ảnh là một điểm dữ liệu. Và vì mỗi điểm dữ liệu là 1 vector (hàng hoặc cột) chứ không phải ma trận như số 7 ở trên, chúng ta phải làm thêm một bước đơn giản trung gian gọi là *vectorization* (vector hóa). Nghĩa là, để có được 1 vector, ta có thể tách các hàng của ma trận pixel ra, sau đó đặt chúng cạnh nhau, và chúng ta được một vector hàng rất dài biểu diễn 1 bức ảnh chữ số.

**Chú ý:** Cách làm này chỉ là cách đơn giản nhất để mô tả dữ liệu ảnh bằng 1 vector. Trên thực tế, người ta áp dụng rất nhiều kỹ thuật khác nhau để có thể tạo ra các vector đặc trưng (feature vector) giúp các thuật toán có được kết quả tốt hơn.

## Làm việc trên Python

Trước tiên các bạn vào trang chủ của MNIST để download (<http://yann.lecun.com/exdb/mnist/>) bộ cơ sở dữ liệu này. Mặc dù trong bài này chúng ta chỉ dùng bộ dữ liệu test với 10k ảnh và không cần label, các bạn vẫn cần download cả hai file `t10k-images-idx3-ubyte.gz` và `t10k-labels-idx1-ubyte.gz` vì thư viện `python-mnist` cần cả hai file này để load dữ liệu từ tập test.

Trước tiên chúng ta cần khai báo một số thư viện:

numpy cho các phép toán liên quan đến ma trận. `mnist` (<https://pypi.python.org/pypi/python-mnist/>) để đọc dữ liệu từ MNIST. `matplotlib` để hiển thị hình vẽ. `sklearn` chính là `scikit-learn` mà chúng ta đã làm quen trong các bài trước. (Về việc cài đặt các thư viện này, tôi hy vọng bạn đọc có thể Google thêm đôi chút. Nếu có khó khăn trong việc cài đặt, hãy để lại comment ở dưới bài. Lưu ý: làm việc trên Windows sẽ khó khăn hơn một chút so với Linux).

```
# %reset
import numpy as np
from mnist import MNIST # require `pip install python-mnist
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

Để hiện thị nhiều bức ảnh các chữ số cùng một lúc, tôi có dùng thêm hàm số `display_network.py` ([https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/kmeans/display\\_network.py](https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/kmeans/display_network.py)).

Thực hiện thuật toán K-means clustering trên toàn bộ 10k chữ số.

```
from display_network import *

mnndata = MNIST('../MNIST/') # path to your MNIST folder
mnndata.load_testing()
X = mnndata.test_images

kmeans = KMeans(n_clusters=K).fit(X)
pred_label = kmeans.predict(X)
```

(Phần còn lại của source code có thể được tìm thấy tại đây (<https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/kmeans/Kmeans2.ipynb>))

Đến đây, sau khi đã tìm được các center và phân nhóm dữ liệu vào từng cluster, tôi muốn hiển thị xem center trông như thế nào và các bức ảnh được phân vào mỗi cluster có giống nhau hay không. Dưới đây là kết quả khi tôi chọn ngẫu nhiên 20 bức ảnh từ mỗi cluster.

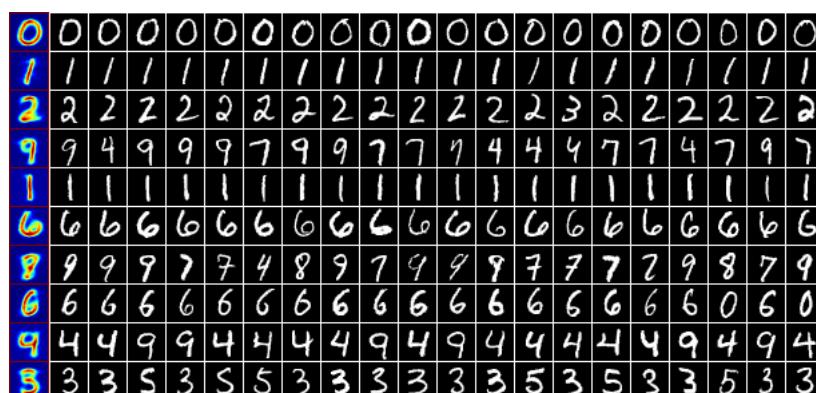


Áp dụng K-means clustering vào tập test set của bộ cơ sở dữ liệu MNIST với  $K = 10$  cluster. Cột 1: centers của các cluster. Các cột còn lại: Mỗi hàng là 20 điểm dữ liệu ngẫu nhiên được chọn ra từ mỗi cluster.

Mỗi hàng tương ứng với một cluster, cột đầu tiên có nền xanh bên trái là centers tìm được của các clusters (màu đỏ hơn là các pixel có giá trị cao hơn). Chúng ta thấy rằng các center đều hoặc là giống với một chữ số nào đó, hoặc là kết hợp của hai/ba chữ số nào đó. Ví dụ: center của nhóm thứ 4 là sự kết hợp của các số 4, 7, 9; của hàng thứ 7 là kết hợp của chữ số 7, 8 và 9.

Tuy nhiên, các bức ảnh lấy ra ngẫu nhiên từ mỗi nhóm trông không thực sự giống nhau. Lý do có thể là những bức ảnh này ở xa các center của mỗi nhóm (mặc dù center đó đã là gần nhất). Như vậy thuật toán K-means clustering làm việc không thực sự tốt trong trường hợp này. (*Thật may là vì thế nên chúng ta vẫn còn nhiều thứ để học nữa.*)

Chúng ta vẫn có thể khai thác một số thông tin hữu ích sau khi thực hiện thuật toán này. Bây giờ, thay vì chọn ngẫu nhiên các bức ảnh trong mỗi cluster, tôi chọn 20 bức ảnh gần center của mỗi cluster nhất, vì càng gần center thì độ tin cậy càng cao. Hãy xem hình dưới đây:



Áp dụng K-means clustering vào tập test set của bộ cơ sở dữ liệu MNIST với  $K = 10$  cluster. Cột 1: centers của các cluster. Các cột còn lại: Mỗi hàng là 20 điểm dữ liệu gần center nhất của mỗi cluster.

Bạn đọc có thể thấy dữ liệu trong mỗi hàng khá giống nhau và giống với center ở cột đầu tiên bên trái. Có một vài quan sát thú vị có thể rút ra từ đây:

- Có hai kiểu viết chữ số 1, một thẳng, một chéo. Và K-means clustering nghĩ rằng đó là hai chữ số khác nhau. Điều này là dễ hiểu vì K-means clustering là thuật toán Unsupervised learning (/2016/12/27/categories/#unsupervised-learning-hoc-khong-giam-sat). Nếu có sự can thiệp của con người, chúng ta có thể nhóm hai clusters này vào làm một.
- Hàng số 9, chữ số 4 và 9 được phân vào cùng 1 cluster. Sự thật là hai chữ số này cũng khá giống nhau. Điều tương tự xảy ra đối với hàng số 7 với các chữ số 7, 8, 9 được xếp vào 1 cluster. Với các cluster này, chúng ta có thể tiếp tục áp dụng K-means clustering để phân nhỏ cluster đó ra.
- Trong clustering có một kỹ thuật thường được sử dụng là Hierarchical clustering (clustering phân tầng) ([https://en.wikipedia.org/wiki/Hierarchical\\_clustering](https://en.wikipedia.org/wiki/Hierarchical_clustering)). Có hai loại Hierarchical clustering:
  - Agglomerative** tức “đi từ dưới lên”. Ban đầu coi mỗi điểm dữ liệu thuộc 1 cluster khác nhau, sau đó các cặp cluster gần giống nhau được gộp lại làm một cluster lớn hơn. Lặp lại quá trình này đến khi nhận được kết quả chấp nhận được.
  - Divisive** tức “đi từ trên xuống”. Ban đầu coi tất cả các điểm dữ liệu thuộc cùng một cluster, sau đó chia nhỏ mỗi cluster bằng một thuật toán clustering nào đó.

## 2. Object Segmentation (tách vật thể trong ảnh)

### Đặt vấn đề

Chúng ta cùng thử áp dụng thuật toán K-means clustering vào một bài toán xử lý ảnh khác: tách vật thể.

Giả sử chúng ta có bức ảnh dưới đây và muốn một thuật toán tự động nhận ra vùng khuôn mặt và tách nó ra.



Credit ảnh: Trọng Vũ (<https://www.facebook.com/photo.php?fbid=1219980151402370&set=a.113129725420757.13101.100001711890571&type=3&theater>)

## Lên ý tưởng

(Lại giả sử rằng chúng ta chưa biết gì khác ngoài K-means clustering, các bạn hãy dùng vài giây để nghĩ xem chúng ta có thể xử lý thế nào. Gợi ý: Có ba màu chủ đạo trong bức ảnh.)

Ok, có ba màu, ba clusters!

Bức ảnh có ba màu chủ đạo: hồng ở khăn và môi; đen ở mắt, tóc, và hậu cảnh; màu da ở vùng còn lại của khuôn mặt. Vậy chúng ta có thể áp dụng thuật toán K-means clustering để phân các pixel ảnh thành 3 clusters, sau đó chọn cluster chứa phần khuôn mặt (phần này do con người làm).

Đây là một bức ảnh màu, mỗi điểm ảnh sẽ được biểu diễn bởi 3 giá trị tương ứng với màu Red, Green, và Blue (mỗi giá trị này cũng là một số tự nhiên không vượt quá 255). Nếu ta coi mỗi điểm dữ liệu là một vector 3 chiều chứa các giá trị này, sau đó áp dụng thuật toán K-means clustering, chúng ta có thể có kết quả mong muốn. Hãy thử xem

## Làm việc trên Python

Khai báo thư viện và load bức ảnh:

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans

img = mpimg.imread('girl13.jpg')
plt.imshow(img)
imgplot = plt.imshow(img)
plt.axis('off')
plt.show()
```

**Biến đổi bức ảnh thành 1 ma trận mà mỗi hàng là 1 pixel với 3 giá trị màu**

```
X = img.reshape((img.shape[0]*img.shape[1], img.shape[2]))
```

(Phần còn lại của source code có thể xem tại đây (<https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/kmeans/Kmeans2.ipynb>)).

Sau khi tìm được các cluster, tôi thay giá trị của mỗi pixel bằng center của cluster chứa nó, và được kết quả như sau:



Ba màu: Hồng, đen, và màu da đã được phân nhóm. Và khuôn mặt có thể được tách ra từ phần có màu da (và vùng bên trong nó). Vậy là K-means clustering tạo ra một kết quả chấp nhận được.

### 3. Image Compression (nén ảnh và nén dữ liệu nói chung)

Để ý thấy rằng mỗi pixel có thể nhận một trong số  $256^3 = 16,777,216$  (16 triệu màu mà chúng ta vẫn nghe khi quảng cáo màn hình). Đây là một số rất lớn (tương đương với 24 bit cho một điểm ảnh). Nếu ta muốn lưu mỗi điểm ảnh với một số bit nhỏ hơn và chấp nhận mất dữ liệu ở một mức nào đó, có cách nào không nếu ta chỉ biết K-means clustering?

Câu trả lời là có. Trong bài toán Segmentation phía trên, chúng ta có 3 clusters, và mỗi một điểm ảnh sau khi xử lý sẽ được biểu diễn bởi 1 số tương ứng với 1 cluster. Tuy nhiên, chất lượng bức ảnh rõ ràng đã giảm đi nhiều. Tôi làm một thí nghiệm nhỏ với số lượng clusters được tăng lên là 5, 10, 15, 20. Sau khi tìm được centers cho mỗi cluster, tôi thay giá trị của một điểm ảnh bằng giá trị của center tương ứng:

```
for K in [5, 10, 15, 20]:
    kmeans = KMeans(n_clusters=K).fit(X)
    label = kmeans.predict(X)

    img4 = np.zeros_like(X)
    # replace each pixel by its center
    for k in range(K):
        img4[label == k] = kmeans.cluster_centers_[k]
    # reshape and display output image
    img5 = img4.reshape((img.shape[0], img.shape[1], img.shape[2]))
    plt.imshow(img5, interpolation='nearest')
    plt.axis('off')
    plt.show()
```

Kết quả:



$K = 5$



$K = 10$



$K = 15$



$K = 20$

Bạn đọc có thể quan sát là khi số lượng clusters tăng lên, chất lượng bức ảnh đã được cải thiện. Đồng thời, chúng ta chỉ cần lưu các centers và label của mỗi điểm ảnh là đã có được một bức ảnh nén (có mất dữ liệu).

### 4. Thảo luận

1. Với một thuật toán K-means clustering đơn giản, chúng ta đã có thể áp dụng nó vào các bài toán thực tế. Mặc dù kết quả chưa thực sự như ý, chúng ta vẫn thấy được tiềm năng của thuật toán đơn giản này.

2. Cách thay một điểm dữ liệu bằng center tương ứng là một trong số các kỹ thuật có tên chung là Vector Quantization (VQ) ([https://en.wikipedia.org/wiki/Vector\\_quantization](https://en.wikipedia.org/wiki/Vector_quantization)). Không chỉ trong nén dữ liệu, VQ còn được áp dụng rộng rãi trong các thuật toán tạo Feature Vector cho các bài toán Phân loại (classification).
3. Các thuật toán trong bài toán này được áp dụng cho xử lý ảnh vì việc này giúp tối dễ dàng minh họa kết quả hơn. Các kỹ thuật này hoàn toàn có thể áp dụng cho các loại cơ sở dữ liệu khác.

## 5. Souce code

Các bạn có thể download source code ở đây (<https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/kmeans/Kmeans2.ipynb>).

## 6. Tài liệu tham khảo

Pattern Recognition and Machine Learning - Christopher Bishop (<http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf>)

Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (<https://www.facebook.com/groups/257768141347267/>) để nhận được câu trả lời sớm hơn.

Bạn đọc có thể ủng hộ blog qua 'Buy me a coffee' (/buymeacoffee/) ở góc trên bên trái của blog.

Tôi đang trong quá trình viết cuốn sách 'Machine Learning cơ bản I', các bạn có thể đặt trước tại đây (/ebook). Cảm ơn bạn.

« Bài 4: K-means Clustering (/2017/01/01/kmeans/)

Bài 6: K-nearest neighbors » (/2017/01/08/knn/)

36 Comments tiepvu

 Login ▾

 Recommend 8  Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name



L\_K • 2 months ago

Anh cho em hỏi với ạ, em có 1 folder ảnh, em muốn áp dụng phương pháp phân nhóm dữ liệu như với chữ viết tay. Vấn đề là em chưa biết import folder đó vào như nào để có thể sử dụng được phương pháp như trong bài viết ạ.

Cảm ơn anh

1 ^ | v • Reply • Share >



Tiep Vu Huu Mod → L\_K • 2 months ago

Em đưa câu hỏi lên Forum nhé. Anh và mọi người sẽ trả lời trên đó để nhiều người thấy. Cảm ơn em.

^ | v • Reply • Share >



Nam Nguyễn • 3 months ago

Ad cho mình hỏi, ngoài bộ chữ số viết tay MNIST thì có bộ ngoài chữ viết tay ko nhỉ? Minh đang muốn nhận dạng cả chữ không phải là chỉ là chữ số và đặc biệt là có cách nào tăng độ chính xác về address hoặc email ... Mong Ad tư vấn.

1 ^ | v • Reply • Share >



Tiep Vu Huu Mod → Nam Nguyễn • 3 months ago

Bạn thử bộ này xem:

<https://www.nist.gov/itl/ia...>

Ngoài ra, bạn có thể lên Forum hỏi thêm các bạn khác. Minh không làm về phần này nhiều nên cũng không rõ lắm.

^ | v • Reply • Share >



thanh hùng trần • 3 months ago

Tại sao trong package mnist không có module MNIST vậy mọi người? Minh import nhưng không được

1 ^ | v • Reply • Share >



Tiep Vu Huu Mod → thanh hùng trần • 3 months ago

Em đã thử `pip install python-mnist` chưa?

^ | v • Reply • Share >



**thanh hùng trần** → Tiep Vu Huu • 3 months ago

Chào anh! Em đã download package mnist bằng pip và có thể import nó. Nhưng không tìm thấy module MNIST trong đó. Em dùng python 3.5

^ | v · Reply · Share >



**Tiep Vu Huu** Mod → thanh hùng trần • 3 months ago

Có lẽ python 3.5 có thay đổi một chút. Em thử tìm các cách khác để đọc MNIST xem.

^ | v · Reply · Share >



**Van Anh Nguyen** • 2 months ago

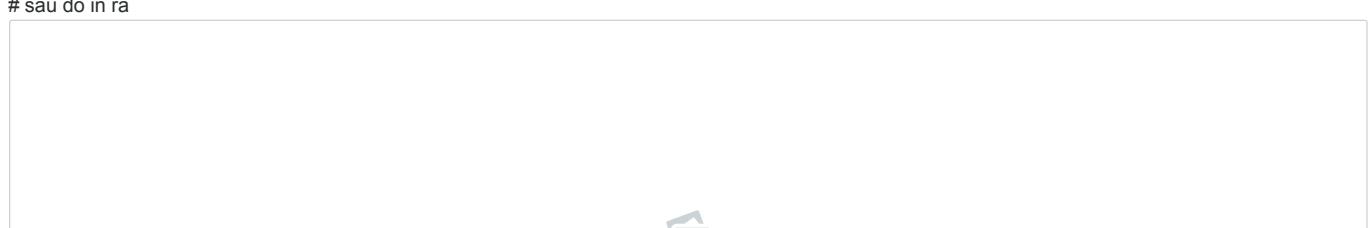
Chào anh. Em mới học python và chưa thành thạo lắm với các thư viện.

Em cho in ra 20 bức ảnh đầu tiên có nhãn là 2:

```
B = np.asarray(X)[pred_label == 2]
H = np.asarray(X)[:20,:]
B = H # ma trận của 20 bức ảnh có label = 2

C = np.asarray(kmeans.cluster_centers_)[2:3,:]
B = np.concatenate((B, C), axis=0) # Thêm vào ma trận B bức ảnh center của cluster có label =2

# sau đó in ra
```



see more

^ | v · Reply · Share >



**Tiệp Hoàng** • 4 months ago

Em chào anh và mọi người.

Em có thắc mắc là phần vectorization mỗi bức ảnh thành một điểm dữ liệu ta sẽ thực hiện như thế nào à? Mong mọi người giải đáp giúp em! Em cảm ơn!

^ | v · Reply · Share >



**Tiep Vu Huu** Mod → Tiệp Hoàng • 4 months ago

Mỗi bức ảnh (xám) được coi như 1 ma trận. Để vectorize nó thì ta chỉ cần đặt tất cả các cột của ma trận chồng lên nhau, như thế sẽ được 1 vector cột rất dài. Mỗi vector dài này là 1 điểm dữ liệu.

1 ^ | v · Reply · Share >



**Darwin** • 4 months ago

X = img.reshape((img.shape[0]\*img.shape[1], img.shape[2]))

Anh ơi a có thể giải thích cho em cái ma trận này như thế nào ko a ?

^ | v · Reply · Share >



**Tiep Vu Huu** Mod → Darwin • 4 months ago

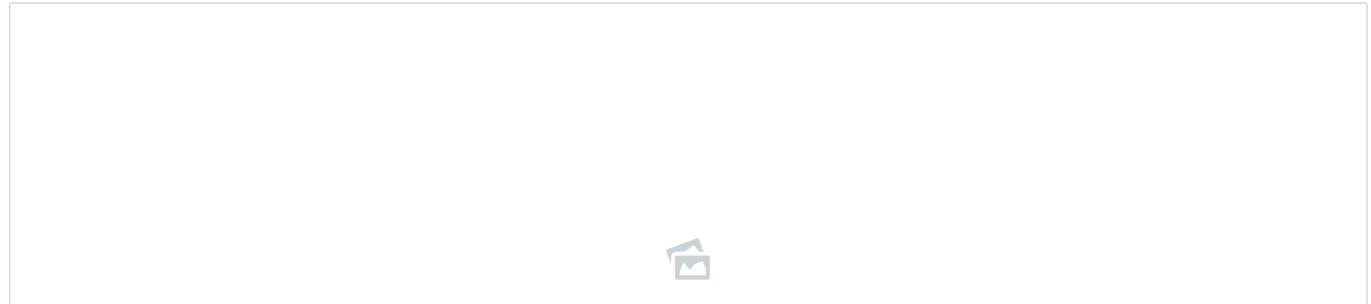
img ban đầu là một bức ảnh có kích thước là  $h \times w \times c$  trong đó  $h$  là chiều cao,  $w$  là chiều rộng,  $c$  là số channels (ở đây là 3 channels RGB). Nhưng vì kmeans làm việc với dữ liệu ở dạng ma trận mà mỗi hàng là 1 điểm dữ liệu. Vậy nên ta phải 'reshape' bức ảnh ở dạng array 3 chiều về dạng array 2 chiều mà mỗi hàng đại diện cho 1 pixel.

^ | v · Reply · Share >



**Nguyễn Khang** • 4 months ago

anh ơi cho em hỏi là, khi em chạy thi chương trình báo lỗi có quá nhiều chỉ số của mảng: too many indices for array tại X0 à,  
Anh giải thích và khắc phục như nào a.



^ | v · Reply · Share >



Tiep Vu Huu Mod → Nguyễn Khang • 4 months ago

Chào bạn, tôi có thử lại code này trên máy tôi thì vẫn chạy bình thường. Bạn thử chạy lại 1 lần nữa xem.

Nếu vẫn không được, bạn có thể làm 'X0 = np.asarray(X)' trước, sau đó 'print X0.shape' để xem kích thước của mảng như thế nào, sau đó cho tôi xem kết quả.

Good luck.

^ | v · Reply · Share >



Darwin • 4 months ago

A ơi em làm tới đoạn:

```
mnndata = MNIST('MNIST/')
```

```
mnndata.load_testing()
```

thì nó lại báo lỗi này hả a ?

ValueError: Magic number mismatch, expected 2049,got 529205248

Mong a giúp đỡ!!

^ | v · Reply · Share >



Thai → Darwin • 4 months ago

bạn giải nén các file .gz tải về chưa?

^ | v · Reply · Share >



Tiep Vu Huu Mod → Darwin • 4 months ago

Có thể tên file hoặc đường dẫn tới file mnist của bạn bị lỗi. Bạn check lại xem.

^ | v · Reply · Share >



Tiep Vu Huu Mod → Darwin • 4 months ago

Mình chưa gặp lỗi này bao giờ. Bạn thử Google xem vẫn đẽ là gì.

Mình search thì ra cái này:

<https://stackoverflow.com/q...>

Theo đây, nếu bạn xoá các file .pyc trong thư mục thì sẽ giải quyết được. Thử xem.

^ | v · Reply · Share >



Stephen Tran • 6 months ago

Anh ơi làm sao đẽ hiển thị bước ảnh như thế này vậy anh

Em làm nhưng nó không ra như vậy :)



^ | v · Reply · Share >



Tiep Vu Huu Mod → Stephen Tran • 5 months ago

À, ảnh này là anh ghép 2 cái lại thôi.

^ | v · Reply · Share >



vanchung1995 • 7 months ago

Ai chỉ cho em cách làm sao đẽ sửa lại lỗi như trên hình không ạ? Em đã edit quyền trong property của file nhưng vẫn không sửa được! Em cảm ơn ạ



^ | v · Reply · Share >



**Phuong Nguyen** → vanchung1995 • 5 months ago

Cái này em delete 2 folder t10k-images-idx3-ubyte và t10k-labels-idx1-ubyte đi và bỏ file extension của 2 file t10k-images.idx3-ubyte và t10k-labels.idx1-ubyte sau đó rename nó thành t10k-images-idx3-ubyte và t10k-labels-idx1-ubyte. Như vậy là chạy ok.

^ | v · Reply · Share >



**DaoTuan** • 9 months ago

Em chạy thử cả trên mac và ubutun đều báo lỗi:

ImportError: No module named mnist

anh có thể xem lại không ạ.

^ | v · Reply · Share >



**Tiep Vu Huu** Mod → DaoTuan • 9 months ago

Python không cung cấp sẵn các package như MATLAB mà ta phải cài đặt thêm khi muốn dùng.

Để có thể cài thêm một package mới, cách đơn giản nhất là dùng pip. Em làm theo hướng dẫn ở đây:

<https://www.rosehosting.com...>

Sau khi có pip, em dùng 'pip install packagename' để install package có tên là 'packagename'. Trong trường hợp này, em cần 'pip install python-mnist':

<https://github.com/sorki/py...>

Một vài package khác cần có: numpy, scipy, matplotlib, sklearn. Em nên tự tìm cách cài đặt rồi chạy các code mẫu.

^ | v · Reply · Share >



**DaoTuan** → Tiep Vu Huu • 9 months ago

Em đã cài "pip install python-mnist" và khắc phục được lỗi trên rồi a. Nhưng khi chạy đến "mnist.load\_testing()" thì lại có lỗi là IOError: [Errno 2] No such file or directory: '../MNIST/t10k-labels-idx1-ubyte'

em đã vào trang chủ của MNIST download t10k-images-idx3-ubyte.gz như anh bảo, và còn download thêm cái t10k-labels-idx1-ubyte.gz dù phòng nhưng vẫn báo lỗi ạ!

^ | v · Reply · Share >



**Tiep Vu Huu** Mod → DaoTuan • 9 months ago

Có thể em quên chưa giải nén.

^ | v · Reply · Share >



**DaoTuan** → Tiep Vu Huu • 9 months ago

em đã sửa nó thành mnist = MNIST('MNIST/') thì chạy oke a. Trong phần còn lại của code có biến "is\_train\_set" trong câu lệnh: "if is\_train\_set:

... kmeans = MiniBatchKMeans(n\_clusters=K, n\_init=30, batch\_size=10000).fit(X)"

python báo lỗi tên a:"NameError: name 'is\_train\_set' is not defined"

anh xem lại giúp em với!!

^ | v · Reply · Share >



**Thanh Nguyen** → DaoTuan • 9 months ago

Chào bạn. cái đoạn MNIST('Mnist') trong ngoặc là bạn đặt cái đường path chứa cái file t10k-images-idx3-ubyte.gz đúng ko? tại sao khi bạn rút ngắn lại còn Mnist thì nó vẫn chạy được nhỉ?

Mình đã thử cả 2 cách. đặt path dẫn tới folder chứa cái file ảnh hoặc làm theo bạn nói nhưng vẫn ko dc. nó đưa lại lỗi giống bạn ở trên là:

IOError: [Errno 2] No such file or directory: '../MNIST/t10k-labels-idx3-ubyte'

Ai có thể giúp mình đoạn này ko nhỉ?

[^](#) | [v](#) • Reply • Share >**Tiep Vu Huu** Mod → Thanh Nguyen • 9 months ago

Bạn thử sửa tên file thành t10k-labels.idx3-ubyte xem. Sự khác nhau là dấu chấm và dấu gạch ngang.

[1 ^](#) | [v](#) • Reply • Share >**DaoTuan** → Thanh Nguyen • 9 months ago

bạn làm 1 cái file đặt tên là MNIST, bên trong thì bạn lưu file t10k-labels-idx3-ubyte.gz đã download về và đừng quên giải nén nó. Đây là cách mình làm, chúc bạn thành công!

!

[^](#) | [v](#) • Reply • Share >**Strike Wade** → DaoTuan • 8 months ago

Chào bạn,

Minh đã làm theo đến bước này nhưng nó vẫn bị lỗi FileNotFoundError như vậy

FileNotFoundException: [Errno 2] No such file or directory: './MNIST/t10k-labels-idx3-ubyte'

Minh đã coi hết comments dưới này và sửa các tên có thể nhưng sao nó đều kiểm ko thấy (t10k-labels.idx3-ubyte, t10k-labels-idx1-ubyte - cái này nó thành file mất định dạng luôn nên mình nghĩ ko phải, t10k-labels-idx1-ubyte.idx3-ubyte, ...) , đã sửa luôn như bạn đã nói ngay line: mndata = MNIST('MINIST') nhưng cũng hỏng được ...

Minh nghĩ đến có thể để sai dir MNIST (có phải để ngay trong folder có file python chạy ko bác? hay dir MNIST này nó để ở chỗ nào trong dir python27 cài packages?)

Thanks bác ...

[^](#) | [v](#) • Reply • Share >**DaoTuan** → Strike Wade • 8 months ago

Bác đã tạo 1 cái file tên MNIST chưa? bác tải cái "t10k-images-idx3-ubyte.gz" như anh Tiệp có cho link trên, lưu vào file MNIST này, sau đó giải nén là chạy được thôi. Bác check lại nhé!

[1 ^](#) | [v](#) • Reply • Share >**Thanh Nguyen** → DaoTuan • 8 months ago

Cảm ơn bạn. mình đã làm đc rồi!

[^](#) | [v](#) • Reply • Share >**Tiep Vu Huu** Mod → DaoTuan • 9 months ago

Anh đã fix lỗi đó. Em xem lại nhé.

[^](#) | [v](#) • Reply • Share >**DaoTuan** → Tiep Vu Huu • 9 months ago

em chạy được rồi, thanks a!

[1 ^](#) | [v](#) • Reply • Share >

## ALSO ON TIEPVU

**ebook Machine Learning cơ bản**

26 comments • 3 months ago

Xuân Xuân — Anh ôi, cái email trường em cấp ý, em test nó không cho các mail ngoài gửi vào:"Hi. This is the qmail-send program at vnu.edu.vn.I'm ...

**Machine Learning cơ bản**

11 comments • 8 months ago

CTVR — Em cũng thấy yêu cầu 64 bit trên win. Chắc em chạy với linux vậy.  
P/s: Giờ em mới đọc xong loạt bài về convex trên file pdf. Hồi mới vào ...[✉ Subscribe](#) [✉ Add Disqus to your site](#)[Add Disqus Add](#) [🔒 Privacy](#)

## Bài 6: K-nearest neighbors

KNN (/tags#KNN) Regression (/tags#Regression) Classification (/tags#Classification) Supervised-learning (/tags#Supervised-learning) MNIST (/tags#MNIST) Iris (/tags#Iris)

Jan 8, 2017

Nếu như con người có kiểu học “nước đến chân mới nhảy”, thì trong Machine Learning cũng có một thuật toán như vậy.

**Trong trang này:**

- 1. Giới thiệu
  - Một câu chuyện vui
  - K-nearest neighbor
  - Khoảng cách trong không gian vector
- 2. Phân tích toán học
- 3. Ví dụ trên Python
  - Bộ cơ sở dữ liệu Iris (Iris flower dataset).
  - Thí nghiệm
    - Tách training và test sets
    - Phương pháp đánh giá (evaluation method)
    - Đánh trọng số cho các điểm lân cận
- 4. Thảo luận
  - KNN cho Regression
  - Chuẩn hóa dữ liệu
  - Sử dụng các phép đo khoảng cách khác nhau
  - Ưu điểm của KNN
  - Nhược điểm của KNN
  - Tăng tốc cho KNN
  - Try this yourself
  - Source code
- 5. Tài liệu tham khảo

### 1. Giới thiệu

#### Một câu chuyện vui

Có một anh bạn chuẩn bị đến ngày thi cuối kỳ. Vì môn này được mở tài liệu khi thi nên anh ta không chịu ôn tập để hiểu ý nghĩa của từng bài học và mối liên hệ giữa các bài. Thay vào đó, anh thu thập tất cả các tài liệu trên lớp, bao gồm ghi chép bài giảng (lecture notes), các slides và bài tập về nhà + lời giải. Để cho chắc, anh ta ra thư viện và các quán Photocopy quanh trường mua hết tất cả các loại tài liệu liên quan (*khá khen cho cậu này chịu khó tìm kiếm tài liệu*). Cuối cùng, anh bạn của chúng ta thu thập được một chồng cao tài liệu để mang vào phòng thi.

Vào ngày thi, anh tự tin mang chồng tài liệu vào phòng thi. Aha, đè này ít nhất mình phải được 8 điểm. Câu 1 giống hệt bài giảng trên lớp. Câu 2 giống hệt đè thi năm ngoái mà lời giải có trong tập tài liệu mua ở quán Photocopy. Câu 3 gần giống với bài tập về nhà. Câu 4 trắc nghiệm thậm chí cậu nhớ chính xác ba tài liệu có ghi đáp án. Câu cuối cùng, 1 câu khó nhưng anh đã từng nhìn thấy, chỉ là không nhớ ở đâu thôi.

Kết quả cuối cùng, cậu ta được 4 điểm, vừa đủ điểm qua môn. Cậu làm chính xác câu 1 vì tìm được ngay trong tập ghi chú bài giảng. Câu 2 cũng tìm được đáp án nhưng lời giải của quán Photocopy sai! Câu ba thấy gần giống bài về nhà, chỉ khác mỗi một số thôi, cậu cho kết quả giống như thế luôn, vậy mà không được điểm nào. Câu 4 thì tìm được cả 3 tài liệu nhưng có hai trong đó cho đáp án A, cái còn lại cho B. Cậu chọn A và được điểm. Câu 5 thì không làm được dù còn tới 20 phút, vì tìm mãi chẳng thấy đáp án đâu - nhiều tài liệu quá cung mệt!!

Không phải ngẫu nhiên mà tôi dành ra ba đoạn văn để kể về chuyện học hành của anh chàng kia. Hôm nay tôi xin trình bày về một phương pháp trong Machine Learning, được gọi là K-nearest neighbor (hay KNN), một thuật toán được xếp vào loại lazy (machine) learning (máy lười học). Thuật toán này khá giống với cách học/thi của anh bạn kém may mắn kia.

#### K-nearest neighbor

K-nearest neighbor là một trong những thuật toán supervised-learning đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong Machine Learning. Khi training, thuật toán này *không học* một điều gì từ dữ liệu training (đây cũng là lý do thuật toán này được xếp vào loại lazy learning ([https://en.wikipedia.org/wiki/Lazy\\_learning](https://en.wikipedia.org/wiki/Lazy_learning))), mọi tính toán được thực hiện khi nó cần dự đoán kết quả của dữ liệu mới. K-nearest neighbor có thể áp

dụng được vào cả hai loại của bài toán Supervised learning là Classification (/2016/12/27/categories/#classification-phan-loai) và Regression (/2016/12/27/categories/#regression-hoi-quy). KNN còn được gọi là một thuật toán Instance-based hay Memory-based learning ([https://en.wikipedia.org/wiki/Instance-based\\_learning](https://en.wikipedia.org/wiki/Instance-based_learning)).

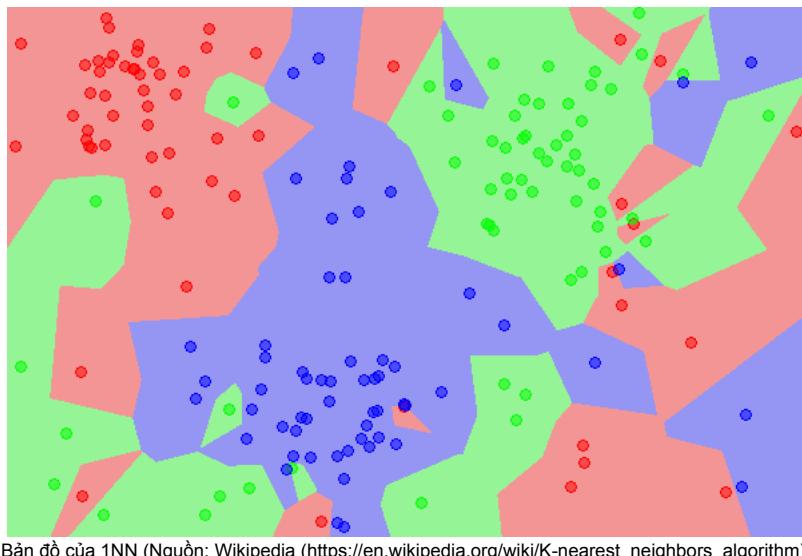
Có một vài khái niệm tương ứng người-máy như sau:

Ngôn ngữ người	Ngôn ngữ Máy Học	in Machine Learning
Câu hỏi	Điểm dữ liệu	Data point
Đáp án	Đầu ra, nhãn	Output, Label
Ôn thi	Huấn luyện	Training
Tập tài liệu mang vào phòng thi	Tập dữ liệu tập huấn	Training set
Đề thi	Tập dữ liệu kiểm thử	Test set
Câu hỏi trong đề thi	Dữ liệu kiểm thử	Test data point
Câu hỏi có đáp án sai	Nhiều	Noise, Outlier
Câu hỏi gần giống	Điểm dữ liệu gần nhất	Nearest Neighbor

Với KNN, trong bài toán Classification, label của một điểm dữ liệu mới (hay kết quả của câu hỏi trong bài thi) được suy ra trực tiếp từ K điểm dữ liệu gần nhất trong training set. Label của một test data có thể được quyết định bằng major voting (bầu chọn theo số phiếu) giữa các điểm gần nhất, hoặc nó có thể được suy ra bằng cách đánh trọng số khác nhau cho mỗi trong các điểm gần nhất đó rồi suy ra label. Chi tiết sẽ được nêu trong phần tiếp theo.

Trong bài toán Regression, đầu ra của một điểm dữ liệu sẽ bằng chính đầu ra của điểm dữ liệu đã biết gần nhất (trong trường hợp K=1), hoặc là trung bình có trọng số của đầu ra của những điểm gần nhất, hoặc bằng một mối quan hệ dựa trên khoảng cách tới các điểm gần nhất đó.

Một cách ngắn gọn, KNN là thuật toán đi tìm đầu ra của một điểm dữ liệu mới bằng cách chỉ dựa trên thông tin của K điểm dữ liệu trong training set gần nó nhất (K-lân cận), *không quan tâm đến việc có một vài điểm dữ liệu trong những điểm gần nhất này là nhiễu*. Hình dưới đây là một ví dụ về KNN trong classification với K = 1.



Bản đồ của 1NN (Nguồn: Wikipedia ([https://en.wikipedia.org/wiki/K-nearest\\_neighbors\\_algorithm](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)))

Ví dụ trên đây là bài toán Classification với 3 classes: Đỏ, Lam, Lục. Mỗi điểm dữ liệu mới (test data point) sẽ được gán label theo màu của điểm mà nó thuộc về. Trong hình này, có một vài vùng nhỏ xem lẩn vào các vùng lớn hơn khác màu. Ví dụ có một điểm màu Lục ở gần góc 11 giờ nằm giữa hai vùng lớn với nhiều dữ liệu màu Đỏ và Lam. Điểm này rất có thể là nhiễu. Dẫn đến nếu dữ liệu test rơi vào vùng này sẽ có nhiều khả năng cho kết quả không chính xác.

### Khoảng cách trong không gian vector

Trong không gian một chiều, khoảng cách giữa hai điểm là trị tuyệt đối giữa hiệu giá trị của hai điểm đó. Trong không gian nhiều chiều, khoảng cách giữa hai điểm có thể được định nghĩa bằng nhiều hàm số khác nhau, trong đó độ dài đường thẳng nối hai điểm chỉ là một trường hợp đặc biệt trong đó. Nhiều thông tin bổ ích (cho Machine Learning) có thể được tìm thấy tại Norms (chuẩn) của vector (/math/#-norms-chuan) trong tab Math (/math/).

## 2. Phân tích toán học

Thuật toán KNN rất dễ hiểu nên sẽ phần “Phân tích toán học” này sẽ chỉ có 3 câu. Tôi trực tiếp đi vào các ví dụ. Có một điều đáng lưu ý là KNN phải nhớ tất cả các điểm dữ liệu training, việc này không được lợi về cả bộ nhớ và thời gian tính toán - giống như khi cậu bạn của chúng ta không tìm được câu trả lời cho câu hỏi cuối cùng.

### 3. Ví dụ trên Python

#### Bộ cơ sở dữ liệu Iris (Iris flower dataset).

Iris flower dataset ([https://en.wikipedia.org/wiki/Iris\\_flower\\_data\\_set](https://en.wikipedia.org/wiki/Iris_flower_data_set)) là một bộ dữ liệu nhỏ (nhỏ hơn rất nhiều so với MNIST (/2017/01/04/kmeans2/#bo-co-so-du-lieu-mnist)). Bộ dữ liệu này bao gồm thông tin của ba loại hoa Iris (một loài hoa lan) khác nhau: Iris setosa, Iris virginica và Iris versicolor. Mỗi loại có 50 bông hoa được đo với dữ liệu là 4 thông tin: chiều dài, chiều rộng đài hoa (sepal), và chiều dài, chiều rộng cánh hoa (petal). Dưới đây là ví dụ về hình ảnh của ba loại hoa. (Chú ý, đây không phải là bộ cơ sở dữ liệu ảnh như MNIST, mỗi điểm dữ liệu trong tập này chỉ là một vector 4 chiều).



Iris setosa



Iris versicolor



Iris virginica

Ví dụ về Iris flower dataset (Nguồn: Wikipedia ([https://en.wikipedia.org/wiki/Iris\\_flower\\_data\\_set](https://en.wikipedia.org/wiki/Iris_flower_data_set)))

Bộ dữ liệu nhỏ này thường được sử dụng trong nhiều thuật toán Machine Learning trong các lớp học. Tôi sẽ giải thích lý do không chọn MNIST vào phần sau.

#### Thí nghiệm

Trong phần này, chúng ta sẽ tách 150 dữ liệu trong Iris flower dataset ra thành 2 phần, gọi là *training set* và *test set*. Thuật toán KNN sẽ dựa vào trông tin ở *training set* để dự đoán xem mỗi dữ liệu trong *test set* tương ứng với loại hoa nào. Dữ liệu được dự đoán này sẽ được đối chiếu với loại hoa thật của mỗi dữ liệu trong *test set* để đánh giá hiệu quả của KNN.

#### Trước tiên, chúng ta cần khai báo vài thư viện.

Iris flower dataset có sẵn trong thư viện scikit-learn (<http://scikit-learn.org/>).

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import neighbors, datasets
```

Tiếp theo, chúng ta load dữ liệu và hiện thị vài dữ liệu mẫu. Các class được gán nhãn là 0, 1, và 2.

```
iris = datasets.load_iris()
iris_X = iris.data
iris_y = iris.target
print 'Number of classes: %d' %len(np.unique(iris_y))
print 'Number of data points: %d' %len(iris_y)

X0 = iris_X[iris_y == 0,:]
print '\nSamples from class 0:\n', X0[:5,:]

X1 = iris_X[iris_y == 1,:]
print '\nSamples from class 1:\n', X1[:5,:]

X2 = iris_X[iris_y == 2,:]
print '\nSamples from class 2:\n', X2[:5,:]
```

```
Number of classes: 3
Number of data points: 150

Samples from class 0:
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.  1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]]
```

```
Samples from class 1:
[[ 7.   3.2  4.7  1.4]
 [ 6.4  3.2  4.5  1.5]
 [ 6.9  3.1  4.9  1.5]
 [ 5.5  2.3  4.   1.3]
 [ 6.5  2.8  4.6  1.5]]
```

```
Samples from class 2:
[[ 6.3  3.3  6.   2.5]
 [ 5.8  2.7  5.1  1.9]
 [ 7.1  3.   5.9  2.1]
 [ 6.3  2.9  5.6  1.8]
 [ 6.5  3.   5.8  2.2]]
```

Nếu nhìn vào vài dữ liệu mẫu, chúng ta thấy rằng hai cột cuối mang khá nhiều thông tin giúp chúng ta có thể phân biệt được chúng. Chúng ta dự đoán rằng kết quả classification cho cơ sở dữ liệu này sẽ tương đối cao.

## Tách training và test sets

Giả sử chúng ta muốn dùng 50 điểm dữ liệu cho test set, 100 điểm còn lại cho training set. Scikit-learn có một hàm số cho phép chúng ta ngẫu nhiên lựa chọn các điểm này, như sau:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_X, iris_y, test_size=50)

print "Training size: %d" %len(y_train)
print "Test size     : %d" %len(y_test)
```

```
Training size: 100
Test size     : 50
```

Sau đây, tôi trước hết xét trường hợp đơn giản K = 1, tức là với mỗi điểm test data, ta chỉ xét 1 điểm training data gần nhất và lấy label của điểm đó để dự đoán cho điểm test này.

```
clf = neighbors.KNeighborsClassifier(n_neighbors = 1, p = 2)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print "Print results for 20 test data points:"
print "Predicted labels: ", y_pred[20:40]
print "Ground truth     : ", y_test[20:40]
```

```
Print results for first 20 test data points:
Predicted labels: [2 1 2 2 1 2 2 0 2 0 2 0 1 0 0 2 2 0 2 0]
Ground truth     : [2 1 2 2 1 2 2 0 2 0 2 0 1 0 0 2 1 0 2 0]
```

Kết quả cho thấy label dự đoán gần giống với label thật của test data, chỉ có 2 điểm trong số 20 điểm được hiển thị có kết quả sai lệch. Ở đây chúng ta làm quen với khái niệm mới: *ground truth*. Một cách đơn giản, *ground truth* chính là nhãn/label/đầu ra *thực sự* của các điểm trong test data. Khái niệm này được dùng nhiều trong Machine Learning, hy vọng lần tới các bạn gặp thì sẽ nhớ ngay nó là gì.

## Phương pháp đánh giá (evaluation method)

Để đánh giá độ chính xác của thuật toán KNN classifier này, chúng ta xem xét có bao nhiêu điểm trong test data được dự đoán đúng. Lấy số lượng này chia cho tổng số lượng trong tập test data sẽ ra độ chính xác. Scikit-learn cung cấp hàm số `accuracy_score` ([http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy\\_score.html](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html)) để thực hiện công việc này.

```
from sklearn.metrics import accuracy_score
print "Accuracy of 1NN: %.2f %%" %(100*accuracy_score(y_test, y_pred))
```

```
Accuracy of 1NN: 94.00 %
```

1NN đã cho chúng ta kết quả là 94%, không tệ! Chú ý rằng đây là một cơ sở dữ liệu dễ vì chỉ với dữ liệu ở hai cột cuối cùng, chúng ta đã có thể suy ra quy luật. Trong ví dụ này, tôi sử dụng  $p = 2$  nghĩa là khoảng cách ở đây được tính là khoảng cách theo norm 2 ( $\sqrt{\sum(x_i - x_{i'})^2}$ ). Các bạn cũng có thể thử bằng cách thay  $p = 1$  cho norm 1 ( $\sum|x_i - x_{i'}|$ ), hoặc các giá trị  $p$  khác cho norm khác. (Xem thêm [sklearn.neighbors.KNeighborsClassifier.html](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html))

Nhận thấy rằng chỉ xét 1 điểm gần nhất có thể dẫn đến kết quả sai nếu điểm đó là nhiễu. Một cách có thể làm tăng độ chính xác là tăng số lượng điểm lân cận lên, ví dụ 10 điểm, và xem xét trong 10 điểm gần nhất, class nào chiếm đa số thì dự đoán kết quả là class đó. Kỹ thuật dựa vào đa số này được gọi là major voting.

```
clf = neighbors.KNeighborsClassifier(n_neighbors = 10, p = 2)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print "Accuracy of 10NN with major voting: %.2f %%" %(100*accuracy_score(y_test, y_pred))
```

Accuracy of 10NN with major voting: 98.00 %

Kết quả đã tăng lên 98%, rất tốt!

### Đánh trọng số cho các điểm lân cận

Là một kẻ tham lam, tôi chưa muôn dừng kết quả ở đây vì thấy rằng mình vẫn có thể cải thiện được. Trong kỹ thuật major voting bên trên, mỗi trong 10 điểm gần nhất được coi là có vai trò như nhau và giá trị *lai phiếu* của mỗi điểm này là như nhau. Tôi cho rằng như thế là không công bằng, vì rõ ràng rằng những điểm gần hơn nên có trọng số cao hơn (*càng thân cận thì càng tin tưởng*). Vậy nên tôi sẽ đánh trọng số khác nhau cho mỗi trong 10 điểm gần nhất này. Cách đánh trọng số phải thoải mãn điều kiện là một điểm càng gần điểm test data thì phải được đánh trọng số càng cao (tin tưởng hơn). Cách đơn giản nhất là lấy nghịch đảo của khoảng cách này. (Trong trường hợp test data trùng với 1 điểm dữ liệu trong training data, tức khoảng cách bằng 0, ta lấy luôn label của điểm training data).

Scikit-learn giúp chúng ta đơn giản hóa việc này bằng cách gán giá trị `weights = 'distance'`. (Giá trị mặc định của `weights` là `'uniform'`, tương ứng với việc coi tất cả các điểm lân cận có giá trị như nhau như ở trên).

```
clf = neighbors.KNeighborsClassifier(n_neighbors = 10, p = 2, weights = 'distance')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print "Accuracy of 10NN (1/distance weights): %.2f %%" %(100*accuracy_score(y_test, y_pred))
```

Accuracy of 10NN (1/distance weights): 100.00 %

Aha, 100%.

**Chú ý:** Ngoài 2 phương pháp đánh trọng số `weights = 'uniform'` và `weights = 'distance'` ở trên, scikit-learn còn cung cấp cho chúng ta một cách để đánh trọng số một cách tùy chọn. Ví dụ, một cách đánh trọng số phổ biến khác trong Machine Learning là:

$$w_i = \exp\left(\frac{-||\mathbf{x} - \mathbf{x}_i||_2^2}{\sigma^2}\right)$$

trong đó  $\mathbf{x}$  là test data,  $\mathbf{x}_i$  là một điểm trong K-lân cận của  $\mathbf{x}$ ,  $w_i$  là trọng số của điểm đó (ứng với điểm dữ liệu đang xét  $\mathbf{x}$ ),  $\sigma$  là một số dương. Nhận thấy rằng hàm số này cũng thỏa mãn điều kiện: điểm càng gần  $\mathbf{x}$  thì trọng số càng cao (cao nhất bằng 1). Với hàm số này, chúng ta có thể lập trình như sau:

```
def myweight(distances):
    sigma2 = .5 # we can change this number
    return np.exp(-distances**2/sigma2)

clf = neighbors.KNeighborsClassifier(n_neighbors = 10, p = 2, weights = myweight)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print "Accuracy of 10NN (customized weights): %.2f %%" %(100*accuracy_score(y_test, y_pred))
```

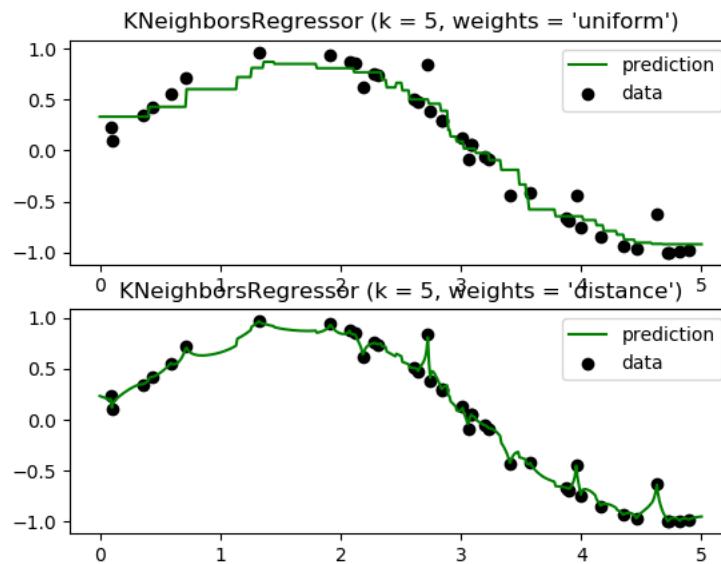
Accuracy of 10NN (customized weights): 98.00 %

Trong trường hợp này, kết quả tương đương với kỹ thuật major voting. Để đánh giá chính xác hơn kết quả của KNN với K khác nhau, cách định nghĩa khoảng cách khác nhau và cách đánh trọng số khác nhau, chúng ta cần thực hiện quá trình trên với nhiều cách chia dữ liệu *training* và *test* khác nhau rồi lấy kết quả trung bình, vì rất có thể dữ liệu phân chia trong 1 trường hợp cụ thể là rất tốt hoặc rất xấu (bias). Đây cũng là cách thường được dùng khi đánh giá hiệu năng của một thuật toán cụ thể nào đó.

## 4. Thảo luận

### KNN cho Regression

Với bài toán Regression, chúng ta cũng hoàn toàn có thể sử dụng phương pháp tương tự: ước lượng đầu ra dựa trên đầu ra và khoảng cách của các điểm trong K-lân cận. Việc ước lượng như thế nào các bạn có thể tự định nghĩa tùy vào từng bài toán.



KNN cho bài toán Regression (Nguồn: Nearest Neighbors regression ([http://scikit-learn.org/stable/auto\\_examples/neighbors/plot\\_regression.html#sphx-glr-auto-examples-neighbors-plot-regression-py](http://scikit-learn.org/stable/auto_examples/neighbors/plot_regression.html#sphx-glr-auto-examples-neighbors-plot-regression-py)))

### Chuẩn hóa dữ liệu

Khi có một thuộc tính trong dữ liệu (hay phần tử trong vector) lớn hơn các thuộc tính khác rất nhiều (ví dụ thay vì đo bằng cm thì một kết quả lại tính bằng mm), khoảng cách giữa các điểm sẽ phụ thuộc vào thuộc tính này rất nhiều. Để có được kết quả chính xác hơn, một kỹ thuật thường được dùng là *Data Normalization* (chuẩn hóa dữ liệu) để đưa các thuộc tính có đơn vị đo khác nhau về cùng một khoảng giá trị, thường là từ 0 đến 1, trước khi thực hiện KNN. Có nhiều kỹ thuật chuẩn hóa khác nhau, các bạn sẽ được thấy khi tiếp tục theo dõi Blog này. Các kỹ thuật chuẩn hóa được áp dụng với không chỉ KNN mà còn với hầu hết các thuật toán khác.

### Sử dụng các phép đo khoảng cách khác nhau

Ngoài norm 1 và norm 2 tôi giới thiệu trong bài này, còn rất nhiều các phép đo khoảng cách khác nhau có thể được dùng. Một ví dụ đơn giản là đếm số lượng thuộc tính khác nhau giữa hai điểm dữ liệu. Số này càng nhỏ thì hai điểm càng gần nhau. Đây chính là giả chuẩn 0 ( $\| \mathbf{x} - \mathbf{y} \|_0$ ) mà tôi đã giới thiệu trong Tab Math ( $\| \mathbf{x} \|_0$ ).

### Ưu điểm của KNN

1. Độ phức tạp tính toán của quá trình training là bằng 0.
2. Việc dự đoán kết quả của dữ liệu mới rất đơn giản.
3. Không cần giả sử gì về phân phối của các class.

### Nhược điểm của KNN

1. KNN rất nhạy cảm với nhiễu khi K nhỏ.
2. Như đã nói, KNN là một thuật toán mà mọi tính toán đều nằm ở khâu test. Trong đó việc tính khoảng cách tới *từng* điểm dữ liệu trong training set sẽ tốn rất nhiều thời gian, đặc biệt là với các cơ sở dữ liệu có số chiều lớn và có nhiều điểm dữ liệu. Với K càng lớn thì độ phức tạp cũng sẽ tăng lên. Ngoài ra, việc lưu toàn bộ dữ liệu trong bộ nhớ cũng ảnh hưởng tới hiệu năng của KNN.

### Tăng tốc cho KNN

Ngoài việc tính toán khoảng cách từ một điểm test data đến tất cả các điểm trong training set (Brute Force), có một số thuật toán khác giúp tăng tốc việc tìm kiếm này. Bạn đọc có thể tìm kiếm thêm với hai từ khóa: K-D Tree ([http://pointclouds.org/documentation/tutorials/kdtree\\_search.php](http://pointclouds.org/documentation/tutorials/kdtree_search.php)) và Ball Tree ([https://en.wikipedia.org/wiki/Ball\\_tree](https://en.wikipedia.org/wiki/Ball_tree)). Tôi xin dành phần này cho độc giả tự tìm hiểu, và sẽ quay lại nếu có dịp. Chúng ta vẫn còn những thuật toán quan trọng hơn khác cần nhiều sự quan tâm hơn.

## Try this yourself

Tôi có viết một đoạn code ngắn để thực hiện việc Classification cho cơ sở dữ liệu MNIST (/2017/01/04/kmeans2/#bo-co-so-du-lieu-mnist). Các bạn hãy download toàn bộ bộ dữ liệu này về vì sau này chúng ta còn dùng nhiều, chạy thử, comment kết quả và nhận xét của các bạn vào phần comment bên dưới. Để trả lời cho câu hỏi vì sao tôi không chọn cơ sở dữ liệu này làm ví dụ, bạn đọc có thể tự tìm ra đáp án khi chạy xong đoạn code này.

Enjoy!

```
# %reset
import numpy as np
from mnist import MNIST # require `pip install python-mnist`
# https://pypi.python.org/pypi/python-mnist/

import matplotlib.pyplot as plt
from sklearn import neighbors
from sklearn.metrics import accuracy_score
import time

# you need to download the MNIST dataset first
# at: http://yann.lecun.com/exdb/mnist/
mndata = MNIST('..../MNIST/') # path to your MNIST folder
mndata.load_testing()
mndata.load_training()
X_test = mndata.test_images
X_train = mndata.train_images
y_test = np.asarray(mndata.test_labels)
y_train = np.asarray(mndata.train_labels)

start_time = time.time()
clf = neighbors.KNeighborsClassifier(n_neighbors = 1, p = 2)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
end_time = time.time()
print "Accuracy of 1NN for MNIST: %.2f %%" %(100*accuracy_score(y_test, y_pred))
print "Running time: %.2f (s)" % (end_time - start_time)
```

## Source code

iPython Notebook cho bài này có thể download tại đây (<https://github.com/tiepvupsu/tiepvupsu.github.io/tree/master/assets/knn/KNN.ipynb>).

## 5. Tài liệu tham khảo

1. [sklearn.neighbors.NearestNeighbors](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors) (<http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors>)
2. [sklearn.model\\_selection.train\\_test\\_split](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) ([http://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.train\\_test\\_split.html](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html))
3. Tutorial To Implement k-Nearest Neighbors in Python From Scratch (<http://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/>)

Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (<https://www.facebook.com/groups/257768141347267/>) để nhận được câu trả lời sớm hơn.

Bạn đọc có thể ủng hộ blog qua 'Buy me a coffee' (/buymeacoffee/) ở góc trên bên trái của blog.

Tôi đang trong quá trình viết cuốn sách 'Machine Learning cơ bản I', các bạn có thể đặt trước tại đây (/ebook/). Cảm ơn bạn.

« Bài 5: K-means Clustering: Simple Applications (/2017/01/04/kmeans2/)

Bài 7: Gradient Descent (phần 1/2) » (/2017/01/12/gradientdescent/)

21 Comments tiepvu

 Login ▾

 Recommend 5  Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name



DaoTuan • 9 months ago

Ở bài này phải dùng đến cái "t10k-labels-idx1-ubyte" bài trước mới chỉ dùng đến "t10k-images-idx3-ubyte" thôi a. Nên em nghĩ anh nên ghi thêm để các bạn test không bị lỗi a. Em đã test và bị lỗi nên đã download về chạy và không có vấn đề a.  
 Kết quả đây a:  
 Accuracy of 1NN for MNIST: 96.91 %  
 Running time: 622.64 (s)  
 Khá tốn thời gian so với cái Iris flower dataset a!

[2 ^](#) [|](#) [v](#) [• Reply](#) [• Share](#)

Tiep Vu Huu Mod → DaoTuan • 9 months ago

Kết quả giống với anh, khá tốt với 96.91%, chứng tỏ bộ cơ sở dữ liệu này hơi dễ.  
 Về thời gian chạy, chậm hơn cho với Iris là hiển nhiên vì số lượng và số chiều dữ liệu lớn. Việc này cho thấy KNN không phù hợp với large-scale problems. Phải có thêm những 'tricks' khác thì mới áp dụng được.

[2 ^](#) [|](#) [v](#) [• Reply](#) [• Share](#)

Thái Tấn Lợi • 3 months ago

A Tiệp có thể giải thích giúp e câu a ghi "không học một điều gì từ dữ liệu training". Em đang muốn so sánh với SVM để tìm cái tối ưu cho bài toán của em

[1 ^](#) [|](#) [v](#) [• Reply](#) [• Share](#)

Tiep Vu Huu Mod → Thái Tấn Lợi • 3 months ago

SVM thường sẽ cho kết quả tốt hơn KNN.

Về cụm "không học một điều gì từ dữ liệu training", ý anh là thời gian training bằng 0, mọi công việc đều được thực hiện ở quá trình test. Mọi công việc đó bao gồm tính toán khoảng cách tới từng điểm trong training rồi sắp xếp các khoảng cách đó.

Ở đây KNN chỉ 'ghi nhớ' dữ liệu training chứ không học gì.

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#)

Thái Tấn Lợi → Tiep Vu Huu • 2 months ago

Vậy em có thể dùng kNN để pre-process cho dữ liệu tranining lớn để sắp xếp lại dữ liệu tranining được không a

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#)

Tiep Vu Huu Mod → Thái Tấn Lợi • 2 months ago

Mình chưa hiểu bạn sẽ pre-process như thế nào. Mình ít khi thấy người ta làm pre-process bằng kNN.

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#)

Quoc Tc • 3 months ago

Cho mình hỏi có vấn đề mình chưa hiểu chỗ này:

Khi cho 1 điểm data test bất kì làm sao thuật toán này biết được n điểm data gần nhất với nó để từ đó đưa ra output? Theo mình nghĩ thì nó sẽ tính hết khoảng cách từ điểm data test này tới tất cả các điểm trong training data sau đó lấy ra 10 điểm có khoảng cách gần nhất đúng ko ad?

[1 ^](#) [|](#) [v](#) [• Reply](#) [• Share](#)

Tiep Vu Huu Mod → Quoc Tc • 3 months ago

Chuẩn rồi bạn.

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#)

Sand King • 6 months ago

#Accuracy: 96.91  
#Running time: 755.15 (s)[1 ^](#) [|](#) [v](#) [• Reply](#) [• Share](#)

Vinh Tống • 3 months ago

Em chạy thử bị lỗi này:

ValueError: Magic number mismatch, expected 2049, got 529205256

Anh giúp em xử lý được k a?

[^](#) [|](#) [v](#) [• Reply](#) [• Share](#)

Tiep Vu Huu Mod → Vinh Tống • 3 months ago

Đây là lỗi của thư viện mnist roi. Anh chưa gộp bao giờ. Em google thêm thư xem.

[^](#) [v](#) • Reply • Share >



**Vinh Tống** → Tiep Vu Huu • 3 months ago

Dạ, em chuyển sang linux chạy và không bị lỗi nữa, em cảm ơn anh ạ!

[^](#) [v](#) • Reply • Share >



**AT** • 5 months ago

e chạy file bị lỗi hoài a ơi. a có thẻ cho e file đầy đủ được không à? e mới học nên cũng chưa biết gì nhiều cả...

[^](#) [v](#) • Reply • Share >



**CTVR** → AT • 5 months ago

Bạn bị lỗi gì? Share lên mình giúp cho

1 [^](#) [v](#) • Reply • Share >



**Hiền** • 8 months ago

Với vai trò classification, em thấy việc phân loại 1 điểm dữ liệu thuộc phụ thuộc vào "khoảng cách" tới các điểm gần nó nhất ==> kNN có phải là bước phân loại cluster trong k means clustering ko anh?

[^](#) [v](#) • Reply • Share >



**Tiep Vu Huu Mod** → Hiền • 8 months ago

Với 1NN thì đúng là như vậy. Về cơ bản vẫn là xét tính chất của một điểm dựa trên điểm gần nó nhất.

[^](#) [v](#) • Reply • Share >



**TRUNG LEVAN** → Tiep Vu Huu • 2 months ago

nếu ngẫu nhiên có nhiều điểm trùng trọng số thì sao nhỉ ad :?

[^](#) [v](#) • Reply • Share >



**nhanth87** → TRUNG LEVAN • a month ago

Nếu nhiều điểm trùng trọng số có nghĩa các điểm thuộc các class khác nhau nằm lấn lộn, lúc đó k-NN sẽ cho ra kết quả k chính xác nữa

[^](#) [v](#) • Reply • Share >



**Khang Nguyễn Vương Duy** • 6 days ago

lỗi thế này sửa s ạ mn !



[^](#) [v](#) • Reply • Share >



**Tiep Vu Huu Mod** → Khang Nguyễn Vương Duy • 6 days ago

Bạn dùng python 3.6, hàm string muốn print phải được đặt trong dấu ngoặc () .

Ví dụ, dòng 7 phải là:

```
print("Number of classes :%d" % len(np.unique(iris_y)))
```

[^](#) [v](#) • Reply • Share >



**Khang Nguyễn Vương Duy** → Tiep Vu Huu • 6 days ago

đã được rồi ạ cảm ơn anh nhiều .

[^](#) [v](#) • Reply • Share >

ALSO ON THIS PAGE

**ebook Machine Learning cơ bản**

26 comments • 3 months ago

Xuân Xuân — Anh ơi, cái email trường em cấp ý, em test nó không cho các mail ngoài gửi vào:"Hi. This is the qmail-send program at vnu.edu.vn.I'm ...

[Subscribe](#) [Add Disqus to your site](#) [Add Disqus Add](#) [Privacy](#)**Machine Learning cơ bản**

11 comments • 8 months ago

CTVR — Em cũng thấy yêu cầu 64 bit trên win. Chắc em chạy với linux vậy.  
P/s: Giờ em mới đọc xong loạt bài về convex trên file pdf. Hồi mới vào ...

Total visits:

## Bài 7: Gradient Descent (phần 1/2)

GD (/tags#GD) Optimization (/tags#Optimization)

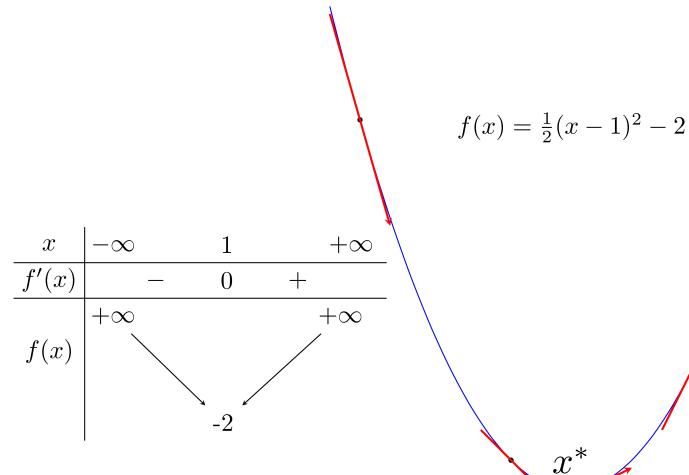
Jan 12, 2017

**Trong trang này:**

- 1. Giới thiệu
  - Gradient Descent
- 2. Gradient Descent cho hàm 1 biến
  - Ví dụ đơn giản với Python
    - Điểm khởi tạo khác nhau
    - Learning rate khác nhau
- 3. Gradient Descent cho hàm nhiều biến
  - Quay lại với bài toán Linear Regression
  - Sau đây là ví dụ trên Python và một vài lưu ý khi lập trình
    - Kiểm tra đạo hàm
      - Giải thích bằng hình học
      - Giải thích bằng giải tích
      - Với hàm nhiều biến
    - Đường đồng mức (level sets)
- 4. Một ví dụ khác
- 5. Thảo luận
- 6. Tài liệu tham khảo

### 1. Giới thiệu

Các bạn hẳn thấy hình vẽ dưới đây quen thuộc:



Điểm màu xanh lục là điểm local minimum (cực tiểu), và cũng là điểm làm cho hàm số đạt giá trị nhỏ nhất. Từ đây trở đi, tôi sẽ dùng *local minimum* để thay cho *điểm cực tiểu*, *global minimum* để thay cho *điểm mà tại đó hàm số đạt giá trị nhỏ nhất*. Global minimum là một trường hợp đặc biệt của local minimum.

Giả sử chúng ta đang quan tâm đến một hàm số một biến có đạo hàm mọi nơi. Xin cho tôi được nhắc lại vài điều đã quá quen thuộc:

1. Điểm local minimum  $x^*$  của hàm số là điểm có đạo hàm  $f'(x^*)$  bằng 0. Hơn thế nữa, trong lân cận của nó, đạo hàm của các điểm phía bên trái  $x^*$  là không dương, đạo hàm của các điểm phía bên phải  $x^*$  là không âm.
2. Đường tiếp tuyến với đồ thị hàm số đó tại 1 điểm bất kỳ có hệ số góc chính bằng đạo hàm của hàm số tại điểm đó.

Trong hình phía trên, các điểm bên trái của điểm local minimum màu xanh lục có đạo hàm âm, các điểm bên phải có đạo hàm dương. Và đối với hàm số này, càng xa về phía trái của điểm local minimum thì đạo hàm càng âm, càng xa về phía phải thì đạo hàm càng dương.

### Gradient Descent

Trong Machine Learning nói riêng và Toán Tối Ưu nói chung, chúng ta thường xuyên phải tìm giá trị nhỏ nhất (hoặc đôi khi là lớn nhất) của một hàm số nào đó. Ví dụ như các hàm mất mát trong hai bài Linear Regression (/2016/12/28/linearregression/) và K-means Clustering (/2017/01/01/kmeans/). Nhìn chung, việc tìm global minimum của các hàm mất mát trong Machine Learning là rất phức tạp, thậm chí là bất khả thi. Thay vào đó, người ta thường cố gắng tìm các điểm local minimum, và ở một mức độ nào đó, coi đó là nghiệm cần tìm của bài toán.

Các điểm local minimum là nghiệm của phương trình đạo hàm bằng 0. Nếu bằng một cách nào đó có thể tìm được toàn bộ (hữu hạn) các điểm cực tiểu, ta chỉ cần thay từng điểm local minimum đó vào hàm số rồi tìm điểm làm cho hàm có giá trị nhỏ nhất (*đoạn này nghe rất quen thuộc, đúng không?*). Tuy nhiên, trong hầu hết các trường hợp, việc giải phương trình đạo hàm bằng 0 là bất khả thi. Nguyên nhân có thể đến từ sự phức tạp của dạng của đạo hàm, từ việc các điểm dữ liệu có số chiều lớn, hoặc từ việc có quá nhiều điểm dữ liệu.

Hướng tiếp cận phổ biến nhất là xuất phát từ một điểm mà chúng ta coi là gần với nghiệm của bài toán, sau đó dùng một phép toán lặp để tiến dần đến điểm cần tìm, tức đến khi đạo hàm gần với 0. Gradient Descent (viết gọn là GD) và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất.

Vì kiến thức về GD khá rộng nên tôi xin phép được chia thành hai phần. Phần 1 này giới thiệu ý tưởng phía sau thuật toán GD và một vài ví dụ đơn giản giúp các bạn làm quen với thuật toán này và vài khái niệm mới. Phần 2 sẽ nói về các phương pháp cải tiến GD và các biến thể của GD trong các bài toán mà số chiều và số điểm dữ liệu lớn. Những bài toán như vậy được gọi là *large-scale*.

## 2. Gradient Descent cho hàm 1 biến

Quay trở lại hình vẽ ban đầu và một vài quan sát tôi đã nêu. Giả sử  $x_t$  là điểm ta tìm được sau vòng lặp thứ  $t$ . Ta cần tìm một thuật toán để đưa  $x_t$  về càng gần  $x^*$  càng tốt.

Trong hình đầu tiên, chúng ta lại có thêm hai quan sát nữa:

- Nếu đạo hàm của hàm số tại  $x_t$ :  $f'(x_t) > 0$  thì  $x_t$  nằm về bên phải so với  $x^*$  (và ngược lại). Để điểm tiếp theo  $x_{t+1}$  gần với  $x^*$  hơn, chúng ta cần di chuyển  $x_t$  về phía bên trái, tức về phía  $\hat{a}m$ . Nói cách khác, **chúng ta cần di chuyển ngược dấu với đạo hàm**:

$$x_{t+1} = x_t + \Delta$$

Trong đó  $\Delta$  là một đại lượng ngược dấu với đạo hàm  $f'(x_t)$ .

- $x_t$  càng xa  $x^*$  về phía bên phải thì  $f'(x_t)$  càng lớn hơn 0 (và ngược lại). Vậy, lượng di chuyển  $\Delta$ , một cách trực quan nhất, là tỉ lệ thuận với  $-f'(x_t)$ .

Hai nhận xét phía trên cho chúng ta một cách cập nhật đơn giản là:

$$x_{t+1} = x_t - \eta f'(x_t)$$

Trong đó  $\eta$  (đọc là *eta*) là một số dương được gọi là *learning rate* (tốc độ học). Dấu trừ thể hiện việc chúng ta phải *đi ngược* với đạo hàm (Đây cũng chính là lý do phương pháp này được gọi là Gradient Descent - *descent* nghĩa là *đi ngược*). Các quan sát đơn giản phía trên, mặc dù không phải đúng cho tất cả các bài toán, là nền tảng cho rất nhiều phương pháp tối ưu nói chung và thuật toán Machine Learning nói riêng.

### Ví dụ đơn giản với Python

Xét hàm số  $f(x) = x^2 + 5 \sin(x)$  với đạo hàm  $f'(x) = 2x + 5 \cos(x)$  (một lý do tôi chọn hàm này vì nó không dễ tìm nghiệm của đạo hàm bằng 0 như hàm phía trên). Giả sử bắt đầu từ một điểm  $x_0$  nào đó, tại vòng lặp thứ  $t$ , chúng ta sẽ cập nhật như sau:

$$x_{t+1} = x_t - \eta(2x_t + 5 \cos(x_t))$$

Như thường lệ, tôi khai báo vài thư viện quen thuộc

```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals
import math
import numpy as np
import matplotlib.pyplot as plt
```

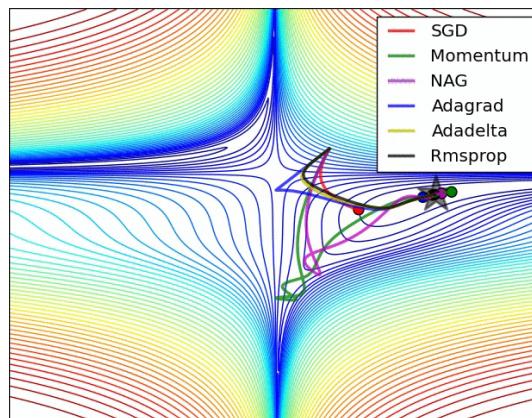
Tiếp theo, tôi viết các hàm số :

- grad để tính đạo hàm
- cost để tính giá trị của hàm số. Hàm này không sử dụng trong thuật toán nhưng thường được dùng để kiểm tra việc tính đạo hàm của đúng không hoặc để xem giá trị của hàm số có giảm theo mỗi vòng lặp hay không.
- myGD1 là phần chính thực hiện thuật toán Gradient Desent nêu phía trên. Đầu vào của hàm số này là learning rate và điểm bắt đầu. Thuật toán dừng lại khi đạo hàm có độ lớn đủ nhỏ.

## Bài 8: Gradient Descent (phần 2/2)

GD (/tags#GD) Optimization (/tags#Optimization) Online-learning (/tags#Online-learning) Batch (/tags#Batch)

Jan 16, 2017



Tốc độ hội tụ của các thuật toán GD khác nhau. (Nguồn (<http://sebastianruder.com/optimizing-gradient-descent/index.html#stochasticgradientdescent>) An overview of gradient descent optimization algorithms).

Trong phần 1 (/2017/01/12/gradientdescent/) của Gradient Descent (GD), tôi đã giới thiệu với bạn đọc về thuật toán Gradient Descent. Tôi xin nhắc lại rằng nghiệm cuối cùng của Gradient Descent phụ thuộc rất nhiều vào điểm khởi tạo và learning rate. Trong bài này, tôi xin đề cập một vài phương pháp thường được dùng để khắc phục những hạn chế của GD. Đồng thời, các thuật toán biến thể của GD thường được áp dụng trong các mô hình Deep Learning cũng sẽ được tổng hợp.

**Trong trang này:**

- 1. Các thuật toán tối ưu Gradient Descent
  - 1.1 Momentum
    - Nhắc lại thuật toán Gradient Descent
    - Gradient dưới góc nhìn vật lý
    - Gradient Descent với Momentum
    - Một ví dụ nhỏ
  - 1.2. Nesterov accelerated gradient (NAG)
    - Ý tưởng chính
    - Công thức cập nhật
    - Ví dụ minh họa
  - 1.3. Các thuật toán khác
- 2. Biến thể của Gradient Descent
  - 2.1. Batch Gradient Descent
  - 2.2. Stochastic Gradient Descent.
    - Ví dụ với bài toán Linear Regression
  - 2.3. Mini-batch Gradient Descent
- 3. Stopping Criteria (điều kiện dừng)
- 4. Một phương pháp tối ưu đơn giản khác: Newton's method
  - Newton's method cho giải phương trình  $f(x) = 0$
  - Newton's method trong bài toán tìm local minimum
  - Hạn chế của Newton's method
- 5. Kết luận
- 6. Tài liệu tham khảo

### 1. Các thuật toán tối ưu Gradient Descent

#### 1.1 Momentum

Nhắc lại thuật toán Gradient Descent

Dành cho các bạn chưa đọc phần 1 (/2017/01/12/gradientdescent/) của Gradient Descent. Để giải bài toán tìm điểm *global optimal* của hàm mất mát  $J(\theta)$  (Hàm mất mát cũng thường được ký hiệu là  $J()$  với  $\theta$  là tập hợp các tham số của mô hình), tôi xin nhắc lại thuật toán GD:

### Thuật toán Gradient Descent:

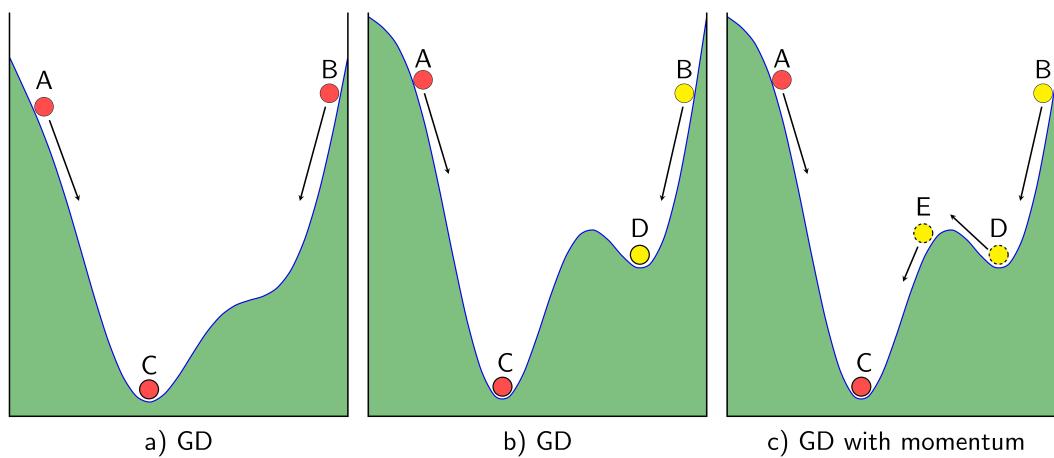
1. Dự đoán một điểm khởi tạo  $\theta = \theta_0$ .
2. Cập nhật  $\theta$  đến khi đạt được kết quả chấp nhận được:

$$\theta = \theta - \eta \nabla_{\theta} J(\theta)$$

với  $\nabla_{\theta} J(\theta)$  là đạo hàm của hàm mất mát tại  $\theta$ .

### Gradient dưới góc nhìn vật lý

Thuật toán GD thường được ví với tác dụng của trọng lực lên một hòn bi đặt trên một mặt có dạng như hình một thung lũng giống như hình 1a) dưới đây. Bất kể ta đặt hòn bi ở A hay B thì cuối cùng hòn bi cũng sẽ lăn xuống và kết thúc ở vị trí C.



Hình 1: So sánh Gradient Descent với các hiện tượng vật lý

Tuy nhiên, nếu như bề mặt có hai đáy thung lũng như Hình 1b) thì tùy vào việc đặt bi ở A hay B, vị trí cuối cùng của bi sẽ ở C hoặc D. Điểm D là một điểm local minimum chúng ta không mong muốn.

Nếu suy nghĩ một cách vật lý hơn, vẫn trong Hình 1b), nếu vận tốc ban đầu của bi khi ở điểm B đủ lớn, khi bi lăn đến điểm D, theo đà, bi có thể tiếp tục di chuyển lên dốc phía bên trái của D. Và nếu giả sử vận tốc ban đầu lớn hơn nữa, bi có thể vượt dốc tới điểm E rồi lăn xuống C như trong Hình 1c). Đây chính là điều chúng ta mong muốn. Bạn đọc có thể đặt câu hỏi rằng liệu bi lăn từ A tới C có theo đà lăn tới E rồi tới D không. Xin trả lời rằng điều này khó xảy ra hơn vì nếu so với dốc DE thì dốc CE cao hơn nhiều.

Dựa trên hiện tượng này, một thuật toán được ra đời nhằm khắc phục việc nghiêm của GD rơi vào một điểm local minimum không mong muốn. Thuật toán đó có tên là Momentum (tức *theo đà* trong tiếng Việt).

### Gradient Descent với Momentum

Để biểu diễn *momentum* bằng toán học thì chúng ta phải làm thế nào?

Trong GD, chúng ta cần tính lượng thay đổi ở thời điểm  $t$  để cập nhật vị trí mới cho nghiệm (tức *hòn bi*). Nếu chúng ta coi đại lượng này như vận tốc  $v_t$  trong vật lý, vị trí mới của *hòn bi* sẽ là  $\theta_{t+1} = \theta_t - v_t$ . Dẫu trù thi hiện việc phải di chuyển ngược với đạo hàm. Công việc của chúng ta bây giờ là tính đại lượng  $v_t$  sao cho nó vừa mang thông tin của *độ dốc* (tức đạo hàm), vừa mang thông tin của *đà*, tức vận tốc trước đó  $v_{t-1}$  (chúng ta coi như vận tốc ban đầu  $v_0 = 0$ ). Một cách đơn giản nhất, ta có thể cộng (có trọng số) hai đại lượng này lại:

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$

Trong đó  $\gamma$  thường được chọn là một giá trị khoảng 0.9,  $v_t$  là vận tốc tại thời điểm trước đó,  $\nabla_{\theta} J(\theta)$  chính là độ dốc của điểm trước đó. Sau đó vị trí mới của *hòn bi* được xác định như sau:

$$\theta = \theta - v_t$$

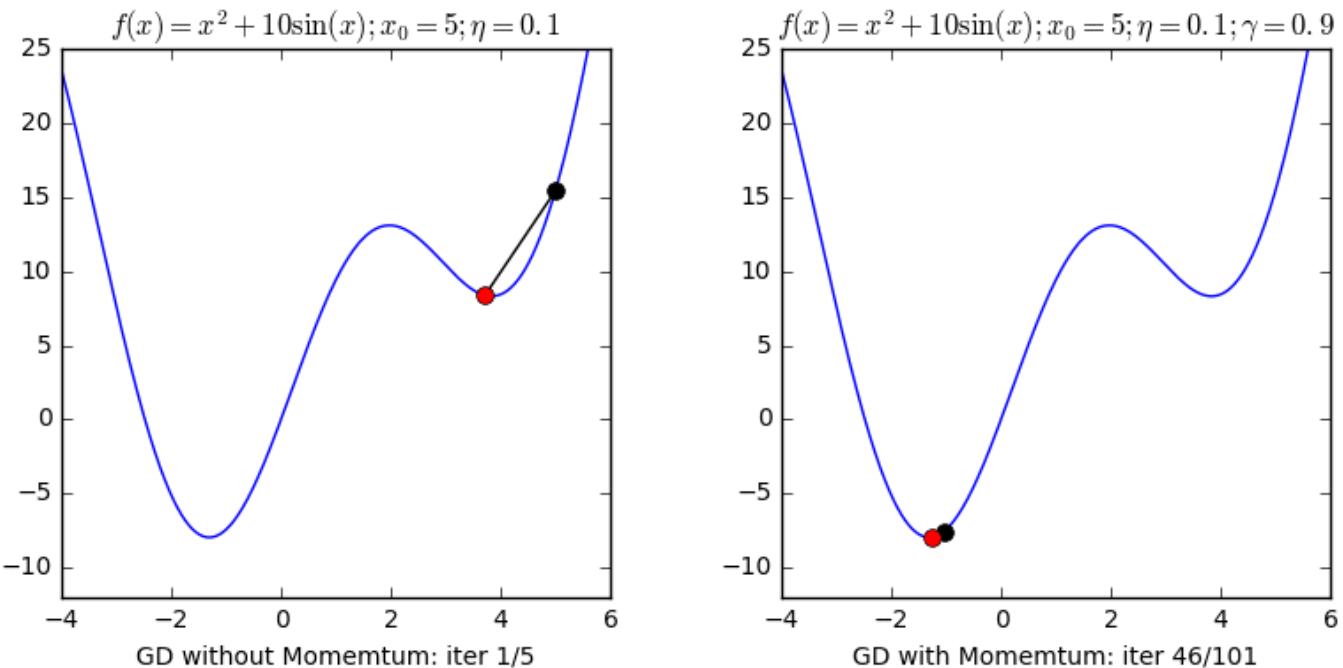
Thuật toán đơn giản này tỏ ra rất hiệu quả trong các bài toán thực tế (trong không gian nhiều chiều, cách tính toán cũng hoàn toàn tương tự). Dưới đây là một ví dụ trong không gian một chiều.

### Một ví dụ nhỏ

Chúng ta xem xét một hàm đơn giản có hai điểm local minimum, trong đó 1 điểm là global minimum:

$$f(x) = x^2 + 10 \sin(x)$$

Có đạo hàm là:  $f'(x) = 2x + 10 \cos(x)$ . Hình 2 dưới đây thể hiện sự khác nhau giữa thuật toán GD và thuật toán GD với Momentum:



Hình 2: Minh họa thuật toán GD với Momentum.

Hình bên trái là đường đi của nghiệm khi không sử dụng Momentum, thuật toán hội tụ sau chỉ 5 vòng lặp nhưng nghiệm tìm được là nghiệm local minimun.

Hình bên phải là đường đi của nghiệm khi có sử dụng Momentum, *hòn bi* đã có thể vượt dốc tới khu vực gần điểm global minimun, sau đó dao động xung quanh điểm này, giảm tốc rồi cuối cùng tới đích. Mặc dù mất nhiều vòng lặp hơn, GD với Momentum cho chúng ta nghiệm chính xác hơn. Quan sát đường đi của *hòn bi* trong trường hợp này, chúng ta thấy rằng điều này giống với vật lý hơn!

Nếu biết trước điểm *đặt bi* ban đầu theta, đạo hàm của hàm mất mát tại một điểm bất kỳ  $\text{grad}(\theta)$ , lượng thông tin lưu trữ từ vận tốc trước đó gamma và learning rate eta, chúng ta có thể viết hàm số GD\_momentum trong Python như sau:

```
# check convergence
def has_converged(theta_new, grad):
    return np.linalg.norm(grad(theta_new)) / len(theta_new) < 1e-3

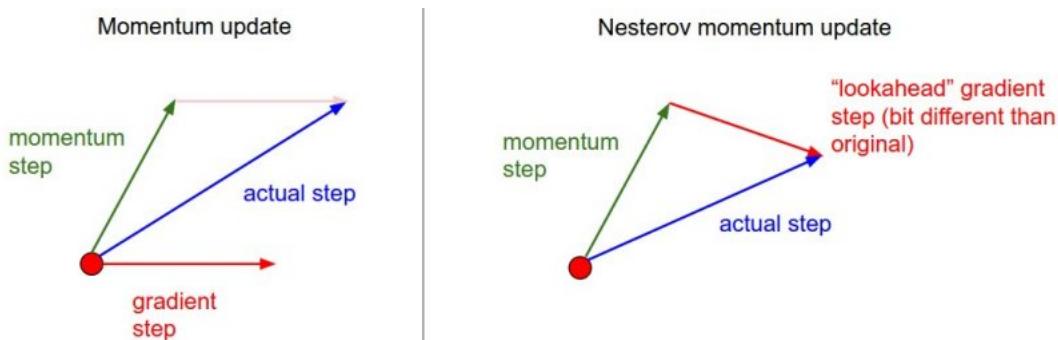
def GD_momentum(theta_init, grad, eta, gamma):
    # Suppose we want to store history of theta
    theta = [theta_init]
    v_old = np.zeros_like(theta_init)
    for it in range(100):
        v_new = gamma*v_old + eta*grad(theta[-1])
        theta_new = theta[-1] - v_new
        if has_converged(theta_new, grad):
            break
        theta.append(theta_new)
        v_old = v_new
    return theta
    # this variable includes all points in the path
    # if you just want the final answer,
    # use `return theta[-1]`
```

## 1.2. Nesterov accelerated gradient (NAG)

Momentum giúp *hòn bi* vượt qua được *dốc locaminimum*, tuy nhiên, có một hạn chế chúng ta có thể thấy trong ví dụ trên: Khi tới gần *đích*, momentum vẫn mất khá nhiều thời gian trước khi dừng lại. Lý do lại cũng chính là vì có *đá*. Có một phương pháp khác tiếp tục giúp khắc phục điều này, phương pháp đó mang tên Nesterov accelerated gradient (NAG), giúp cho thuật toán hội tụ nhanh hơn.

Ý tưởng chính

Ý tưởng cơ bản là *dự đoán hướng đi trong tương lai*, tức nhìn trước một bước! Cụ thể, nếu sử dụng số hạng *momentum*  $\gamma v_{t-1}$  để cập nhật thì ta có thể xấp xỉ được vị trí tiếp theo của hòn bi là  $\theta - \gamma v_{t-1}$  (chúng ta không định kèm phần gradient ở đây vì sẽ sử dụng nó trong bước cuối cùng). Vậy, thay vì sử dụng gradient của điểm hiện tại, NAG *đi trước một bước*, sử dụng gradient của điểm tiếp theo. Theo dõi hình dưới đây:



Ý tưởng của Nesterov accelerated gradient. (Nguồn: CS231n Stanford: Convolutional Neural Networks for Visual Recognition (<https://cs231n.github.io/neural-networks-3/>)

- Với momentum thông thường: *lượng thay đổi* là tổng của hai vector: momentum vector và gradient ở thời điểm hiện tại.
- Với Nesterove momentum: *lượng thay đổi* là tổng của hai vector: momentum vector và gradient ở thời điểm được xấp xỉ là điểm tiếp theo.

### Công thức cập nhật

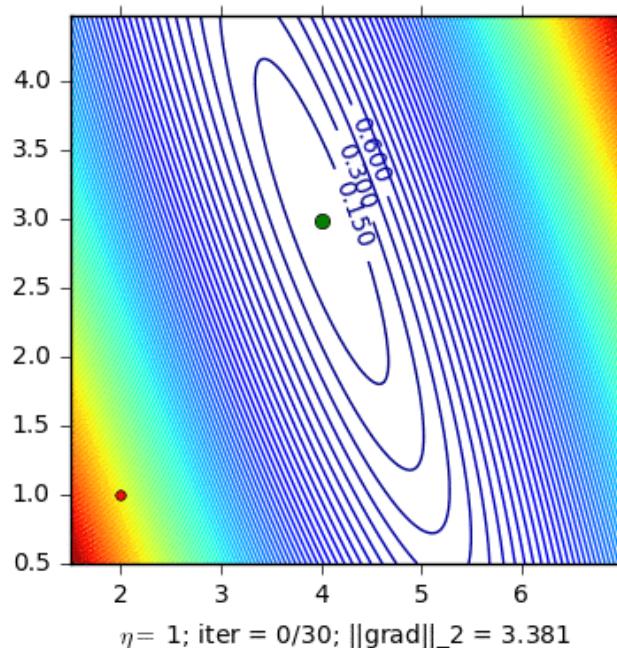
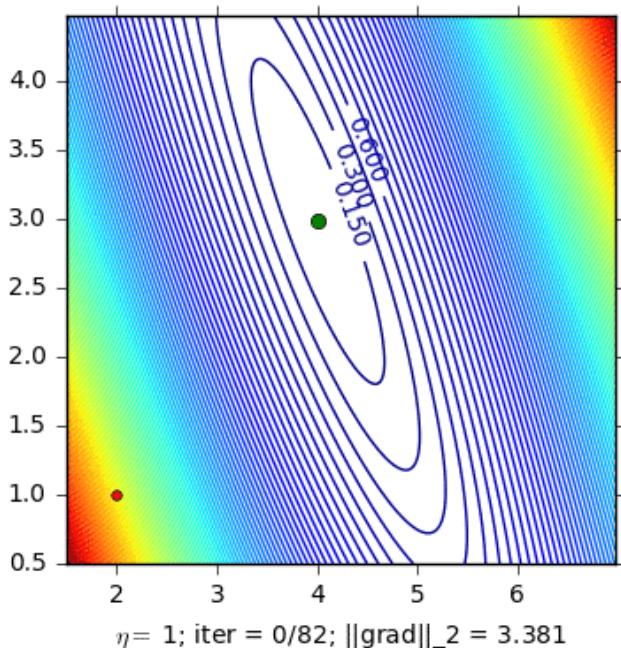
Công thức cập nhật của NAG được cho như sau:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \\ \theta &= \theta - v_t \end{aligned}$$

Để ý một chút các bạn sẽ thấy điểm được tính đạo hàm đã thay đổi.

### Ví dụ minh họa

Dưới đây là ví dụ so sánh Momentum và NAG cho bài toán Linear Regression:



Minh họa thuật toán GD với Momentum và NAG.

Hình bên trái là đường đi của nghiệm với phương pháp Momentum. nghiệm đi khá là zigzag và mất nhiều vòng lặp hơn. Hình bên phải là đường đi của nghiệm với phương pháp NAG, nghiệm hội tụ nhanh hơn, và đường đi ít zigzag hơn.

(Source code cho hình bên trái (<https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/GD/LR%20Momentum.ipynb>) và hình bên phải (<https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/GD/LR%20NAG.ipynb>)).

### 1.3. Các thuật toán khác

Ngoài hai thuật toán trên, có rất nhiều thuật toán nâng cao khác được sử dụng trong các bài toán thực tế, đặc biệt là các bài toán Deep Learning. Có thể nêu một vài từ khóa như Adagrad, Adam, RMSprop,... Tôi sẽ không đề cập đến các thuật toán đó trong bài này mà sẽ dành thời gian nói tới nếu có dịp trong tương lai, khi blog đã đủ lớn và đã trang bị cho các bạn một lượng kiến thức nhất định.

Tuy nhiên, bạn đọc nào muốn đọc thêm có thể tìm được rất nhiều thông tin hữu ích trong bài này: An overview of gradient descent optimization algorithms (<http://sebastianruder.com/optimizing-gradient-descent/index.html#stochasticgradientdescent>).

## 2. Biến thể của Gradient Descent

Tôi xin một lần nữa dùng bài toán Linear Regression (/2016/12/28/linearregression/) làm ví dụ. Hàm mất mát và đạo hàm của nó cho bài toán này lần lượt là (để cho thuận tiện, trong bài này tôi sẽ dùng ký hiệu  $\mathbf{X}$  thay cho dữ liệu mở rộng  $\bar{\mathbf{X}}$ ):

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{2N} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 \\ &= \frac{1}{2N} \sum_{i=1}^N (\mathbf{x}_i \mathbf{w} - y_i)^2 \end{aligned}$$

và:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i^T (\mathbf{x}_i \mathbf{w} - y_i)$$

### 2.1. Batch Gradient Descent

Thuật toán Gradient Descent chúng ta nói từ đầu phần 1 đến giờ còn được gọi là Batch Gradient Descent. Batch ở đây được hiểu là **tất cả**, tức khi cập nhật  $\theta = \mathbf{w}$ , chúng ta sử dụng **tất cả** các điểm dữ liệu  $\mathbf{x}_i$ .

Cách làm này có một vài hạn chế đối với cơ sở dữ liệu có vô cùng nhiều điểm (hơn 1 tỉ người dùng của facebook chẳng hạn). Việc phải tính toán lại đạo hàm với tất cả các điểm này sau mỗi vòng lặp trở nên cồng kềnh và không hiệu quả.Thêm nữa, thuật toán này được coi là không hiệu quả với *online learning*.

**Online learning** là khi cơ sở dữ liệu được cập nhật liên tục (thêm người dùng đăng ký hàng ngày chẳng hạn), mỗi lần thêm vài điểm dữ liệu mới. Kéo theo đó là mô hình của chúng ta cũng phải thay đổi một chút để phù hợp với các dữ liệu mới này. Nếu làm theo Batch Gradient Descent, tức tính lại đạo hàm của hàm mất mát tại tất cả các điểm dữ liệu, thì thời gian tính toán sẽ rất lâu, và thuật toán của chúng ta coi như không *online* nữa do mất quá nhiều thời gian tính toán.

Trên thực tế, có một thuật toán đơn giản hơn và tỏ ra rất hiệu quả, có tên gọi là Stochastic Gradient Descent (SGD).

### 2.2. Stochastic Gradient Descent.

Trong thuật toán này, tại 1 thời điểm, ta chỉ tính đạo hàm của hàm mất mát dựa trên *chỉ một* điểm dữ liệu  $\mathbf{x}_i$ ; rồi cập nhật  $\theta$  dựa trên đạo hàm này. Việc này được thực hiện với từng điểm trên toàn bộ dữ liệu, sau đó lặp lại quá trình trên. Thuật toán rất đơn giản này sẽ thực tế lại làm việc rất hiệu quả.

Mỗi lần duyệt qua *tất cả* các điểm trên toàn bộ dữ liệu được gọi là một epoch. Với GD thông thường thì mỗi epoch ứng với 1 lần cập nhật  $\theta$ , với SGD thì mỗi epoch ứng với  $N$  lần cập nhật  $\theta$  với  $N$  là số điểm dữ liệu. Nhìn vào một mặt, việc cập nhật từng điểm một như thế này có thể làm giảm đi tốc độ thực hiện 1 epoch. Nhưng nhìn vào một mặt khác, SGD chỉ yêu cầu một lượng epoch rất nhỏ (thường là 10 cho lần đầu tiên, sau đó khi có dữ liệu mới thì chỉ cần chạy dưới một epoch là đã có nghiệm tốt). Vì vậy SGD phù hợp với các bài toán có lượng cơ sở dữ liệu lớn (chủ yếu là Deep Learning mà chúng ta sẽ thấy trong phần sau của blog) và các bài toán yêu cầu mô hình thay đổi liên tục, tức *online learning*.

#### Thứ tự lựa chọn điểm dữ liệu

Một điểm cần lưu ý đó là: sau mỗi epoch, chúng ta cần shuffle (xáo trộn) thứ tự của các dữ liệu để đảm bảo tính ngẫu nhiên. Việc này cũng ảnh hưởng tới hiệu năng của SGD.

Một cách toán học, quy tắc cập nhật của SGD là:

$$\theta = \theta - \eta \nabla_{\theta} J(\theta; \mathbf{x}_i; \mathbf{y}_i)$$

trong đó  $J(\theta; \mathbf{x}_i; \mathbf{y}_i)$  là hàm mất mát với chỉ 1 cặp điểm dữ liệu (input, label) là  $(\mathbf{x}_i, \mathbf{y}_i)$ . **Chú ý:** chúng ta hoàn toàn có thể áp dụng các thuật toán tăng tốc GD như Momentum, AdaGrad,... vào SGD.

#### Ví dụ với bài toán Linear Regression

Với bài toán Linear Regression,  $\theta = \mathbf{w}$ , hàm mất mát tại một điểm dữ liệu là:

$$J(\mathbf{w}; \mathbf{x}_i; y_i) = \frac{1}{2} (\mathbf{x}_i \mathbf{w} - y_i)^2$$

Đạo hàm theo  $\mathbf{w}$  tương ứng là:

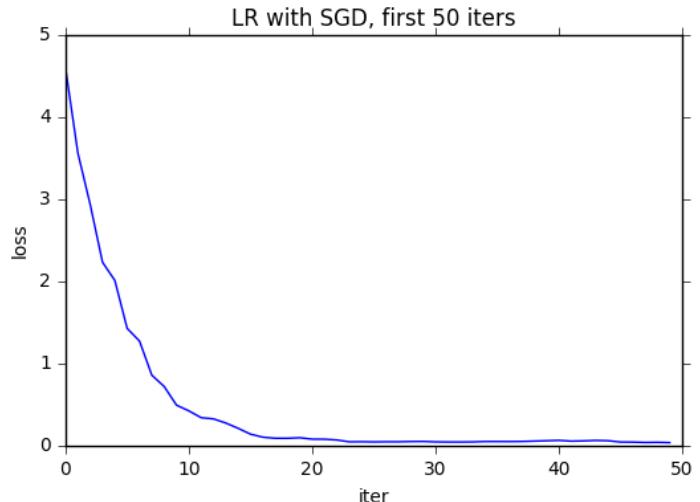
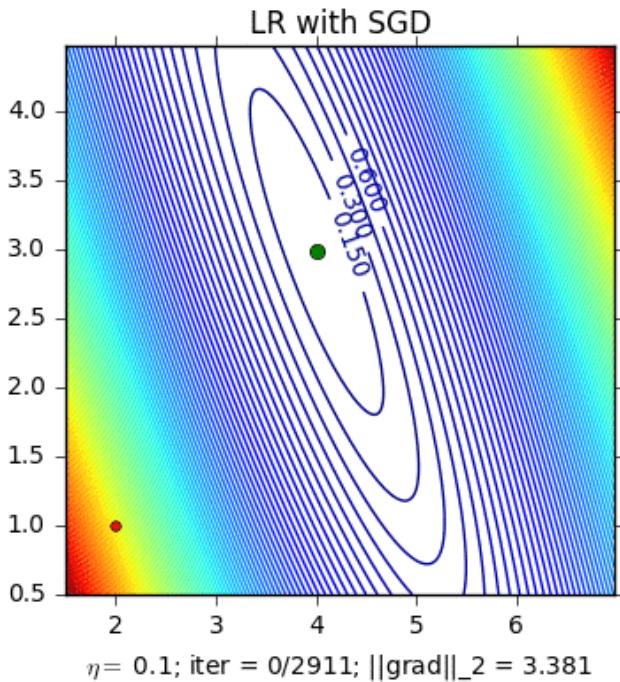
$$\nabla_{\mathbf{w}} J(\mathbf{w}; \mathbf{x}_i; y_i) = \mathbf{x}_i^T (\mathbf{x}_i \mathbf{w} - y_i)$$

Và dưới đây là hàm số trong python để giải Linear Regression theo SGD:

```
# single point gradient
def sgrad(w, i, rd_id):
    true_i = rd_id[i]
    xi = Xbar[true_i, :]
    yi = y[true_i]
    a = np.dot(xi, w) - yi
    return (xi*a).reshape(2, 1)

def SGD(w_init, grad, eta):
    w = [w_init]
    w_last_check = w_init
    iter_check_w = 10
    N = X.shape[0]
    count = 0
    for it in range(10):
        # shuffle data
        rd_id = np.random.permutation(N)
        for i in range(N):
            count += 1
            g = sgrad(w[-1], i, rd_id)
            w_new = w[-1] - eta*g
            w.append(w_new)
            if count%iter_check_w == 0:
                w_this_check = w_new
                if np.linalg.norm(w_this_check - w_last_check)/len(w_init) < 1e-3:
                    return w
                w_last_check = w_this_check
    return w
```

Kết quả được cho như hình dưới đây (với dữ liệu được tạo giống như ở phần 1 (/2017/01/12/gradientdescent/#quay-lai-voi-bai-toan-linear-regression)).



Trái: đường đi của nghiệm với SGD. Phải: giá trị của loss function tại 50 vòng lặp đầu tiên.

Hình bên trái mô tả đường đi của nghiệm. Chúng ta thấy rằng đường đi khá là zigzag chứ không mượt như khi sử dụng GD. Điều này là dễ hiểu vì một điểm dữ liệu không thể đại diện cho toàn bộ dữ liệu được. Tuy nhiên, chúng ta cũng thấy rằng thuật toán hội tụ khá nhanh đến vùng lân cận của nghiệm. Với 1000 điểm dữ liệu, SGD chỉ cần gần 3 epoches (2911 tương ứng với 2911 lần cập nhật, mỗi lần lấy 1 điểm). Nếu so với con số 49 vòng lặp (epoches) như kết quả tốt nhất có được bằng GD, thì kết quả này lợi hơn rất nhiều.

Hình bên phải mô tả hàm mất mát cho toàn bộ dữ liệu sau khi chỉ sử dụng 50 điểm dữ liệu đầu tiên. Mặc dù không mượt, tốc độ hội tụ vẫn rất nhanh.

Thực tế cho thấy chỉ lấy khoảng 10 điểm là ta đã có thể xác định được gần đúng phương trình đường thẳng cần tìm rồi. Đây chính là ưu điểm của SGD - hội tụ rất nhanh.

## 2.3. Mini-batch Gradient Descent

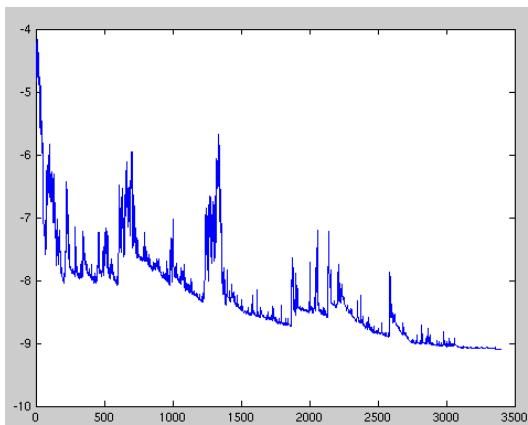
Khác với SGD, mini-batch sử dụng một số lượng  $n$  lớn hơn 1 (nhưng vẫn nhỏ hơn tổng số dữ liệu  $N$  rất nhiều). Giống với SGD, Mini-batch Gradient Descent bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu rồi chia toàn bộ dữ liệu thành các *mini-batch*, mỗi *mini-batch* có  $n$  điểm dữ liệu (trừ mini-batch cuối có thể có ít hơn nếu  $N$  không chia hết cho  $n$ ). Mỗi lần cập nhật, thuật toán này lấy ra một mini-batch để tính toán đạo hàm rồi cập nhật. Công thức có thể viết dưới dạng:

$$\theta = \theta - \eta \nabla_{\theta} J(\theta; \mathbf{x}_{i:i+n}; \mathbf{y}_{i:i+n})$$

Với  $\mathbf{x}_{i:i+n}$  được hiểu là dữ liệu từ thứ  $i$  tới thứ  $i + n - 1$  (theo ký hiệu của Python). Dữ liệu này sau mỗi epoch là khác nhau vì chúng cần được xáo trộn. Một lần nữa, các thuật toán khác cho GD như Momentum, Adagrad, Adadelta,... cũng có thể được áp dụng vào đây.

Mini-batch GD được sử dụng trong hầu hết các thuật toán Machine Learning, đặc biệt là trong Deep Learning. Giá trị  $n$  thường được chọn là khoảng từ 50 đến 100.

Dưới đây là ví dụ về giá trị của hàm mất mát mỗi khi cập nhật tham số  $\theta$  của một bài toán phức tạp hơn.



Hàm mất mát nhảy lên nhảy xuống (fluctuate) sau mỗi lần cập nhật nhưng nhìn chung giảm dần và có xu hướng hội tụ về cuối. (Nguồn: Wikipedia ([https://en.wikipedia.org/wiki/Stochastic\\_gradient\\_descent](https://en.wikipedia.org/wiki/Stochastic_gradient_descent))).

Để có thêm thông tin chi tiết hơn, bạn đọc có thể tìm trong bài viết rất tốt này (<http://sebastianruder.com/optimizing-gradient-descent/index.html#stochasticgradientdescent>).

### 3. Stopping Criteria (điều kiện dừng)

Có một điểm cũng quan trọng mà từ đầu tôi chưa nhắc đến: khi nào thì chúng ta biết thuật toán đã hội tụ và dừng lại?

Trong thực nghiệm, có một vài phương pháp như dưới đây:

1. Giới hạn số vòng lặp: đây là phương pháp phổ biến nhất và cũng dễ đảm bảo rằng chương trình chạy không quá lâu. Tuy nhiên, một nhược điểm của cách làm này là có thể thuật toán dừng lại trước khi đủ gần với nghiệm.
2. So sánh gradient của nghiệm tại hai lần cập nhật liên tiếp, khi nào giá trị này đủ nhỏ thì dừng lại. Phương pháp này cũng có một nhược điểm lớn là việc tính đạo hàm đôi khi trở nên quá phức tạp (ví dụ như khi có quá nhiều dữ liệu), nếu áp dụng phương pháp này thì coi như ta không được lợi khi sử dụng SGD và mini-batch GD.
3. So sánh giá trị của hàm mất mát của nghiệm tại hai lần cập nhật liên tiếp, khi nào giá trị này đủ nhỏ thì dừng lại. Nhược điểm của phương pháp này là nếu tại một thời điểm, đồ thị hàm số có dạng *bangs phảng* tại một khu vực nhưng khu vực đó không chứa điểm local minimum (khu vực này thường được gọi là saddle points), thuật toán cũng dừng lại trước khi đạt giá trị mong muốn.
4. Trong SGD và mini-batch GD, cách thường dùng là so sánh nghiệm sau một vài lần cập nhật. Trong đoạn code Python phía trên về SGD, tôi áp dụng việc so sánh này mỗi khi nghiệm được cập nhật 10 lần. Việc làm này cũng tỏ ra khá hiệu quả.

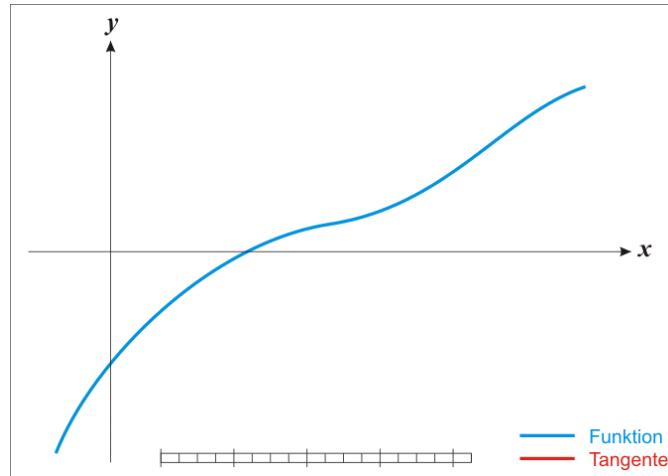
### 4. Một phương pháp tối ưu đơn giản khác: Newton's method

Nhân tiện đang nói về tối ưu, tôi xin giới thiệu một phương pháp nữa có cách giải thích đơn giản: Newton's method. Các phương pháp GD tôi đã trình bày còn được gọi là first-order methods, vì lời giải tìm được dựa trên đạo hàm bậc nhất của hàm số. Newton's method là một second-order method, tức lời giải yêu cầu tính đến đạo hàm bậc hai.

Nhắc lại rằng, cho tới thời điểm này, chúng ta luôn giải phương trình đạo hàm của hàm mất mát bằng 0 để tìm các điểm local minimum. (Và trong nhiều trường hợp, coi nghiệm tìm được là nghiệm của bài toán tìm giá trị nhỏ nhất của hàm mất mát). Có một thuật toán nổi tiếng giúp giải bài toán  $f(x) = 0$ , thuật toán đó có tên là Newton's method.

#### Newton's method cho giải phương trình $f(x) = 0$

Thuật toán Newton's method được mô tả trong hình động minh họa dưới đây:



Hình 3: Minh họa thuật toán Newton's method trong giải phương trình. ( Nguồn: Newton's method - Wikipedia ([https://en.wikipedia.org/wiki/Newton's\\_method](https://en.wikipedia.org/wiki/Newton's_method))).

Ý tưởng giải bài toán  $f(x) = 0$  bằng phương pháp Newton's method như sau. Xuất phát từ một điểm  $x_0$  được cho là gần với nghiệm  $x^*$ . Sau đó vẽ đường tiếp tuyến (mặt tiếp tuyến trong không gian nhiều chiều) với đồ thị hàm số  $y = f(x)$  tại điểm trên đồ thị có hoành độ  $x_0$ . Giao điểm  $x_1$  của đường tiếp tuyến này với trục hoành được xem là gần với nghiệm  $x^*$  hơn. Thuật toán lặp lại với điểm mới  $x_1$  và cứ như vậy đến khi ta được  $f(x_t) \approx 0$ .

Đó là ý nghĩa hình học của Newton's method, chúng ta cần một công thức để có thể dựa vào đó để lập trình. Việc này không quá phức tạp với các bạn thi đại học môn toán ở VN. Thật vậy, phương trình tiếp tuyến với đồ thị của hàm  $f(x)$  tại điểm có hoành độ  $x_t$  là:

$$y = f'(x_t)(x - x_t) + f(x_t)$$

Giao điểm của đường thẳng này với trục  $x$  tìm được bằng cách giải phương trình về phái của biểu thức trên bằng 0, tức là:

$$x = x_t - \frac{f(x_t)}{f'(x_t)} \triangleq x_{t+1}$$

### Newton's method trong bài toán tìm local minimum

Áp dụng phương pháp này cho việc giải phương trình  $f'(x) = 0$  ta có:

$$x_{t+1} = x_t - (f''(x_t))^{-1} f'(x_t)$$

Và trong không gian nhiều chiều với  $\theta$  là biến:

$$\theta = \theta - \mathbf{H}(J(\theta))^{-1} \nabla_{\theta} J(\theta)$$

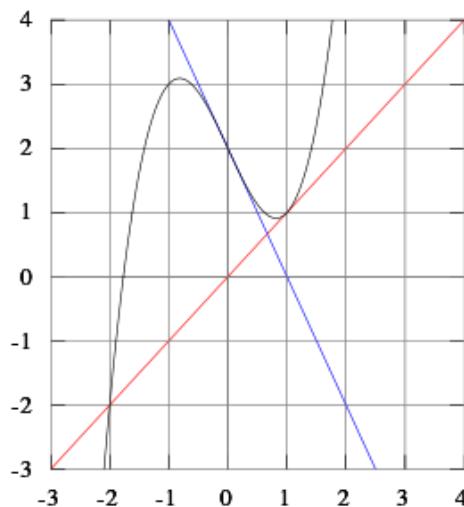
trong đó  $\mathbf{H}(J(\theta))$  là đạo hàm bậc hai của hàm mất mát (còn gọi là Hessian matrix ([https://en.wikipedia.org/wiki/Hessian\\_matrix](https://en.wikipedia.org/wiki/Hessian_matrix))). Biểu thức này là một ma trận nếu  $\theta$  là một vector. Và  $\mathbf{H}(J(\theta))^{-1}$  chính là nghịch đảo của ma trận đó.

### Hạn chế của Newton's method

- Điểm khởi tạo phải rất gần với nghiệm  $x^*$ . Ý tưởng sâu xa hơn của Newton's method là dựa trên khai triển Taylor của hàm số  $f(x)$  tới đạo hàm thứ nhất:

$$0 = f(x^*) \approx f(x_t) + f'(x_t)(x_t - x^*)$$

Từ đó suy ra:  $x^* \approx x_t - \frac{f(x_t)}{f'(x_t)}$ . Một điểm rất quan trọng, khai triển Taylor chỉ đúng nếu  $x_t$  rất gần với  $x^*$ ! Dưới đây là một ví dụ kinh điển trên Wikipedia về việc Newton's method cho một dãy số phân kỳ (divergence).



Hình 4: Nghiệm là một điểm gần -2. Tiếp tuyến của đồ thị hàm số tại điểm có hoành độ bằng 0 cắt trục hoành tại 1, và ngược lại. Trong trường hợp này, Newton's method không bao giờ hội tụ. (Nguồn: Wikipedia ([https://en.wikipedia.org/wiki/Newton's\\_method](https://en.wikipedia.org/wiki/Newton's_method))).

- Nhận thấy rằng trong việc giải phương trình  $f(x) = 0$ , chúng ta có đạo hàm ở mẫu số. Khi đạo hàm này gần với 0, ta sẽ được một đường thẳng song song hoặc gần song song với trục hoành. Ta sẽ hoặc không tìm được giao điểm, hoặc được một giao điểm ở vô cùng. Đặc biệt, khi nghiệm chính là điểm có đạo hàm bằng 0, thuật toán gần như sẽ không tìm được nghiệm!
- Khi áp dụng Newton's method cho bài toán tối ưu trong không gian nhiều chiều, chúng ta cần tính nghịch đảo của Hessian matrix. Khi số chiều và số điểm dữ liệu lớn, đạo hàm bậc hai của hàm mất mát sẽ là một ma trận rất lớn, ảnh hưởng tới cả memory và tốc độ tính toán của hệ thống.

## 5. Kết luận

Qua hai bài viết về Gradient Descent này, tôi hy vọng các bạn đã hiểu và làm quen với một thuật toán tối ưu được sử dụng nhiều nhất trong Machine Learning và đặc biệt là Deep Learning. Còn nhiều biến thể khác khá thú vị về GD (mà rất có thể tôi chưa biết tới), nhưng tôi xin phép được dừng chuỗi bài về GD tại đây và tiếp tục chuyển sang các thuật toán thú vị khác.

Hy vọng bài viết có ích với các bạn.

## 6. Tài liệu tham khảo

[1] Newton's method - Wikipedia ([https://en.wikipedia.org/wiki/Newton's\\_method](https://en.wikipedia.org/wiki/Newton's_method))

[2] An overview of gradient descent optimization algorithms (<http://sebastianruder.com/optimizing-gradient-descent/index.html#stochasticgradientdescent>)

[3] Stochastic Gradient Descent - Wikipedia ([https://en.wikipedia.org/wiki/Stochastic\\_gradient\\_descent](https://en.wikipedia.org/wiki/Stochastic_gradient_descent))

[4] Stochastic Gradient Descent - Andrew Ng (<https://www.youtube.com/watch?v=UfNU3Vhv5CA>)

[5] Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence  $o(1/k^2)$ . Doklady ANSSSR (translated as Soviet.Math.Docl.), vol. 269, pp. 543–547.

Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (<https://www.facebook.com/groups/257768141347267/>) để nhận được câu trả lời sớm hơn.

Bạn đọc có thể ủng hộ blog qua 'Buy me a coffee' (/buymeacoffee/) ở góc trên bên trái của blog.

Tôi đang trong quá trình viết cuốn sách 'Machine Learning cơ bản I', các bạn có thể đặt trước tại đây (/ebook/). Cảm ơn bạn.

« Bài 7: Gradient Descent (phần 1/2) (/2017/01/12/gradientdescent/)

Bài 9: Perceptron Learning Algorithm » (/2017/01/21/perceptron/)

27 Comments tiepvu

Login

Recommend 7 Share

Sort by Best



Join the discussion...



Trần Quốc Hào • 4 months ago

Anh ơi, cho em hỏi, trong NAG, khi ta thay  $J(\theta)$  bằng  $J(\theta - \gamma v(t-1))$ , vận tốc cũng sẽ bị ảnh hưởng, như vậy có cần tính độ giảm của vận tốc ko? Làm sao ta biết dc vận tốc vẫn đủ lớn?

[2 ^](#) | [v](#) • Reply • Share >



đức nam đoàn • 4 months ago

em phát hiện 1 lỗi nhỏ phần code gradient descent momentum:

```
if has_converged(theta_new, grad):
    break
w.append(theta_new)
```

Đoạn này phải sửa thành:

```
theta.append(theta_new)
if has_converged(theta_new, grad):
    break
```

Do ta phải append trước khi check converge, và a đang kí hiệu là theta chứ k phải là w

[1 ^](#) | [v](#) • Reply • Share >



Tiếp Vũ Huu Mod → đức nam đoàn • 4 months ago

Cảm ơn bạn. Tôi đã sửa lại.

[^](#) | [v](#) • Reply • Share >



CTVR • 5 months ago

1. θ là vị trí (tương đương vs quãng đường or độ dịch chuyển) thì sao lại có công thức như hình đc anh Tiệp nhỉ?



2. v(t) là vận tốc tại thời điểm trước đó => v(t-1) chứ anh nhỉ?

[1 ^](#) | [v](#) • Reply • Share >



Cự Nhân Nguyễn Liêm → CTVR • 5 months ago

Chỗ này mình cũng đang thắc mắc, vận tốc là đạo hàm của quãng đường trên t, vậy tại sao quãng đường θ lại có thể làm phép cộng tuyến tính với chính vận tốc v như vậy nhỉ?

[^](#) | [v](#) • Reply • Share >



Tiếp Vũ Huu Mod → Cự Nhân Nguyễn Liêm • 5 months ago

Lúc đó ta coi thời gian là 1 thì sẽ cộng được.

[1 ^](#) | [v](#) • Reply • Share >



Nguyễn Hữu Quang • 9 months ago

great!

[1 ^](#) | [v](#) • Reply • Share >



Nguyễn Văn Đức • 5 days ago

```
N = X.shape[0]
a1 = np.linalg.norm(y, 2)**2/N
b1 = 2*np.sum(X)/N
c1 = np.linalg.norm(X, 2)**2/N
d1 = -2*np.sum(y)/N
e1 = -2*X.T.dot(y)/N
....
```

$Z = a1 + Xg^*2 + b1*Xg*Yg + c1*Yg^*2 + d1*Xg + e1*Yg$

Em không hiểu chỗ này, anh giải thích giúp em với ạ. Tại sao tạo một ma trận plot để vẽ lại dùng công thức như trên ạ?

[^](#) | [v](#) • Reply • Share >



Tiếp Vũ Huu Mod → Nguyễn Văn Đức • 5 days ago

Chỗ đó là công thức của những hình ellipse ở contours. Chỉ có mục đích minh họa. Anh tính bằng tay rồi ra công thức đấy. Cũng không quan trọng lắm đâu.

[^](#) | [v](#) • Reply • Share >



dinhthang • 4 months ago

anh ơi, em muốn hỏi ở trong phần Gradient Descent với Momentum, lượng thông tin lưu trữ trước đó (gamma) mình sẽ tìm nó như thế nào?  
 ^ | v • Reply • Share >



Tiep Vu Huu Mod → dinhthang • 3 months ago

Chỗ này người ta thường chọn gamma = 0.9. Sau đó điều chỉnh một chút cho phù hợp bằng cách chạy xem kết quả có hợp lý không.  
 ^ | v • Reply • Share >



Darwin • 4 months ago

Anh ơi khi e làm NAG thì nó báo lỗi này  
 ValueError: Cannot save animation: no writers are available. Please install ffmpeg to save animations.  
 e cài đặt ffmpeg vẫn báo lỗi này. A có thể giúp e lỗi này ko ,em cảm ơn  
 ^ | v • Reply • Share >



Tiep Vu Huu Mod → Darwin • 3 months ago

Lỗi này là do windows không hỗ trợ tốt phần đó. Anh không dùng windows nên không rõ lắm.

Chỗ animation là chỉ để hiển thị cái hình vẽ lên thôi. Em có thể bỏ đoạn đó đi rồi nhìn kết quả thôi.  
 ^ | v • Reply • Share >



Học trò • 4 months ago

Thầy ơi cho em hỏi, phần kiểm tra hội tụ có đoạn numpy.linalg.norm() có ý nghĩa gì vậy ạ?  
 Em cảm ơn thầy rất nhiều!  
 ^ | v • Reply • Share >



Tiep Vu Huu Mod → Học trò • 4 months ago

Phần này là để kiểm tra điều kiện dừng của thuật toán. Nếu norm của gradient quá nhỏ thì ta sẽ dừng lại.  
 ^ | v • Reply • Share >



Live Tuấn • 5 months ago

Mình có một câu hỏi là giả dụ trong bài toán thực tế, numerical simulation optimization chẳng hạn.  
 Input là những variable vector được design trên parameter space D (giả sử 2 chiều), còn output là một field vector U. Minh muốn tìm optimal variable trên D. Cost function của mình chính là Non Linear least square của  $\|U - U_{target}\|$ . Minh dùng những sample ban đầu simulation (giả sử 10 điểm) để learning cho những sample tiếp theo. Minh nghĩ đây cũng có thể là một dạng Online Learning. Bạn có biết những sample tiếp theo sẽ làm như thế nào ko?  
 Minh cảm ơn bạn.

^ | v • Reply • Share >



Trí Nam Nguyễn → Live Tuấn • 5 months ago

Mình nghĩ bạn đọc về thuật toán RANSAC thử.  
 1 ^ | v • Reply • Share >



Cong-Thanh TRAN • 9 months ago

Anh Tiệp có thể giải thích kỹ hơn tại sao ở SGD ta tính GD cho từng điểm để tìm giá trị hội tụ của w mà giá trị w đó lại là giá trị làm cho hàm mất mát J(w) của toàn tập training là nhỏ nhất vậy ah? Thank anh.  
 ^ | v • Reply • Share >



Tiep Vu Huu Mod → Cong-Thanh TRAN • 9 months ago

Anh không thể giải thích kỹ hơn được nữa :D. Em có thể đọc thêm các tài liệu hoặc video anh cho ở phần Tài liệu tham khảo.  
 Cố gắng tự tìm tòi thêm nhé, đừng dựa vào chỉ một blog hoặc một tài liệu nào.  
 ^ | v • Reply • Share >



Cong-Thanh TRAN → Tiep Vu Huu • 9 months ago

Vâng, lần sau em sẽ đọc thêm phần tham khảo của anh. Thank anh :D  
 ^ | v • Reply • Share >



Dinh Khanh • 9 months ago

Phần code python cho stochastic gradient descent, phần def sgrad: tại sao phải dùng reshape(2,1) bạn nhỉ. Nếu xi có size là 1x2, w là 2x1 thì return ra 1 vector size 2x1. Trong khi mình thử `np.dot(xi,a)` thì lại báo lỗi. Phải dùng `(xi*a).reshape(2,1)`. Cảm ơn bạn đã giải đáp  
 ^ | v • Reply • Share >



Nguyễn Văn Đức → Dinh Khanh • 5 days ago

Mình cũng đang định hỏi câu này :))  
 ^ | v • Reply • Share >



Tiep Vu Huu Mod → Dinh Khanh • 9 months ago

Cảm ơn bạn đã đặt câu hỏi.

xi thực ra không có shape là  $1 \times 2$  mà là  $(2,)$  -- đây là cách index của numpy (Cái này bạn thử check print(xi.shape) là ra). Vì vậy  $a^*xi$  có shape là  $(2,)$  không cùng shape với  $w$  là  $(2, 1)$ . Vì vậy, để cộng vector được thì ta cần  $(a^*xi).reshape(2, 1)$

[^](#) [v](#) • Reply • Share ›



**Cong-Thanh TRAN** → Tiep Vu Huu • 9 months ago

Hi anh, khi em đọc các bài của anh thì phần đọc hiểu code python với em tốn khá thời gian bởi vì em phải tìm xem các methods anh dùng có tác dụng gì output ra sao mới có thể hiểu được đầy đủ từng bước thuật toán làm gì. Thường em phải in ra giá trị của biến sau mỗi bước để hiểu tác dụng của mỗi methods. Thêm nữa một số chỗ cách dùng numpy lib cho array cũng khá rắc rối, có thể do em quen dùng matlab.

Nếu anh có thời gian rảnh, anh có thể làm 1 phần wiki về các câu lệnh numpy python hay dùng trong code của anh được không ah, tiện cho việc tra cứu và đọc hiểu code với những ai chưa quen với python bao giờ.

Thank anh, chúc anh năm mới sức khỏe, nhiều may mắn và thành công.

[^](#) [v](#) • Reply • Share ›



**Tiep Vu Huu** Mod → Cong-Thanh TRAN • 9 months ago

Chào em,

Về python, hay bất cứ ngôn ngữ lập trình nào, nếu em không hiểu từng dòng làm gì thì có một số bước sau đây em có thể thực hiện (những kỹ thuật này là cần thiết cho bất kỳ kỹ sư nào):

1. In kết quả sau mỗi dòng lệnh. Trong python thì là print()
2. debug. Nếu em chưa biết debug trong python thì có thể google.

3. Vào đọc manual của thư viện em đang sử dụng. Ví dụ, nếu em không hiểu np.dot làm gì thì em có thể gõ 'numpy dot' trên Google, sau đó nó sẽ cho em link này: [https://docs.scipy.org/doc/...](https://docs.scipy.org/doc/)

Trên đó là tài liệu gốc, họ viết chắc chắn đầy đủ hơn anh.

Anh không có tham vọng viết cả về python nữa, vì phần này mỗi kỹ sư đều có thể tự học và tra cứu được.

[1](#) [^](#) [v](#) • Reply • Share ›



**Cong-Thanh TRAN** → Tiep Vu Huu • 9 months ago

Vâng, những bước đó em đều làm rồi. Em chỉ thấy nếu có thêm mục tra cứu những hàm hay dùng thì khá là tiện ah.

Thank anh.

[1](#) [^](#) [v](#) • Reply • Share ›



**Tiep Vu Huu** Mod → Cong-Thanh TRAN • 9 months ago

Enjoy việc tra cứu đi nhé. Làm nhiều rồi quen tay thôi. Em vừa học vừa note lại những cái hay dùng, dần dần sẽ nhớ. Đây là cách anh học.

[2](#) [^](#) [v](#) • Reply • Share ›

#### ALSO ON TIEPVU

#### Machine Learning cơ bản

11 comments • 8 months ago

**CTVR** — Em cũng thấy yêu cầu 64 bit trên win. Chắc em chạy với linux vậy.

P/s: Giờ em mới đọc xong loạt bài về convex trên file pdf. Hồi mới vào ...

#### ebook Machine Learning cơ bản

26 comments • 3 months ago

**Xuân Xuân** — Anh ơi, cái email trường em cấp ý, em test nó không cho các mail ngoài gửi vào:"Hi. This is the qmail-send program at vnu.edu.vn.I'm ...

[Subscribe](#) [Add Disqus to your site](#) [Add Disqus Add](#) [Privacy](#)

Total visits:



```

def grad(x):
    return 2*x+ 5*np.cos(x)

def cost(x):
    return x**2 + 5*np.sin(x)

def myGD1(eta, x0):
    x = [x0]
    for it in range(100):
        x_new = x[-1] - eta*grad(x[-1])
        if abs(grad(x_new)) < 1e-3:
            break
        x.append(x_new)
    return (x, it)

```

## Điểm khởi tạo khác nhau

Sau khi có các hàm cần thiết, tôi thử tìm nghiệm với các điểm khởi tạo khác nhau là  $x_0 = -5$  và  $x_0 = 5$ .

```

(x1, it1) = myGD1(.1, -5)
(x2, it2) = myGD1(.1, 5)
print('Solution x1 = %f, cost = %f, obtained after %d iterations'%(x1[-1], cost(x1[-1]), it1))
print('Solution x2 = %f, cost = %f, obtained after %d iterations'%(x2[-1], cost(x2[-1]), it2))

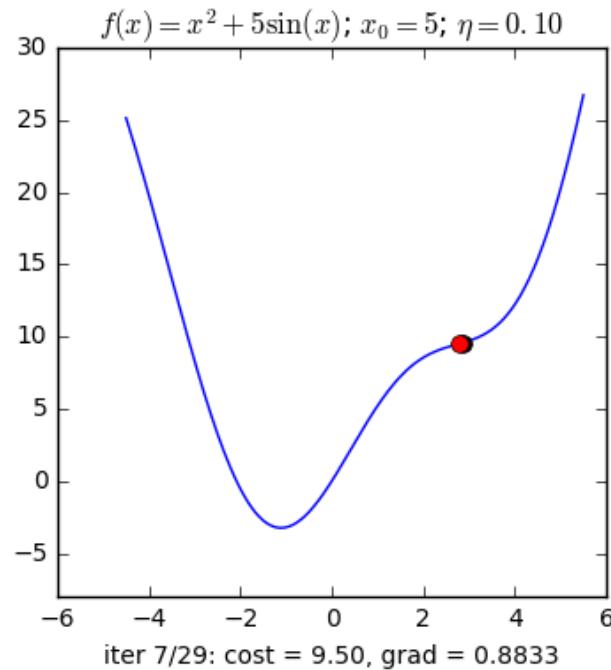
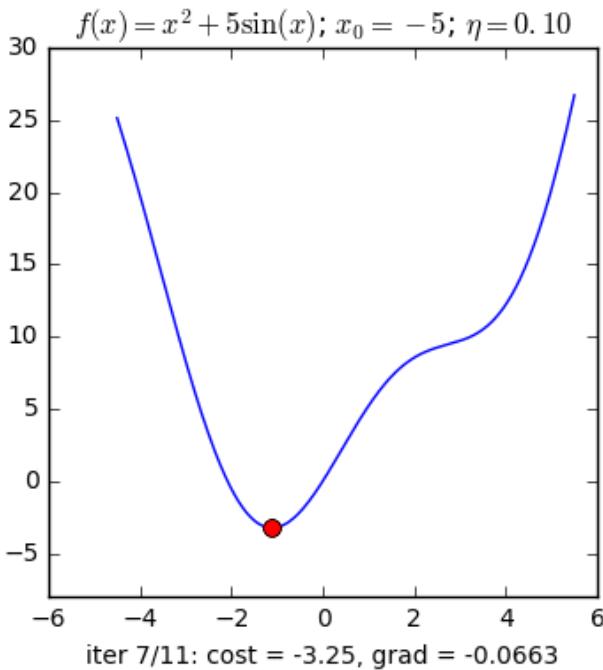
```

```

Solution x1 = -1.110667, cost = -3.246394, obtained after 11 iterations
Solution x2 = -1.110341, cost = -3.246394, obtained after 29 iterations

```

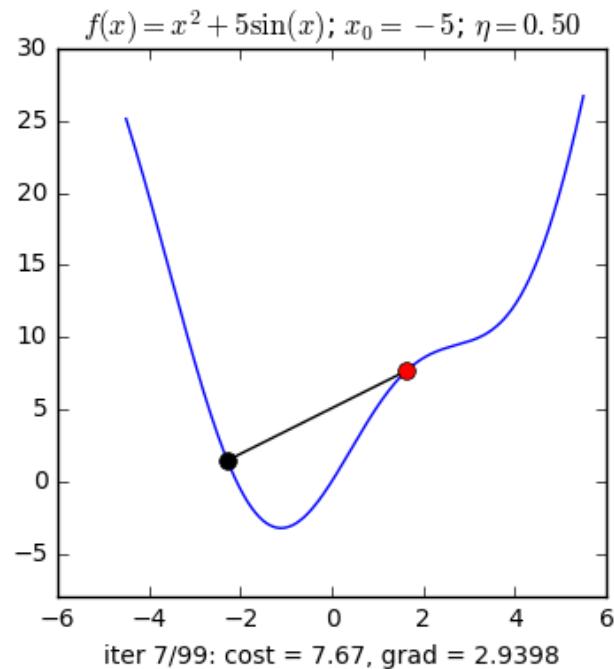
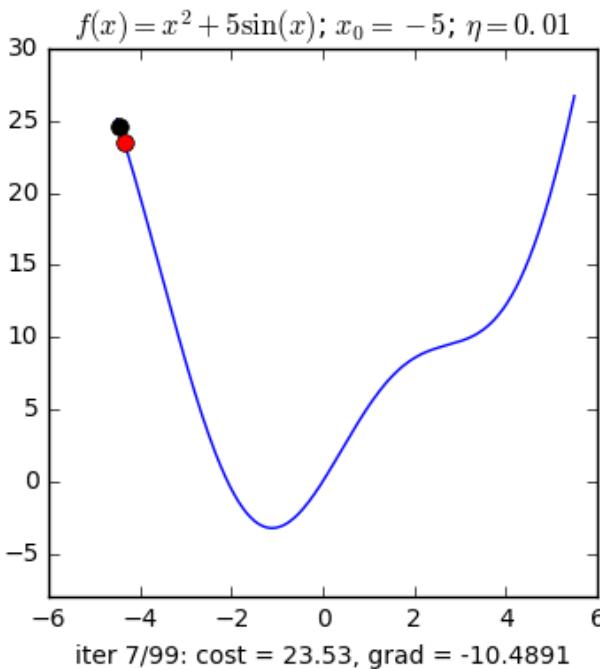
Vậy là với các điểm ban đầu khác nhau, thuật toán của chúng ta tìm được nghiệm gần giống nhau, mặc dù với tốc độ hội tụ khác nhau. Dưới đây là hình ảnh minh họa thuật toán GD cho bài toán này (xem tốt trên Desktop ở chế độ full màn hình).



Từ hình minh họa trên ta thấy rằng ở hình bên trái, tương ứng với  $x_0 = -5$ , nghiệm hội tụ nhanh hơn, vì điểm ban đầu  $x_0$  gần với nghiệm  $x^* \approx -1$  hơn. Hơn nữa, với  $x_0 = 5$  ở hình bên phải, *đường đi* của nghiệm có chứa một khu vực có đạo hàm khá nhỏ gần điểm có hoành độ bằng 2. Điều này khiến cho thuật toán lả cả ở đây khá lâu. Khi vượt qua được điểm này thì mọi việc diễn ra rất tốt đẹp.

## Learning rate khác nhau

Tốc độ hội tụ của GD không những phụ thuộc vào điểm khởi tạo ban đầu mà còn phụ thuộc vào *learning rate*. Dưới đây là một ví dụ với cùng điểm khởi tạo  $x_0 = -5$  nhưng learning rate khác nhau:



Ta quan sát thấy hai điều:

1. Với *learning rate* nhỏ  $\eta = 0.01$ , tốc độ hội tụ rất chậm. Trong ví dụ này tôi chọn tối đa 100 vòng lặp nên thuật toán dừng lại trước khi tới đích, mặc dù đã rất gần. Trong thực tế, khi việc tính toán trở nên phức tạp, *learning rate* quá thấp sẽ ảnh hưởng tới tốc độ của thuật toán rất nhiều, thậm chí không bao giờ tới được đích.
2. Với *learning rate* lớn  $\eta = 0.5$ , thuật toán tiến rất nhanh tới gần đích sau vài vòng lặp. Tuy nhiên, thuật toán không hội tụ được vì *bước nhảy* quá lớn, khiến nó cứ *quắn quanh* ở đích.

Việc lựa chọn *learning rate* rất quan trọng trong các bài toán thực tế. Việc lựa chọn giá trị này phụ thuộc nhiều vào từng bài toán và phải làm một vài thí nghiệm để chọn ra giá trị tốt nhất. Ngoài ra, tùy vào một số bài toán, GD có thể làm việc hiệu quả hơn bằng cách chọn ra *learning rate* phù hợp hoặc chọn *learning rate* khác nhau ở mỗi vòng lặp. Tôi sẽ quay lại vấn đề này ở phần 2.

### 3. Gradient Descent cho hàm nhiều biến

Giả sử ta cần tìm global minimum cho hàm  $f(\theta)$  trong đó  $\theta$  (*theta*) là một vector, thường được dùng để ký hiệu tập hợp các tham số của một mô hình cần tối ưu (trong Linear Regression thì các tham số chính là hệ số  $w$ ). Đạo hàm của hàm số đó tại một điểm  $\theta$  bất kỳ được ký hiệu là  $\nabla_{\theta}f(\theta)$  (hình tam giác ngược đọc là *nabla*). Tương tự như hàm 1 biến, thuật toán GD cho hàm nhiều biến cũng bắt đầu bằng một điểm dự đoán  $\theta_0$ , sau đó, ở vòng lặp thứ  $t$ , quy tắc cập nhật là:

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta}f(\theta_t)$$

Hoặc viết dưới dạng đơn giản hơn:  $\theta = \theta - \eta \nabla_{\theta}f(\theta)$ .

Quy tắc cần nhớ: **luôn luôn đi ngược hướng với đạo hàm**.

Việc tính toán đạo hàm của các hàm nhiều biến là một kỹ năng cần thiết. Một vài đạo hàm đơn giản có thể được tìm thấy ở đây (/math/#bangcac-dao-ham-co-ban).

### Quay lại với bài toán Linear Regression

Trong mục này, chúng ta quay lại với bài toán Linear Regression (/2016/12/28/linearregression/) và thử tối ưu hàm mất mát của nó bằng thuật toán GD.

Hàm mất mát của Linear Regression là:

$$\mathcal{L}(w) = \frac{1}{2N} \|y - \bar{X}w\|_2^2$$

**Chú ý:** hàm này có khác một chút so với hàm tôi nêu trong bài Linear Regression (/2016/12/28/linearregression/). Mẫu số có thêm  $N$  là số lượng dữ liệu trong training set. Việc lấy trung bình cộng của lỗi này nhằm giúp tránh trường hợp hàm mất mát và đạo hàm có giá trị là một số rất lớn, ảnh hưởng tới độ chính xác của các phép toán khi thực hiện trên máy tính. Về mặt toán học, nghiệm của hai bài toán là như nhau.

Đạo hàm của hàm mất mát là:

$$\nabla_w \mathcal{L}(w) = \frac{1}{N} \bar{X}^T (\bar{X}w - y) \quad (1)$$

## Sau đây là ví dụ trên Python và một vài lưu ý khi lập trình

Load thư viện

```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
np.random.seed(2)
```

Tiếp theo, chúng ta tạo 1000 điểm dữ liệu được chọn gần với đường thẳng  $y = 4 + 3x$ , hiển thị chúng và tìm nghiệm theo công thức:

```
X = np.random.rand(1000, 1)
y = 4 + 3 * X + .2*np.random.randn(1000, 1) # noise added

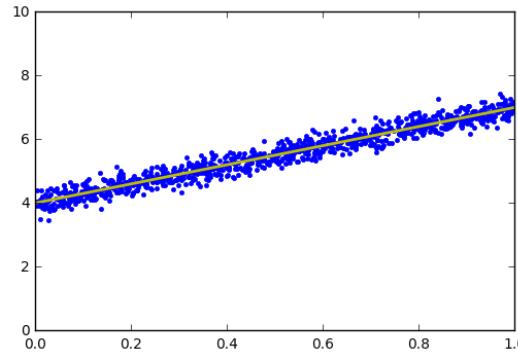
# Building Xbar
one = np.ones((X.shape[0],1))
Xbar = np.concatenate((one, X), axis = 1)

A = np.dot(Xbar.T, Xbar)
b = np.dot(Xbar.T, y)
w_lr = np.dot(np.linalg.pinv(A), b)
print('Solution found by formula: w = ',w_lr.T)

# Display result
w = w_lr
w_0 = w[0][0]
w_1 = w[1][0]
x0 = np.linspace(0, 1, 2, endpoint=True)
y0 = w_0 + w_1*x0

# Draw the fitting line
plt.plot(X.T, y.T, 'b.')      # data
plt.plot(x0, y0, 'y', linewidth = 2) # the fitting line
plt.axis([0, 1, 0, 10])
plt.show()
```

Solution found by formula: w = [[ 4.00305242 2.99862665]]



Đường thẳng tìm được là đường có màu vàng có phương trình  $y \approx 4 + 2.998x$ .

Tiếp theo ta viết đạo hàm và hàm mất mát:

```
def grad(w):
    N = Xbar.shape[0]
    return 1/N * Xbar.T.dot(Xbar.dot(w) - y)

def cost(w):
    N = Xbar.shape[0]
    return .5/N*np.linalg.norm(y - Xbar.dot(w), 2)**2;
```

### Kiểm tra đạo hàm

Việc tính đạo hàm của hàm nhiều biến thông thường khá phức tạp và rất dễ mắc lỗi, nếu chúng ta tính sai đạo hàm thì thuật toán GD không thể chạy đúng được. Trong thực nghiệm, có một cách để kiểm tra liệu đạo hàm tính được có chính xác không. Cách này dựa trên định nghĩa của đạo hàm (cho hàm 1 biến):

$$f'(x) = \lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - f(x)}{\varepsilon}$$

Một cách thường được sử dụng là lấy một giá trị  $\varepsilon$  rất nhỏ, ví dụ  $10^{-6}$ , và sử dụng công thức:

$$f'(x) \approx \frac{f(x + \varepsilon) - f(x - \varepsilon)}{2\varepsilon} \quad (2)$$

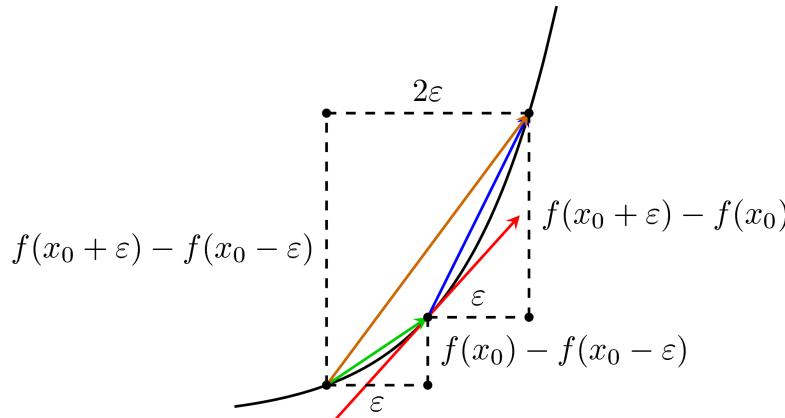
Cách tính này được gọi là *numerical gradient*.

**Câu hỏi:** Tại sao công thức xấp xỉ hai phía trên đây lại được sử dụng rộng rãi, sao không sử dụng công thức xấp xỉ đạo hàm bên phải hoặc bên trái?

Có hai các giải thích cho vấn đề này, một bằng hình học, một bằng giải tích.

### Giải thích bằng hình học

Quan sát hình dưới đây:



Trong hình, vector màu đỏ là đạo hàm *chính xác* của hàm số tại điểm có hoành độ bằng  $x_0$ . Vector màu xanh lam (có vẻ là hơi tím sau khi convert từ .pdf sang .png) thể hiện cách xấp xỉ đạo hàm phía phải. Vector màu xanh lục thể hiện cách xấp xỉ đạo hàm phía trái. Vector màu nâu thể hiện cách xấp xỉ đạo hàm hai phía. Trong ba vector xấp xỉ đó, vector xấp xỉ hai phía màu nâu là gần với vector đỏ nhất nếu xét theo hướng.

Sự khác biệt giữa các cách xấp xỉ còn lớn hơn nữa nếu tại điểm x, hàm số bị *bend* mạnh hơn. Khi đó, xấp xỉ trái và phải sẽ khác nhau rất nhiều. Xấp xỉ hai bên sẽ  *ổn định* hơn.

### Giải thích bằng giải tích

Chúng ta cùng quay lại một chút với Giải tích I năm thứ nhất đại học: Khai triển Taylor (<http://mathworld.wolfram.com/TaylorSeries.html>).

Với  $\varepsilon$  rất nhỏ, ta có hai xấp xỉ sau:

$$f(x + \varepsilon) \approx f(x) + f'(x)\varepsilon + \frac{f''(x)}{2}\varepsilon^2 + \dots$$

và:

$$f(x - \varepsilon) \approx f(x) - f'(x)\varepsilon + \frac{f''(x)}{2}\varepsilon^2 - \dots$$

Từ đó ta có:

$$\frac{f(x + \varepsilon) - f(x)}{\varepsilon} \approx f'(x) + \frac{f''(x)}{2}\varepsilon + \dots = f'(x) + O(\varepsilon) \quad (3)$$

$$\frac{f(x + \varepsilon) - f(x - \varepsilon)}{2\varepsilon} \approx f'(x) + \frac{f^{(3)}(x)}{6}\varepsilon^2 + \dots = f'(x) + O(\varepsilon^2) \quad (4)$$

trong đó  $O()$  là Big O notation ([https://en.wikipedia.org/wiki/Big\\_O\\_notation](https://en.wikipedia.org/wiki/Big_O_notation)).

Từ đó, nếu xấp xỉ đạo hàm bằng công thức (3) (xấp xỉ đạo hàm phải), sai số sẽ là  $O(\varepsilon)$ . Trong khi đó, nếu xấp xỉ đạo hàm bằng công thức (4) (xấp xỉ đạo hàm hai phía), sai số sẽ là  $O(\varepsilon^2) \ll O(\varepsilon)$  nếu  $\varepsilon$  nhỏ.

Cả hai cách giải thích trên đây đều cho chúng ta thấy rằng, xấp xỉ đạo hàm hai phía là xấp xỉ tốt hơn.

### Với hàm nhiều biến

Với hàm nhiều biến, công thức (2) được áp dụng cho từng biến khi các biến khác cố định. Cách tính này thường cho giá trị khá chính xác. Tuy nhiên, cách này không được sử dụng để tính đạo hàm vì độ phức tạp quá cao so với cách tính trực tiếp. Khi so sánh đạo hàm này với đạo hàm chính xác tính theo công thức, người ta thường giảm số chiều dữ liệu và giảm số điểm dữ liệu để thuận tiện cho tính toán. Một khi đạo hàm tính được rất gần với

*numerical gradient*, chúng ta có thể tự tin rằng đạo hàm tính được là chính xác.

Dưới đây là một đoạn code đơn giản để kiểm tra đạo hàm và có thể áp dụng với một hàm số (của một vector) bất kỳ với cost và grad đã tính ở phía trên.

```
def numerical_grad(w, cost):
    eps = 1e-4
    g = np.zeros_like(w)
    for i in range(len(w)):
        w_p = w.copy()
        w_n = w.copy()
        w_p[i] += eps
        w_n[i] -= eps
        g[i] = (cost(w_p) - cost(w_n))/(2*eps)
    return g

def check_grad(w, cost, grad):
    w = np.random.rand(w.shape[0], w.shape[1])
    grad1 = grad(w)
    grad2 = numerical_grad(w, cost)
    return True if np.linalg.norm(grad1 - grad2) < 1e-6 else False

print('Checking gradient...', check_grad(np.random.rand(2, 1), cost, grad))
```

Checking gradient... True

(Với các hàm số khác, bạn đọc chỉ cần viết lại hàm grad và cost ở phần trên rồi áp dụng đoạn code này để kiểm tra đạo hàm. Nếu hàm số là hàm của một ma trận thì chúng ta thay đổi một chút trong hàm numerical\_grad, tôi hy vọng không quá phức tạp).

Với bài toán Linear Regression, cách tính đạo hàm như trong (1) phía trên được coi là đúng vì sai số giữa hai cách tính là rất nhỏ (nhỏ hơn  $10^{-6}$ ). Sau khi có được đạo hàm chính xác, chúng ta viết hàm cho GD:

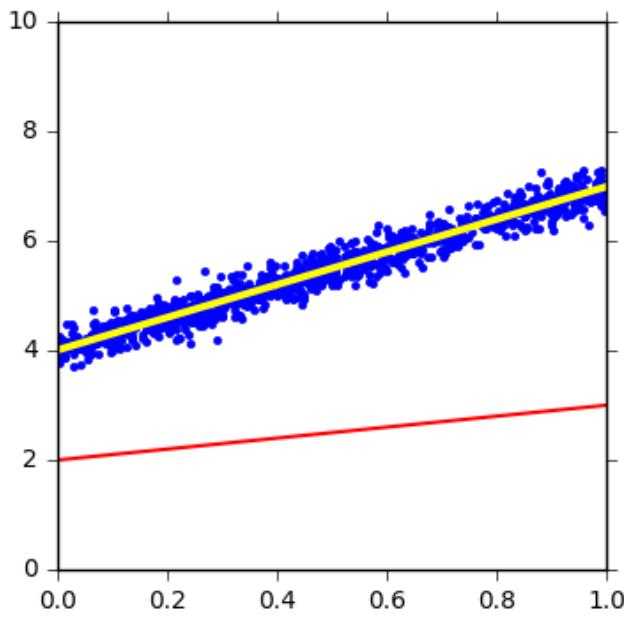
```
def myGD(w_init, grad, eta):
    w = [w_init]
    for it in range(100):
        w_new = w[-1] - eta*grad(w[-1])
        if np.linalg.norm(grad(w_new))/len(w_new) < 1e-3:
            break
        w.append(w_new)
    return (w, it)

w_init = np.array([[2], [1]])
(w1, it1) = myGD(w_init, grad, 1)
print('Solution found by GD: w = ', w1[-1].T, '\nafter %d iterations.' %(it1+1))
```

Solution found by GD: w = [[ 4.01780793 2.97133693]] ,  
after 49 iterations.

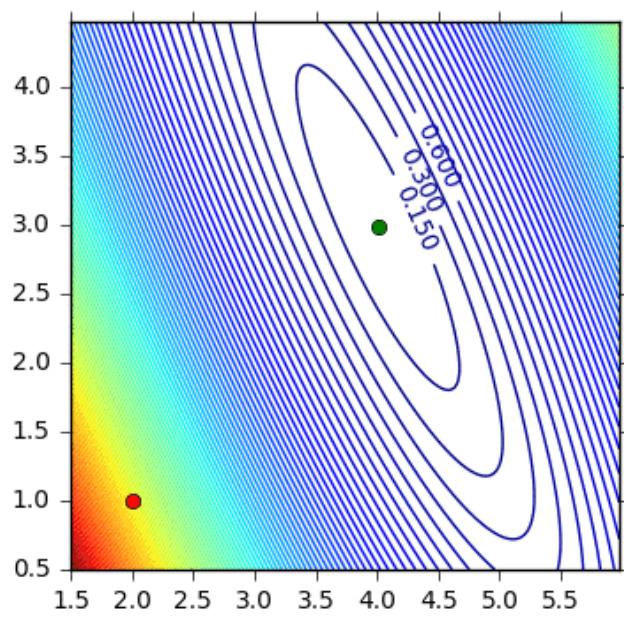
Sau 49 vòng lặp, thuật toán đã hội tụ với một nghiệm khá gần với nghiệm tìm được theo công thức.

Dưới đây là hình động minh họa thuật toán GD.



$\eta = 1$ ; iter = 0/49;  $\|\text{grad}\|_2 = 3.381$

Trong hình bên trái, các đường thẳng màu đỏ là nghiệm tìm được sau mỗi vòng lặp.



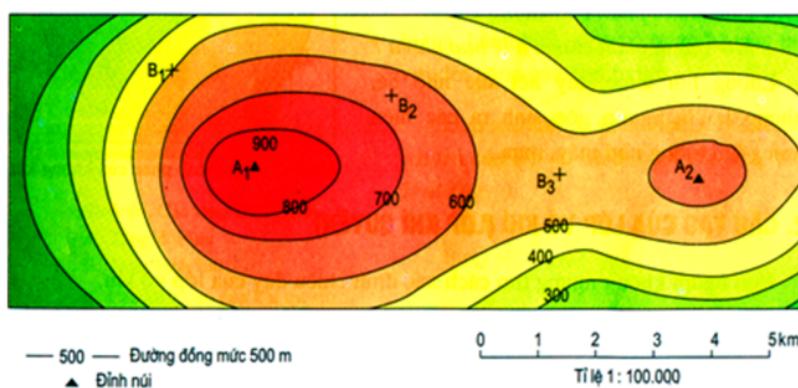
$\eta = 1$ ; iter = 0/49;  $\|\text{grad}\|_2 = 3.381$

Trong hình bên phải, tôi xin giới thiệu một thuật ngữ mới: *đường đồng mức*.

### Đường đồng mức (level sets)

Với đồ thị của một hàm số với hai biến đầu vào cần được vẽ trong không gian ba chiều, nhiều khi chúng ta khó nhìn được nghiệm có khoảng tọa độ bao nhiêu. Trong toán tối ưu, người ta thường dùng một cách vẽ sử dụng khái niệm *đường đồng mức* (level sets).

Nếu các bạn để ý trong các bản đồ tự nhiên, để miêu tả độ cao của các dãy núi, người ta dùng nhiều đường cong kín bao quanh nhau như sau:



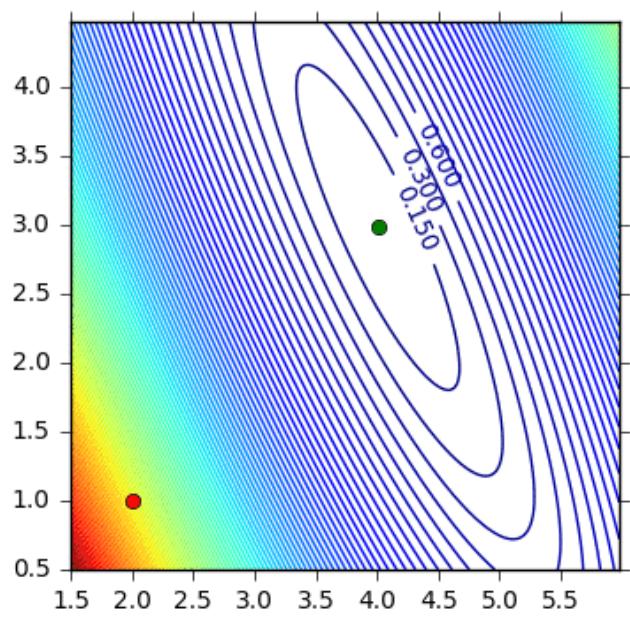
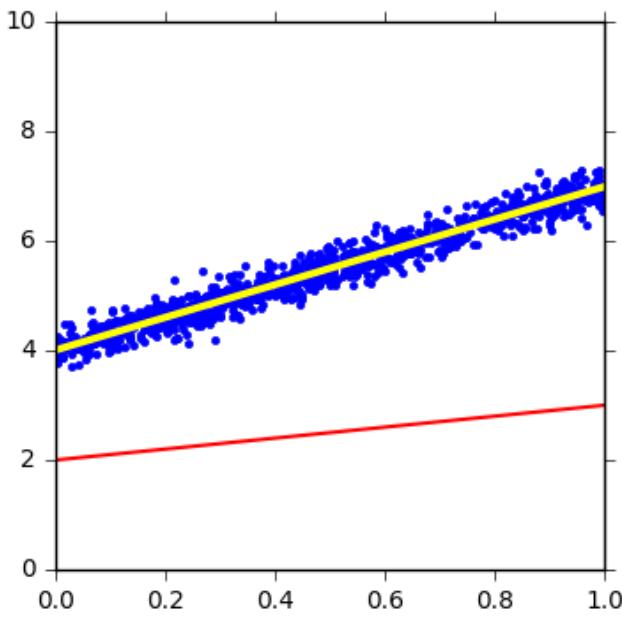
Ví dụ về đường đồng mức trong các bản đồ tự nhiên. (Nguồn: Địa lý 6: Đường đồng mức là những đường như thế nào? (<http://vforum.vn/diendan/showthread.php?90166-Dia-ly-6-Duong-dong-muc-la-nhung-duong-nhu-the-nao->))

Các vòng nhỏ màu đỏ hơn thể hiện các điểm ở trên cao hơn.

Trong toán tối ưu, người ta cũng dùng phương pháp này để thể hiện các bề mặt trong không gian hai chiều.

Quay trở lại với hình minh họa thuật toán GD cho bài toán Linear Regression bên trên, hình bên phải là hình biểu diễn các level sets. Tức là tại các điểm trên cùng một vòng, hàm mất mát có giá trị như nhau. Trong ví dụ này, tôi hiển thị giá trị của hàm số tại một số vòng. Các vòng màu xanh có giá trị thấp, các vòng tròn màu đỏ phía ngoài có giá trị cao hơn. Điểm này khác một chút so với đường đồng mức trong tự nhiên là các vòng bên trong thường thể hiện một thung lũng hơn là một đỉnh núi (vì chúng ta đang đi tìm giá trị nhỏ nhất).

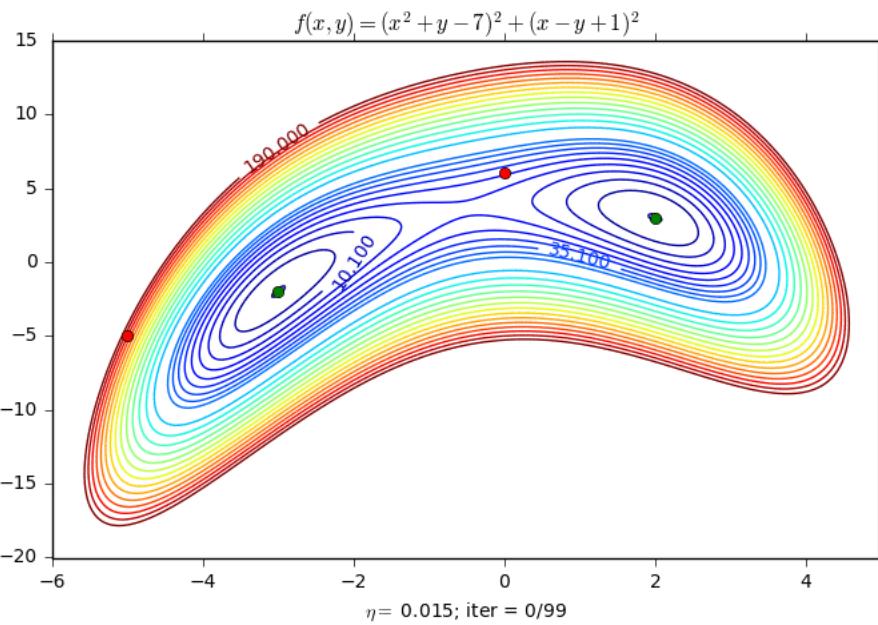
Tôi thử với *learning rate* nhỏ hơn, kết quả như sau:



Tốc độ hội tụ đã chậm đi nhiều, thậm chí sau 99 vòng lặp, GD vẫn chưa tới gần được nghiệm tốt nhất. Trong các bài toán thực tế, chúng ta cần nhiều vòng lặp hơn 99 rất nhiều, vì số chiều và số điểm dữ liệu thường là rất lớn.

#### 4. Một ví dụ khác

Để kết thúc phần 1 của Gradient Descent, tôi xin nêu thêm một ví dụ khác.



Hàm số  $f(x, y) = (x^2 + y - 7)^2 + (x - y + 1)^2$  có hai điểm local minimum màu xanh lục tại  $(2, 3)$  và  $(-3, -2)$ , và chúng cũng là hai điểm global minimum. Trong ví dụ này, tùy vào điểm khởi tạo mà chúng ta thu được các nghiệm cuối cùng khác nhau.

#### 5. Thảo luận

Dựa trên GD, có rất nhiều thuật toán phức tạp và hiệu quả hơn được thiết kế cho những loại bài toán khác nhau. Vì bài này đã đủ dài, tôi xin phép dừng lại ở đây. Mọi các bạn đón đọc bài Gradient Descent phần 2 với nhiều kỹ thuật nâng cao hơn.

#### 6. Tài liệu tham khảo

1. An overview of gradient descent optimization algorithms (<http://sebastianruder.com/optimizing-gradient-descent/>)
2. An Interactive Tutorial on Numerical Optimization (<http://www.benfrederickson.com/numerical-optimization/>)

3. Gradient Descent by Andrew NG (<https://www.youtube.com/watch?v=eikJboPQDT0>)

Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (<https://www.facebook.com/groups/257768141347267/>) để nhận được câu trả lời sớm hơn.

Bạn đọc có thể ủng hộ blog qua 'Buy me a coffee' (/buymeacoffee/) ở góc bên trái của blog.

Tôi đang trong quá trình viết cuốn sách 'Machine Learning cơ bản I', các bạn có thể đặt trước tại đây (/ebook). Cảm ơn bạn.

« Bài 6: K-nearest neighbors (/2017/01/08/knn/)

Bài 8: Gradient Descent (phần 2/2) » (/2017/01/16/gradientdescent2/)

34 Comments tiepvu

 Login ▾

 Recommend 18  Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name



Nguyễn Văn Đức • 6 days ago

Bài viết hay quá anh ơi!!!! Cảm ơn anh nhiều <3  
1 ^ | v · Reply · Share >



Quoc Tc • 3 months ago

Bài viết rất hay!, cảm ơn anh nhiều.  
Có chỗ này em chưa hiểu lắm:  
Phản ứng hàm của bài Linear Regression. Anh /N với N là số lượng data points. Chỗ này nếu mình normalize data để cho phản ứng hàm không quá lớn được ko a? giống như anh đã làm trong cái vd ở bài Linear Regression?  
1 ^ | v · Reply · Share >



Tiep Vu Huu Mod → Quoc Tc • 3 months ago

Tốt nhất là nên làm cả hai, tức cả normalize data và lấy trung bình.  
^ | v · Reply · Share >



Quoc Tc → Tiep Vu Huu • 3 months ago

Ok, thank anh.  
^ | v · Reply · Share >



thuong • 8 months ago

Các bài viết trong blog rất dễ hiểu, cảm ơn anh rất nhiều.  
1 ^ | v · Reply · Share >



Duong Nguyen Anh Khoa • 8 months ago

Chào anh, em xem các bài viết của anh thì rất thích các hình ảnh minh họa mà anh làm, đặc biệt là ảnh động. Anh có thể cho em biết anh làm như thế nào không ạ? Em đã làm thử bằng matplotlib trên python nhưng mỗi cửa sổ là 1 ảnh tĩnh. Em đoán rằng có cách làm nó liên tục và xuất ra được ảnh gif mà vẫn chưa làm được.:D  
Mong anh ra thêm nhiều bài viết hay nữa, em cảm ơn!  
1 ^ | v · Reply · Share >



Tiep Vu Huu Mod → Duong Nguyen Anh Khoa • 8 months ago

Cảm ơn em.

Em xem ví dụ anh làm hình động ở đây nhé:

<https://tiepyupsu.github.io...>

1 ^ | v · Reply · Share >



Thanh Nguyen → Tiep Vu Huu • 8 months ago

cung giong voi ban tren. e cung muon lam anh dong giong nhu anh trong bai. Nhung bi loi la "Cannot save animation: no writers are available. Please install ffmpeg to save animations.". E da cai ffmpeg vao trong may va thu cac cach khac nhu chi dan tren mang nhung van ko thanh cong. Hy vong a noi cu the hon 1 chut nen cai nhu the nao de dc anh gif nhu the ah.  
E xloi vi may tinh e hien khong the go bang tieng viet co dau dc.  
^ | v · Reply · Share >



**Tiep Vu Huu** Mod → Thanh Nguyen • 8 months ago

Anh dùng Ubuntu, ko gặp khó khăn nào cả. Windows thì anh ko rõ.

^ | v • Reply • Share >



**Thanh Nguyen** Tiep Vu Huu • 8 months ago

dạ vâng. sau một hồi mày mò thì e cũng thành công.

Cho các bạn dùng windows thì sau khi cài ffmpeg thành công, nếu lưu file gif mà mắc lỗi thì có thể đầu tiên lưu dưới dạng .mp4 sau đó dùng ffmpeg convert sang ảnh gif bằng 1 dòng lệnh đơn giản ffmpeg -i path\animation.mp4 path\animation.gif

Nếu dùng python thi

```
import os
```

```
os.system("ffmpeg -i path\animation.mp4 path\animation.gif")
```

^ | v • Reply • Share >



**Quốc Nguyễn Đình** • 8 months ago

anh oi cho em hỏi sao cái chỗ kiểm tra đạo hàm ý, chỗ pt (2), sao anh k lấy là  $(f(x+e)-f(x))/e$ ?  
bài anh hay lắm ạ, em cảm ơn!

1 ^ | v • Reply • Share >



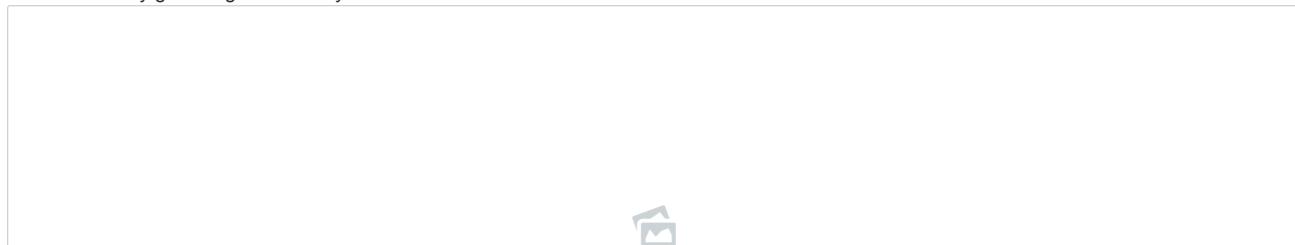
**Tiep Vu Huu** Mod → Quốc Nguyễn Đình • 8 months ago

Công thức đó được gọi là xấp xỉ đạo hàm theo hai phía. Nếu như em nói thi là xấp xỉ đạo hàm một phía.

Đạo hàm hai phía được cho là ổn định hơn đạo hàm một phía. Lấy một ví dụ mà đạo hàm tại 1 điểm  $x_0$  thay đổi \*cực nhanh\* với đạo hàm phía trái nhỏ, đạo hàm phía phải lớn. Nếu chỉ lấy đạo hàm phía trái hoặc phía phải để xấp xỉ thì sẽ không tốt bằng cách lấy xấp xỉ theo cả hai phía.

Để có thể dễ hình dung hơn, anh lấy một hàm mà tại một điểm, đồ thị có nó có dạng  $\_$  / (tất nhiên là nó cong ở chỗ gấp khúc đó để có thể lấy đạo hàm được). Như thế nếu em lấy đạo hàm phía phải thì nó sẽ xấp xỉ bằng 1. Nếu lấy đạo hàm phía trái thì nó xấp xỉ bằng 0. Tốt nhất là lấy đạo hàm hai phía, ổn định hơn. :)

Em xem thêm lý giải bằng hình vẽ này nhé:



see more

^ | v • Reply • Share >



**Thai** • 9 months ago

Bài viết rất chất lượng, cảm ơn anh.

1 ^ | v • Reply • Share >



**Tiep Vu Huu** Mod → Thai • 9 months ago

Cảm ơn bạn đã theo dõi blog.

^ | v • Reply • Share >



**Dat Than** • 15 days ago

Bài viết của anh hay quá! Cảm ơn anh rất nhiều!

^ | v • Reply • Share >



**Bao anh** • 24 days ago

a cho e hỏi phần tính numerical grad, vì sao phải gán eps cho từng element của  $g_p$ ,  $g_n$  qua từng vòng lặp mà ko phải là cộng eps cho tất cả element của  $g_p$ ,  $g_n$  1 lúc luon ạ

^ | v • Reply • Share >



**Tiep Vu Huu** Mod → Bao anh • 24 days ago

Em thử nghĩ một chút nhé. Nếu em làm như thế thì kết quả  $(g_p - g_n)/2eps$  sẽ là gì? Gợi ý nhé, kết quả này là một số vô hướng, trong khi đạo hàm theo vector hoặc ma trận phải trả về một vector hoặc ma trận cùng kích thước.

^ | v • Reply • Share >



**Bao anh** • 24 days ago

Chào anh, e cảm ơn bài viết của a.

Cho e hỏi e bị lỗi trong hàm numerical grad, khi cho ma trận cộng vs eps:

```
"TypeError: Cannot cast ufunc add output from dtype('float64') to dtype('int64') with casting rule 'same_kind'"
```

^ | v • Reply • Share >



**Tiep Vu Huu** Mod → Bao anh • 24 days ago  
Lỗi nói rõ rồi đó em. Lỗi là do type khác nhau.  
^ | v • Reply • Share

**Bao anh** Tiep Vu Huu • 24 days ago  
e giai quyết được rồi. Thank a  
1 ^ | v • Reply • Share



**livw** • 3 months ago  
Anh ơi, cho em hỏi, tại sao mình lại cộng vào theta một lượng có trị tuyệt đối tỉ lệ thuận với đạo hàm à?  
^ | v • Reply • Share



**Tiep Vu Huu** Mod → livw • 3 months ago  
Cái này là có được một trực quan từ hình đầu tiên. Càng xa điểm tối ưu màu xanh lục thì đạo hàm càng lớn. Mà mình muốn tiến gần đến điểm màu xanh lục nên mình cần đi ngược lại một lượng tỉ lệ thuận với đạo hàm. Tức nếu đạo hàm càng lớn thì càng xa, càng phải đi nhanh hơn.  
^ | v • Reply • Share

**livw** Tiep Vu Huu • 3 months ago  
À em hiểu rồi. Tại em đang hiểu là nếu đạo hàm đang dương (âm) thì trong suốt quá trình hội tụ chỉ được đi về bên trái (phải), và delta tỉ lệ thuận với đạo hàm sẽ giúp thỏa mãn điều này, còn thực tế thì trong quá trình hội tụ, đạo hàm có thể đổi dấu.  
^ | v • Reply • Share



**Vinh Tống** • 3 months ago  
Anh Tiệp cho em hỏi dòng code:  
np.random.seed(2), có ý nghĩa gì ạ?  
em có thể thay đổi số 2 bằng một số khác được không ạ?  
^ | v • Reply • Share



**Tiep Vu Huu** Mod → Vinh Tống • 3 months ago  
Google sẽ thấy ngay ;)  
^ | v • Reply • Share



**Darwin** • 4 months ago  
A = np.dot(Xbar.T, Xbar)  
  
b = np.dot(Xbar.T, y)  
  
w\_lr = np.dot(np.linalg.pinv(A), b)  
  
print('Solution found by formula: w = ', w\_lr.T)  
A ơi cho em hỏi đoạn này mình tính theo công thức nào vậy a?  
^ | v • Reply • Share



**Tiep Vu Huu** Mod → Darwin • 4 months ago  
Cái này là công thức theo bài 3 nhé bạn:  
<http://machinelearningcaban...>  
^ | v • Reply • Share



**Nguyễn Bình An** • 5 months ago  
Anh ơi cho em hỏi ở phần trên tại công thức tính grad thì mình đã chia cho N rồi, tại sao phần thuật toán GD ở cuối cùng mình vẫn phải chia cho (len(w\_new)) nữa ạ ?  
np.linalg.norm(grad(w\_new))/len(w\_new) < 1e-3:

Em không hiểu, mong anh giải thích ạ, em cảm ơn !  
^ | v • Reply • Share



**Tiep Vu Huu** Mod → Nguyễn Bình An • 4 months ago  
Chỗ này chỉ để đảm bảo là trung bình của trị tuyệt đối grad tại từng thành phần là nhỏ hơn 1e-3.  
Nếu không chia cho len(w\_new) thì với ma trận lớn, mọi thành phần đều nhỏ nhưng tổng của chúng có thể lớn. Vậy nên ta cần tính trung bình.  
^ | v • Reply • Share

**Nguyễn Bình An** Tiep Vu Huu • 4 months ago  
em cảm ơn ạ, bây giờ em đã hiểu  
^ | v • Reply • Share



**Hoang Ha** • 6 months ago  
Anh cho em hỏi chỗ hàm:  
<https://machinelearningcaban.com/2017/01/12/gradientdescent/>

```
def grad(w):
N = Xbar.shape[0]
return 1/N * Xbar.T.dot(Xbar.dot(w) - y)
```

Theo em làm bằng tay thì  $Xbar \cdot w$  là ma trận  $1000 \times 1$  và  $y$  cũng là ma trận  $1000 \times 1$ . Vậy  $Xbar \cdot dot(w) - y$  phải là ma trận  $1000 \times 1$ , nhưng em chạy trên python ra  $Xbar \cdot dot(w) - y$  là ma trận  $1000 \times 1000$ . Anh có thể giúp em gỡ rối chỗ này được không ạ, Em cảm ơn anh về bài viết.

[^](#) [v](#) [• Reply](#) [Share](#)



Tiep Vu Huu Mod → Hoang Ha • 4 months ago

Để trừ ít bị lỗi thì nên đưa 2 vector về cùng chiều. Trong trường hợp của em, có thể em đã lấy 1 vector hàng trừ đi vector cột. Kết quả có thể là 1 ma trận mà phần tử ở hàng thứ i, cột thứ j bằng với phần tử thứ i của vector thứ nhất trừ đi phần tử ở hàng thứ j của vector thứ 2.

Nếu chúng cùng là vector hàng hoặc cột thì sẽ không có lỗi.

em có thể dùng `a.reshape(1, -1)` để biến vector a thành vector hàng, `a.reshape(-1, 1)` để đưa nó về vector cột.

Cụ thể về hàm reshape, em có thể google: numpy reshape.

[^](#) [v](#) [• Reply](#) [Share](#)



Stephen Tran • 6 months ago

Anh ơi cái hàm này  $y=4+3x$  và khi vào code anh lại ghi là như thế này là sao hả anh

```
y = 4 + 3 * X + .2*np.random.randn(1000, 1)
```

[^](#) [v](#) [• Reply](#) [Share](#)



Tiep Vu Huu Mod → Stephen Tran • 6 months ago

Cái đó để tạo nhiễu đó em. Nếu không có cái đó thì toàn bộ các điểm đều nằm trên 1 đường thẳng. Khi đó, bài toán không thú vị nữa vì chỉ cần 2 điểm phân biệt là xác định được 1 đường thẳng rồi.

Dữ liệu trong thực tế đâu có nằm hoàn toàn thẳng trên 1 đường mà luôn luôn có nhiễu.

[^](#) [v](#) [• Reply](#) [Share](#)

#### ALSO ON TIEPVU

#### ebook Machine Learning cơ bản

26 comments • 3 months ago

Xuân Xuân — Anh ơi, cái email trường em cấp ý, em test nó không cho các mail ngoài gửi vào:"Hi. This is the qmail-send program at vnu.edu.vn.l'm ...

#### Machine Learning cơ bản

11 comments • 8 months ago

CTVR — Em cũng thấy yêu cầu 64 bit trên win. Chắc em chạy với linux vậy.  
P/s: Giờ em mới đọc xong loạt bài về convex trên file pdf. Hồi mới vào ...

[✉ Subscribe](#) [🔗 Add Disqus to your site](#) [Add Disqus](#) [Privacy](#)

Total visits:

## Bài 9: Perceptron Learning Algorithm

Neural-nets (/tags#Neural-nets) Supervised-learning (/tags#Supervised-learning) Classification (/tags#Classification) Linear-models (/tags#Linear-models) GD (/tags#GD)

Jan 21, 2017

Cứ làm đi, sai đâu sửa đấy, cuối cùng sẽ thành công!

Đó chính là ý tưởng chính của một thuật toán rất quan trọng trong Machine Learning - thuật toán Perceptron Learning Algorithm hay PLA.

Trong trang này:

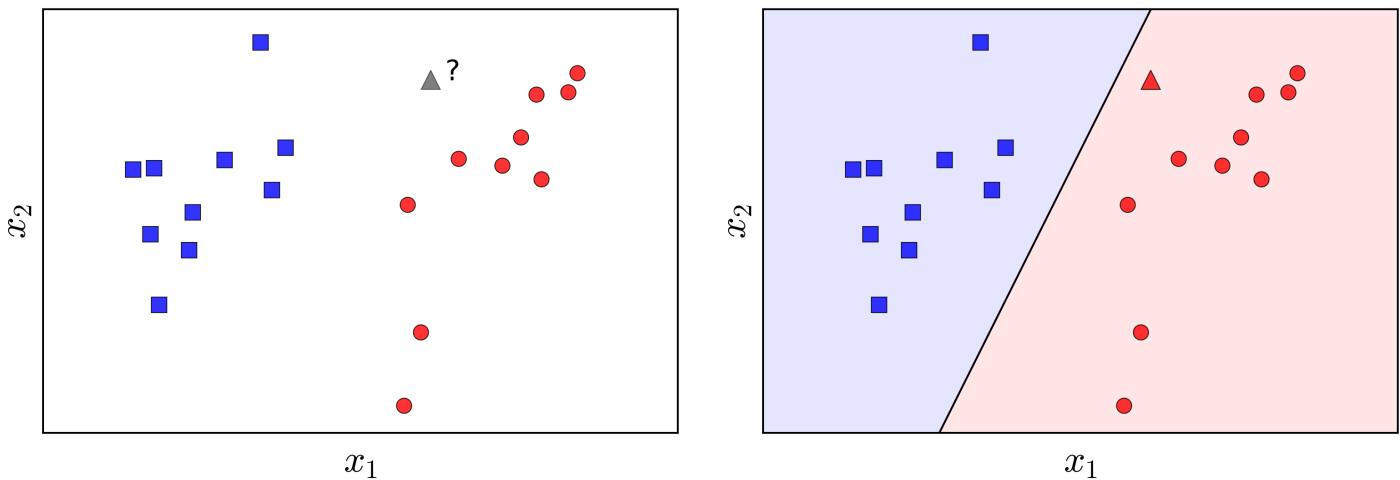
- 1. Giới thiệu
  - Bài toán Perceptron
- 2. Thuật toán Perceptron (PLA)
  - Một số ký hiệu
  - Xây dựng hàm mất mát
  - Tóm tắt PLA
- 3. Ví dụ trên Python
  - Load thư viện và tạo dữ liệu
  - Các hàm số cho PLA
- 4. Chứng minh hội tụ
- 5. Mô hình Neural Network đầu tiên
- 6. Thảo Luận
  - PLA có thể cho vô số nghiệm khác nhau
  - PLA đòi hỏi dữ liệu linearly separable
  - Pocket Algorithm
- 7. Kết luận
- 8. Tài liệu tham khảo

### 1. Giới thiệu

Trong bài này, tôi sẽ giới thiệu thuật toán đầu tiên trong Classification có tên là Perceptron Learning Algorithm (PLA) hoặc đôi khi được viết gọn là Perceptron.

Perceptron là một thuật toán Classification cho trường hợp đơn giản nhất: chỉ có hai class (lớp) (*bài toán với chỉ hai class được gọi là binary classification*) và cũng chỉ hoạt động được trong một trường hợp rất cụ thể. Tuy nhiên, nó là nền tảng cho một mảng lớn quan trọng của Machine Learning là Neural Networks và sau này là Deep Learning. (Tại sao lại gọi là Neural Networks - tức mạng dây thần kinh - các bạn sẽ được thấy ở cuối bài).

Giả sử chúng ta có hai tập hợp dữ liệu đã được gán nhãn được minh họa trong Hình 1 bên trái dưới đây. Hai class của chúng ta là tập các điểm màu xanh và tập các điểm màu đỏ. Bài toán đặt ra là: từ dữ liệu của hai tập được gán nhãn cho trước, hãy xây dựng một *classifier* (bộ phân lớp) để khi có một điểm dữ liệu hình tam giác màu xám mới, ta có thể dự đoán được màu (nhãn) của nó.



Hình 1: Bài toán Perceptron

Hiểu theo một cách khác, chúng ta cần tìm *lãnh thổ* của mỗi class sao cho, với mỗi một điểm mới, ta chỉ cần xác định xem nó nằm vào lãnh thổ của class nào rồi quyết định nó thuộc class đó. Để tìm *lãnh thổ* của mỗi class, chúng ta cần đi tìm biên giới (boundary) giữa hai *lãnh thổ* này. Vậy bài toán classification có thể coi là bài toán đi tìm boundary giữa các class. Và boundary đơn giản nhất trong không gian hai chiều là một đường thẳng, trong không gian ba chiều là một mặt phẳng, trong không gian nhiều chiều là một siêu mặt phẳng (hyperplane) (tôi gọi chung những boundary này là *đường phẳng*). Những boundary phẳng này được coi là đơn giản vì nó có thể biểu diễn dưới dạng toán học bằng một hàm số đơn giản có dạng tuyến tính, tức linear. Tuy nhiên, chúng ta đang giả sử rằng tồn tại một đường phẳng để có thể phân định *lãnh thổ* của hai class. Hình 1 bên phải minh họa một đường thẳng phân chia hai class trong mặt phẳng. Phần có nền màu xanh được coi là *lãnh thổ* của lớp xanh, phần có nền màu đỏ được coi là *lãnh thổ* của lớp đỏ. Trong trường hợp này, điểm dữ liệu mới hình tam giác được phân vào class đỏ.

## Bài toán Perceptron

Bài toán Perceptron được phát biểu như sau: *Cho hai class được gán nhãn, hãy tìm một đường phẳng sao cho toàn bộ các điểm thuộc class 1 nằm về 1 phía, toàn bộ các điểm thuộc class 2 nằm về phía còn lại của đường phẳng đó. Với giả định rằng tồn tại một đường phẳng như thế.*

Nếu tồn tại một đường phẳng phân chia hai class thì ta gọi hai class đó là *linearly separable*. Các thuật toán classification tạo ra các boundary là các đường phẳng được gọi chung là Linear Classifier.

## 2. Thuật toán Perceptron (PLA)

Cũng giống như các thuật toán lặp trong K-means Clustering (/2017/01/01/kmeans/) và Gradient Descent (/2017/01/12/gradientdescent/), ý tưởng cơ bản của PLA là xuất phát từ một nghiệm dự đoán nào đó, qua mỗi vòng lặp, nghiệm sẽ được cập nhật tới một vị trí tốt hơn. Việc cập nhật này dựa trên việc giảm giá trị của một hàm mất mát nào đó.

### Một số ký hiệu

Giả sử  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N] \in \mathbb{R}^{d \times N}$  là ma trận chứa các điểm dữ liệu mà mỗi cột  $\mathbf{x}_i \in \mathbb{R}^{d \times 1}$  là một điểm dữ liệu trong không gian  $d$  chiều. (Chú ý: khác với các bài trước tôi thường dùng các vector hàng để mô tả dữ liệu, trong bài này tôi dùng vector cột để biểu diễn. Việc biểu diễn dữ liệu ở dạng hàng hay cột tùy thuộc vào từng bài toán, miễn sao cách biểu diễn toán học của nó khiến cho người đọc thấy dễ hiểu).

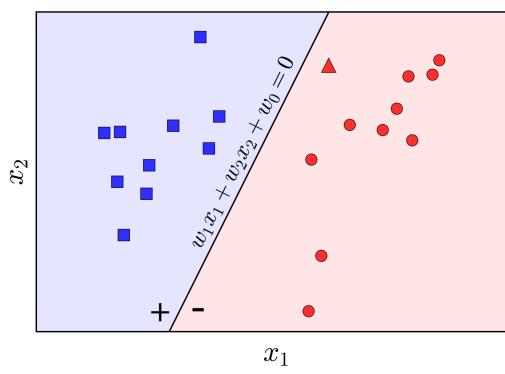
Giả sử thêm các nhãn tương ứng với từng điểm dữ liệu được lưu trong một vector hàng  $\mathbf{y} = [y_1, y_2, \dots, y_N] \in \mathbb{R}^{1 \times N}$ , với  $y_i = 1$  nếu  $\mathbf{x}_i$  thuộc class 1 (xanh) và  $y_i = -1$  nếu  $\mathbf{x}_i$  thuộc class 2 (đỏ).

Tại một thời điểm, giả sử ta tìm được boundary là đường phẳng có phương trình:

$$\begin{aligned} f_{\mathbf{w}}(\mathbf{x}) &= w_1 x_1 + \dots + w_d x_d + w_0 \\ &= \mathbf{w}^T \bar{\mathbf{x}} = 0 \end{aligned}$$

với  $\bar{\mathbf{x}}$  là điểm dữ liệu mở rộng bằng cách thêm phần tử  $x_0 = 1$  lên trước vector  $\mathbf{x}$  tương tự như trong Linear Regression (/2016/12/28/linearregression/). Và từ đây, khi nói  $\mathbf{x}$ , tôi cũng ngầm hiểu là điểm dữ liệu mở rộng.

Để cho đơn giản, chúng ta hãy cùng làm việc với trường hợp mỗi điểm dữ liệu có số chiều  $d = 2$ . Giả sử đường thẳng  $w_1 x_1 + w_2 x_2 + w_0 = 0$  chính là nghiệm cần tìm như Hình 2 dưới đây:



Hình 2: Phương trình đường thẳng boundary.

Nhận xét rằng các điểm nằm về cùng 1 phía so với đường thẳng này sẽ làm cho hàm số  $f_{\mathbf{w}}(\mathbf{x})$  mang cùng dấu. Chỉ cần đổi dấu của  $\mathbf{w}$  nếu cần thiết, ta có thể giả sử các điểm nằm trong nửa mặt phẳng nền xanh mang dấu dương (+), các điểm nằm trong nửa mặt phẳng nền đỏ mang dấu âm (-). Các dấu này cũng tương đương với nhãn  $y$  của mỗi class. Vậy nếu  $\mathbf{w}$  là một nghiệm của bài toán Perceptron, với một điểm dữ liệu mới  $\mathbf{x}$  chưa được gán nhãn, ta có thể xác định class của nó bằng phép toán đơn giản như sau:

$$\text{label}(\mathbf{x}) = 1 \text{ if } \mathbf{w}^T \mathbf{x} \geq 0, \text{ otherwise } -1$$

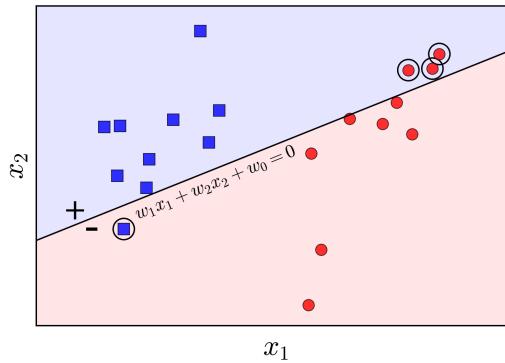
Ngắn gọn hơn:

$$\text{label}(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \mathbf{x})$$

trong đó,  $\text{sgn}$  là hàm xác định dấu, với giả sử rằng  $\text{sgn}(0) = 1$ .

## Xây dựng hàm măt măt

Tiếp theo, chúng ta cần xây dựng hàm măt măt với tham số  $\mathbf{w}$  bất kỳ. Vẫn trong không gian hai chiều, giả sử đường thẳng  $w_1x_1 + w_2x_2 + w_0 = 0$  được cho như Hình 3 dưới đây:



Hình 3: Đường thẳng bát kỳ và các điểm bị misclassified được khoanh tròn.

Trong trường hợp này, các điểm được khoanh tròn là các điểm bị misclassified (phân lớp lỗi). Điều chúng ta mong muốn là không có điểm nào bị misclassified. Hàm măt măt đơn giản nhất chúng ta nghĩ đến là hàm *đếm số* lượng các điểm bị misclassified và tìm cách tối thiểu hàm số này:

$$J_1(\mathbf{w}) = \sum_{\mathbf{x}_i \in \mathcal{M}} (-y_i \text{sgn}(\mathbf{w}^T \mathbf{x}_i))$$

trong đó  $\mathcal{M}$  là tập hợp các điểm bị misclassified (tập hợp này thay đổi theo  $\mathbf{w}$ ). Với mỗi điểm  $\mathbf{x}_i \in \mathcal{M}$ , vì điểm này bị misclassified nên  $y_i$  và  $\text{sgn}(\mathbf{w}^T \mathbf{x})$  khác nhau, và vì thế  $-y_i \text{sgn}(\mathbf{w}^T \mathbf{x}_i) = 1$ . Vậy  $J_1(\mathbf{w})$  chính là hàm *đếm số* lượng các điểm bị misclassified. Khi hàm số này đạt giá trị nhỏ nhất bằng 0 thì ta không còn điểm nào bị misclassified.

Một điểm quan trọng, hàm số này là rời rạc, không tính được đạo hàm theo  $\mathbf{w}$  nên rất khó tối ưu. Chúng ta cần tìm một hàm măt măt khác để việc tối ưu khả thi hơn.

Xét hàm măt măt sau đây:

$$J(\mathbf{w}) = \sum_{\mathbf{x}_i \in \mathcal{M}} (-y_i \mathbf{w}^T \mathbf{x}_i)$$

Hàm  $J()$  khác một chút với hàm  $J_1()$  ở việc bỏ đi hàm  $\text{sgn}$ . Nhận xét rằng khi một điểm misclassified  $\mathbf{x}_i$  nằm càng xa boundary thì giá trị  $-y_i \mathbf{w}^T \mathbf{x}_i$  sẽ càng lớn, nghĩa là sự sai lệch càng lớn. Giá trị nhỏ nhất của hàm măt măt này cũng bằng 0 nếu không có điểm nào bị misclassified. Hàm măt măt này cũng được cho là tốt hơn hàm  $J_1()$  vì nó *trừng phạt* rất nặng những điểm *lần sâu sang lãnh thổ của class kia*. Trong khi đó,  $J_1()$  *trừng phạt* các điểm misclassified như nhau (đều = 1), bắt kể chúng xa hay gần với đường biên giới.

Tại một thời điểm, nếu chúng ta chỉ quan tâm tới các điểm bị misclassified thì hàm số  $J(\mathbf{w})$  khả vi (tính được đạo hàm), vậy chúng ta có thể sử dụng Gradient Descent (/2017/01/12/gradientdescent/) hoặc Stochastic Gradient Descent (SGD) (/2017/01/16/gradientdescent2/#stochastic-gradient-descent) để tối ưu hàm măt măt này. Với ưu điểm của SGD cho các bài toán large-scale (/2017/01/12/gradientdescent/#large-scale), chúng ta sẽ làm theo thuật toán này.

Với *một* điểm dữ liệu  $\mathbf{x}_i$  bị misclassified, hàm măt măt trở thành:

$$J(\mathbf{w}; \mathbf{x}_i; y_i) = -y_i \mathbf{w}^T \mathbf{x}_i$$

Đạo hàm tương ứng:

$$\nabla_{\mathbf{w}} J(\mathbf{w}; \mathbf{x}_i; y_i) = -y_i \mathbf{x}_i$$

Vậy quy tắc cập nhật là:

$$\mathbf{w} = \mathbf{w} + \eta y_i \mathbf{x}_i$$

với  $\eta$  là learning rate.

Nhận xét rằng nếu  $\mathbf{w}$  là nghiệm thì  $\eta \mathbf{w}$  cũng là nghiệm với  $\eta$  là một số khác 0 bất kỳ. Vậy nếu  $\mathbf{w}_0$  nhỏ gần với 0 và số vòng lặp đủ lớn, ta có thể coi như learning rate  $\eta = 1$ . Ta có một quy tắc cập nhật rất gọn là:  $\mathbf{w}_{t+1} = \mathbf{w}_t + y_t \mathbf{x}_t$ . Nói cách khác, với mỗi điểm  $\mathbf{x}_i$  bị misclassified, ta chỉ cần nhân điểm đó với nhãn  $y_i$  của nó, lấy kết quả cộng vào  $\mathbf{w}$  ta sẽ được  $\mathbf{w}$  mới.

Ta có một quan sát nhỏ ở đây:

$$\begin{aligned}\mathbf{w}_{t+1}^T \mathbf{x}_i &= (\mathbf{w}_t + y_i \mathbf{x}_i)^T \mathbf{x}_i \\ &= \mathbf{w}_t^T \mathbf{x}_i + y_i \|\mathbf{x}_i\|_2^2\end{aligned}$$

Nếu  $y_i = 1$ , vì  $\mathbf{x}_i$  bị misclassified nên  $\mathbf{w}_t^T \mathbf{x}_i < 0$ . Cũng vì  $y_i = 1$  nên  $y_i \|\mathbf{x}_i\|_2^2 = \|\mathbf{x}_i\|_2^2 \geq 1$  (chú ý  $x_0 = 1$ ), nghĩa là  $\mathbf{w}_{t+1}^T \mathbf{x}_i > \mathbf{w}_t^T \mathbf{x}_i$ . Lý giải bằng lời,  $\mathbf{w}_{t+1}$  tiến về phía làm cho  $\mathbf{x}_i$  được phân lớp đúng. Điều tương tự xảy ra nếu  $y_i = -1$ .

Đến đây, cảm nhận của chúng ta với thuật toán này là: cứ chọn đường boundary đi. Xét từng điểm một, nếu điểm đó bị misclassified thì tiến đường boundary về phía làm cho điểm đó được classified đúng. Có thể thấy rằng, khi di chuyển đường boundary này, các điểm trước đó được classified đúng có thể lại bị misclassified. Mặc dù vậy, PLA vẫn được đảm bảo sẽ hội tụ sau một số hữu hạn bước (tôi sẽ chứng minh việc này ở phía sau của bài viết). Tức là cuối cùng, ta sẽ tìm được đường phẳng phân chia hai lớp, miễn là hai lớp đó là linearly separable. Đây cũng chính là lý do câu đầu tiên trong bài này tôi nói với các bạn là: "Cứ làm đi, sai đâu sửa đấy, cuối cùng sẽ thành công!".

Tóm lại, thuật toán Perceptron có thể được viết như sau:

## Tóm tắt PLA

1. Chọn ngẫu nhiên một vector hệ số  $\mathbf{w}$  với các phần tử gần 0.
2. Duyệt ngẫu nhiên qua từng điểm dữ liệu  $\mathbf{x}_i$ :
  - Nếu  $\mathbf{x}_i$  được phân lớp đúng, tức  $\text{sgn}(\mathbf{w}^T \mathbf{x}_i) = y_i$ , chúng ta không cần làm gì.
  - Nếu  $\mathbf{x}_i$  bị misclassified, cập nhật  $\mathbf{w}$  theo công thức:

$$\mathbf{w} = \mathbf{w} + y_i \mathbf{x}_i$$

3. Kiểm tra xem có bao nhiêu điểm bị misclassified. Nếu không còn điểm nào, dừng thuật toán. Nếu còn, quay lại bước 2.

## 3. Ví dụ trên Python

Như thường lệ, chúng ta sẽ thử một ví dụ nhỏ với Python.

### Load thư viện và tạo dữ liệu

Chúng ta sẽ tạo hai nhóm dữ liệu, mỗi nhóm có 10 điểm, mỗi điểm dữ liệu có hai chiều để thuận tiện cho việc minh họa. Sau đó, tạo dữ liệu mở rộng bằng cách thêm 1 vào đầu mỗi điểm dữ liệu.

```
# generate data
# List of points
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(2)

means = [[2, 2], [4, 2]]
cov = [[.3, .2], [.2, .3]]
N = 10
X0 = np.random.multivariate_normal(means[0], cov, N).T
X1 = np.random.multivariate_normal(means[1], cov, N).T

X = np.concatenate((X0, X1), axis = 1)
y = np.concatenate((np.ones((1, N)), -1*np.ones((1, N))), axis = 1)
# Xbar
X = np.concatenate((np.ones((1, 2*N)), X), axis = 0)
```

Sau khi thực hiện đoạn code này, biến  $X$  sẽ chứa dữ liệu input (mở rộng), biến  $y$  sẽ chứa nhãn của mỗi điểm dữ liệu trong  $X$ .

### Các hàm số cho PLA

Tiếp theo chúng ta cần viết 3 hàm số cho PLA:

1.  $h(w, x)$ : tính đầu ra khi biết đầu vào  $x$  và weights  $w$ .
2.  $\text{has_converged}(X, y, w)$ : kiểm tra xem thuật toán đã hội tụ chưa. Ta chỉ cần so sánh  $h(w, x)$  với *ground truth*  $y$ . Nếu giống nhau thì dừng thuật toán.
3.  $\text{perceptron}(X, y, w_{\text{init}})$ : hàm chính thực hiện PLA.

```

def h(w, x):
    return np.sign(np.dot(w.T, x))

def has_converged(X, y, w):
    return np.array_equal(h(w, X), y)

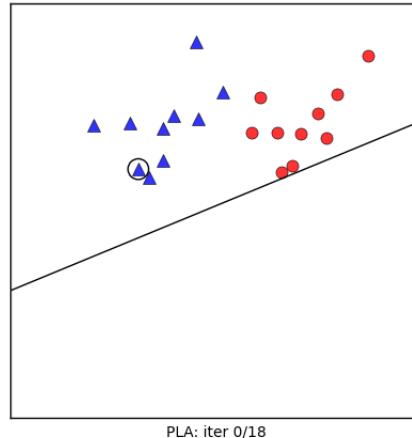
def perceptron(X, y, w_init):
    w = [w_init]
    N = X.shape[1]
    d = X.shape[0]
    mis_points = []
    while True:
        # mix data
        mix_id = np.random.permutation(N)
        for i in range(N):
            xi = X[:, mix_id[i]].reshape(d, 1)
            yi = y[0, mix_id[i]]
            if h(w[-1], xi)[0] != yi: # misclassified point
                mis_points.append(mix_id[i])
            w_new = w[-1] + yi*xi
            w.append(w_new)

    if has_converged(X, y, w[-1]):
        break
    return (w, mis_points)

d = X.shape[0]
w_init = np.random.randn(d, 1)
(w, m) = perceptron(X, y, w_init)

```

Dưới đây là hình minh họa thuật toán PLA cho bài toán nhỏ này:



Hình 4: Minh họa thuật toán PLA

Sau khi cập nhật 18 lần, PLA đã hội tụ. Điểm được khoanh tròn màu đen là điểm misclassified tương ứng được chọn để cập nhật đường boundary.

Source code cho phần này (bao gồm hình động) có thể được tìm thấy ở đây (<https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/pla/perceptron.py>).

#### 4. Chứng minh hội tụ

Giả sử rằng  $\mathbf{w}^*$  là một nghiệm của bài toán (ta có thể giả sử việc này được vì chúng ta đã có giả thiết hai class là linearly separable - tức tồn tại nghiệm). Có thể thấy rằng, với mọi  $\alpha > 0$ , nếu  $\mathbf{w}^*$  là nghiệm,  $\alpha\mathbf{w}^*$  cũng là nghiệm của bài toán. Xét dãy số không âm  $u_\alpha(t) = \|\mathbf{w}_t - \alpha\mathbf{w}^*\|_2^2$ . Với  $\mathbf{x}_i$  là một điểm bị misclassified nếu dùng nghiệm  $\mathbf{w}_t$  ta có:

$$\begin{aligned}
 u_\alpha(t+1) &= \|\mathbf{w}_{t+1} - \alpha\mathbf{w}^*\|_2^2 \\
 &= \|\mathbf{w}_t + y_i\mathbf{x}_i - \alpha\mathbf{w}^*\|_2^2 \\
 &= \|\mathbf{w}_t - \alpha\mathbf{w}^*\|_2^2 + y_i^2\|\mathbf{x}_i\|_2^2 + 2y_i\mathbf{x}_i^T(\mathbf{w} - \alpha\mathbf{w}^*) \\
 &< u_\alpha(t) + \|\mathbf{x}_i\|_2^2 - 2\alpha y_i \mathbf{x}_i^T \mathbf{w}^*
 \end{aligned}$$

Dấu nhỏ hơn ở dòng cuối là vì  $y_i^2 = 1$  và  $2y_i\mathbf{x}_i^T \mathbf{w}_t < 0$ . Nếu ta đặt:

$$\beta^2 = \max_{i=1,2,\dots,N} \|\mathbf{x}_i\|_2^2$$

$$\gamma = \min_{i=1,2,\dots,N} y_i \mathbf{x}_i^T \mathbf{w}^*$$

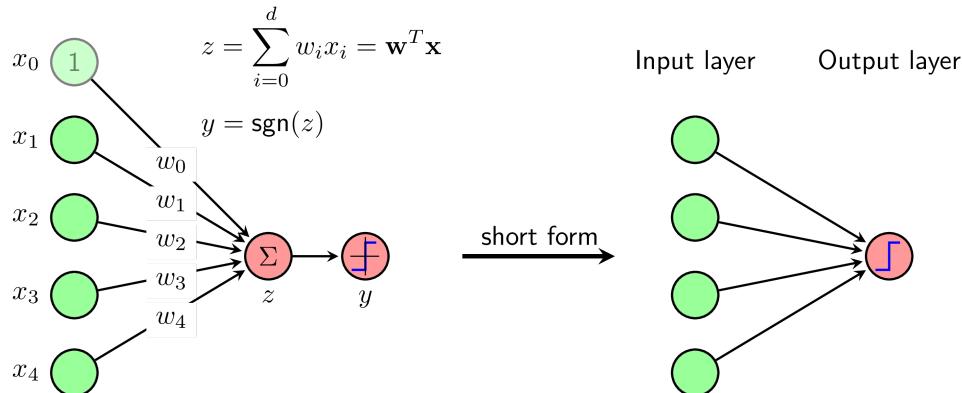
và chọn  $\alpha = \frac{\beta^2}{\gamma}$ , ta có:

$$0 \leq u_\alpha(t+1) < u_\alpha(t) + \beta^2 - 2\alpha\gamma = u_\alpha(t) - \beta^2$$

Điều này nghĩa là: nếu luôn luôn có các điểm bị misclassified thì dãy  $u_\alpha(t)$  là dãy giảm, bị chặn dưới bởi 0, và phần tử sau kém phần tử trước ít nhất một lượng là  $\beta^2 > 0$ . Điều vô lý này chứng tỏ đến một lúc nào đó sẽ không còn điểm nào bị misclassified. Nói cách khác, thuật toán PLA hội tụ sau một số hữu hạn bước.

## 5. Mô hình Neural Network đầu tiên

Hàm số xác định class của Perceptron  $\text{label}(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \mathbf{x})$  có thể được mô tả như hình vẽ (được gọi là network) dưới đây:



Hình 5: Biểu diễn của Perceptron dưới dạng Neural Network.

Đầu vào của network  $\mathbf{x}$  được minh họa bằng các node màu xanh lục với node  $x_0$  luôn luôn bằng 1. Tập hợp các node màu xanh lục được gọi là *Input layer*. Trong ví dụ này, tôi giả sử số chiều của dữ liệu  $d = 4$ . Số node trong input layer luôn luôn là  $d + 1$  với một node là 1 được thêm vào. Node  $x_0 = 1$  này đôi khi được ẩn đi.

Các trọng số (*weights*)  $w_0, w_1, \dots, w_d$  được gán vào các mũi tên đi tới node  $z = \sum_{i=0}^d w_i x_i = \mathbf{w}^T \mathbf{x}$ . Node  $y = \text{sgn}(z)$  là *output* của network. Ký hiệu

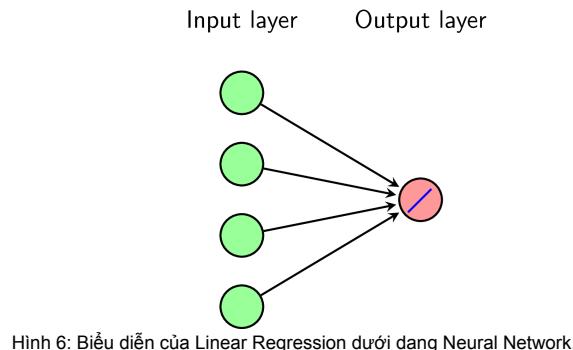
hình chữ Z ngược màu xanh trong node  $y$  thể hiện đồ thị của hàm số  $\text{sgn}$ .

Trong thuật toán PLA, ta phải tìm các weights trên các mũi tên sao cho với mỗi  $\mathbf{x}_i$  ở tập các điểm dữ liệu đã biết được đặt ở Input layer, output của network này trùng với nhãn  $y_i$  tương ứng.

Hàm số  $y = \text{sgn}(z)$  còn được gọi là *activation function*. Đây chính là dạng đơn giản nhất của Neural Network.

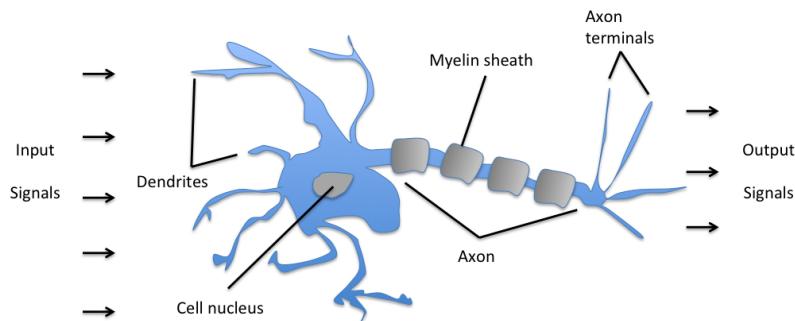
Các Neural Networks sau này có thể có nhiều node ở output tạo thành một *output layer*, hoặc có thể có thêm các layer trung gian giữa *input layer* và *output layer*. Các layer trung gian đó được gọi là *hidden layer*. Khi biểu diễn các Networks lớn, người ta thường giàn lược hình bên trái thành hình bên phải. Trong đó node  $x_0 = 1$  thường được ẩn đi. Node  $z$  cũng được ẩn đi và viết gộp vào trong node  $y$ . Perceptron thường được vẽ dưới dạng đơn giản như Hình 5 bên phải.

Để ý rằng nếu ta thay *activation function* bởi  $y = z$ , ta sẽ có Neural Network mô tả thuật toán Linear Regression như hình dưới. Với đường thẳng chéo màu xanh thể hiện đồ thị hàm số  $y = z$ . Các trực tọa độ đã được lược bỏ.



Hình 6: Biểu diễn của Linear Regression dưới dạng Neural Network.

Mô hình perceptron ở trên khá giống với một node nhỏ của dây thần kinh sinh học như hình sau đây:



Schematic of a biological neuron.

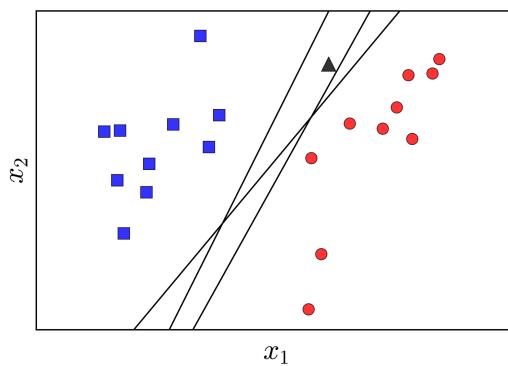
Hình 7: Mô tả một neuron thần kinh sinh học. (Nguồn: Single-Layer Neural Networks and Gradient Descent ([http://sebastianraschka.com/Articles/2015\\_singlenet\\_neurons.html](http://sebastianraschka.com/Articles/2015_singlenet_neurons.html)))

Dữ liệu từ nhiều dây thần kinh đi về một *cell nucleus*. Thông tin được tổng hợp và được đưa ra ở output. Nhiều bộ phận như thế này kết hợp với nhau tạo nên hệ thần kinh sinh học. Chính vì vậy mà có tên Neural Networks trong Machine Learning. Đôi khi mạng này còn được gọi là Artificial Neural Networks (ANN) tức *hệ neuron nhân tạo*.

## 6. Thảo Luận

### PLA có thể cho vô số nghiệm khác nhau

Rõ ràng rằng, nếu hai class là linearly separable thì có vô số đường thẳng phân cách 2 class đó. Dưới đây là một ví dụ:

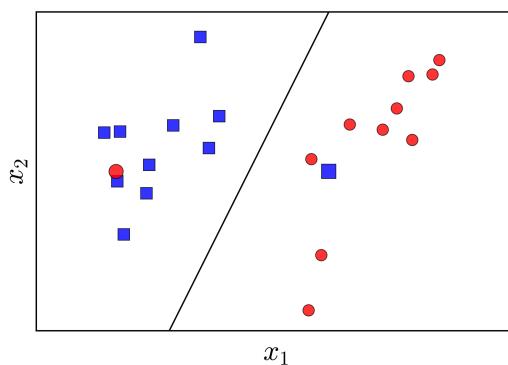


Hình 8: PLA có thể cho vô số nghiệm khác nhau.

Tất cả các đường thẳng màu đen đều thỏa mãn. Tuy nhiên, các đường khác nhau sẽ quyết định điểm hình tam giác thuộc các lớp khác nhau. Trong các đường đó, đường nào là tốt nhất? Và định nghĩa “tốt nhất” được hiểu theo nghĩa nào? Có một thuật toán khác định nghĩa và tìm đường tốt nhất như thế, tôi sẽ giới thiệu trong 1 vài bài tới. Mời các bạn đón đọc.

### PLA đòi hỏi dữ liệu linearly separable

Hai class trong ví dụ dưới đây *tương đối* linearly separable. Mỗi class có 1 điểm coi như *nhiều* nằm trong khu vực các điểm của class kia. PLA sẽ không làm việc trong trường hợp này vì luôn luôn có ít nhất 2 điểm bị misclassified.



Hình 9: PLA không làm việc nếu chỉ có một nhiễu nhỏ.

Trong một chủng mực nào đó, đường thẳng màu đen vẫn có thể coi là một nghiệm tốt vì nó đã giúp phân loại chính xác hầu hết các điểm. Việc không hội tụ với dữ liệu *gắn* linearly separable chính là một nhược điểm lớn của PLA.

Để khắc phục nhược điểm này, có một cải tiến nhỏ như thuật toán Pocket Algorithm dưới đây:

## Pocket Algorithm

Một cách tự nhiên, nếu có một vài *nhiều*, ta sẽ đi tìm một đường thẳng phân chia hai class sao cho có ít điểm bị misclassified nhất. Việc này có thể được thực hiện thông qua PLA với một chút thay đổi nhỏ như sau:

1. Giới hạn số lượng vòng lặp của PLA.
2. Mỗi lần cập nhật nghiệm  $w$  mới, ta đếm xem có bao nhiêu điểm bị misclassified. Nếu là lần đầu tiên, giữ lại nghiệm này trong *pocket* (túi quần). Nếu không, so sánh số điểm misclassified này với số điểm misclassified của nghiệm trong *pocket*, nếu nhỏ hơn thì *lấy* nghiệm cũ ra, đặt nghiệm mới này vào.

Thuật toán này giống với thuật toán tìm phần tử nhỏ nhất trong 1 mảng.

## 7. Kết luận

Hy vọng rằng bài viết này sẽ giúp các bạn phần nào hiểu được một số khái niệm trong Neural Networks. Trong một số bài tiếp theo, tôi sẽ tiếp tục nói về các thuật toán cơ bản khác trong Neural Networks trước khi chuyển sang phần khác.

Trong tương lai, nếu có thể, tôi sẽ viết tiếp về Deep Learning và chúng ta sẽ lại quay lại với Neural Networks.

## 8. Tài liệu tham khảo

- [1] F. Rosenblatt. The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory, 1957.
- [2] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115–133, 1943.
- [3] B. Widrow et al. Adaptive "Adaline" neuron using chemical "memistors". Number Technical Report 1553-2. Stanford Electron. Labs., Stanford, CA, October 1960.
- [3] Abu-Mostafa, Yaser S., Malik Magdon-Ismail, and Hsuan-Tien Lin. Learning from data. Vol. 4. New York, NY, USA:: AMLBook, 2012. (link to course (<http://work.caltech.edu/telecourse.html>))
- [4] Bishop, Christopher M. "Pattern recognition and Machine Learning.", Springer (2006). (book (<http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf>))
- [5] Duda, Richard O., Peter E. Hart, and David G. Stork. Pattern classification. John Wiley & Sons, 2012.

Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (<https://www.facebook.com/groups/257768141347267/>) để nhận được câu trả lời sớm hơn.

Bạn đọc có thể ủng hộ blog qua 'Buy me a coffee' (/buymeacoffee/) ở góc trên bên trái của blog.

Tôi đang trong quá trình viết cuốn sách 'Machine Learning cơ bản I', các bạn có thể đặt trước tại đây (/ebook/). Cảm ơn bạn.

« Bài 8: Gradient Descent (phần 2/2) (/2017/01/16/gradientdescent2/)

Bài 10: Logistic Regression » (/2017/01/27/logisticregression/)

27 Comments tiepvu

 Login ▾

 Recommend 8  Share

Sort by Best ▾

 Join the discussion...

LOG IN WITH
OR SIGN UP WITH DISQUS (?)

 Chu\_Chính • 4 months ago  
cám ơn anh.  
2 ^ | v • Reply • Share >

Avatar

This comment was deleted.



Tiep Vu Huu Mod → Guest • 2 months ago

Chào bạn. Câu hỏi của bạn rất hay. Có mấy điểm mình muốn nhấn mạnh thế này:

1. Như ở phần 6 đã nói, PLA có thể cho vô số nghiệm khác nhau vì có vô số đường thẳng phân chia hai lớp dữ liệu. Hai bức ảnh của bạn cũng cho nghiệm thỏa mãn, nên không có vấn đề gì cần khắc phục cả.
2. Hàm mất mát của PLA không chỉ có một điểm cực trị toàn cục mà có thể có vô số, vậy nên nó có nhiều điểm cực tiểu. Thuật toán sử dụng trong PLA về bản chất là Gradient Descent (Bài 7 và 8). Nếu bạn đọc ở đoạn đầu bài 8 thì sẽ thấy các hàm có nhiều điểm cực tiểu khi áp dụng Gradient Descent sẽ có thể hội tụ tại những điểm khác nhau phụ thuộc vào điểm khởi tạo. Các hàm mất mát của Neural Nets nói chung là các hàm không lồi, nên kết quả cuối phụ thuộc vào điểm khởi tạo đầu. Và có thể có nhiều kết quả đều cho nghiệm tốt.
3. Thường thì người ta khởi tạo w có các thành phần ngẫu nhiên và nhỏ gần 0.

^ | v · Reply · Share ›



Ngoc NK → Tiep Vu Huu • 2 months ago

trong trường hợp này dẫn đến:

- nếu không may trong quá trình chạy chương trình có 1 số các điểm (misclassified) đặc biệt suất hiện trong dữ liệu đầu vào đồng thời nằm ở số liệu này cũng được nằm ở một vị trí đặc biệt trong tập số liệu (ví dụ là cuối ở đây)
- điểm đặc biệt này nằm giữa 2 đường thẳng như trong 2 trường hợp trên, trong khi đó ta lại chọn 1 giá trị eta cũng đặc biệt để khi trong quá trình lặp nó cứ lặp đi lặp lại vị trí boundary là 2 đường thẳng như này thì lúc đó vòng lặp sẽ vô hạn
- trên thực tế xác suất gặp nhau thế vô cùng hiếm, nhưng nếu gặp phải thì sao?
- với điểm misclassified như thế thì chạy theo cách khác nhau cho kq khác nhau, nếu clients biết điều đó và họ không công nhận một thuật toán như vậy thì sao (không phải client nào cũng biết)

như vậy để kq có tính ổn định cao, mong tác giả có thể chỉ ra cách nào đó để dù chọn w theo cách nào cũng vẫn cho kq như nhau.

Trân thành cảm ơn.

1 ^ | v · Reply · Share ›



Tiep Vu Huu Mod → Ngoc NK • 2 months ago

P/S: Trên thực tế không có thuật toán nào chạy phân loại đúng 100% các điểm cả, chỉ là độ chính xác cao hơn thôi. Tuỳ vào từng bài toán mà người làm cần điều chỉnh tham số mô hình để đạt được hiệu quả như mong muốn.

Minh nghĩ là bạn mới học Machine Learning, và đang học rất chăm chỉ, nên có nhiều thắc mắc giống với thắc mắc của hầu hết các bạn khác, bao gồm cả mình, khi bắt đầu học lĩnh vực này. Bạn có thể tham gia và đặt câu hỏi trên Forum để nhiều người cùng tham gia thảo luận hơn. Link tới Forum ở cột bên phải của blog.

Cảm ơn bạn một lần nữa.

^ | v · Reply · Share ›



Tiep Vu Huu Mod → Ngoc NK • 2 months ago

PLA là phương pháp rất cổ, là nền tảng kiến thức cho các phương pháp phức tạp hơn sau này. Các trường hợp đặc biệt mà bạn chỉ ra có thể được khắc phục bằng Logistic Regression (bài 10), Soft margin Support Vector Machine (Bài 20), Kernel Support Vector Machine (Bài 21). Mời bạn đọc thêm các bài đó và các bài liên quan để có cái nhìn toàn cảnh hơn.

Cảm ơn bạn.

^ | v · Reply · Share ›



Ngoc NK → Tiep Vu Huu • 2 months ago

vấn đề là có cách nào khắc phục cho phương pháp này không, khi bắt buộc phải sử dụng nó thì sao?

bản thân Logistic Regression trong trường hợp này cũng sẽ không thực sự hiệu quả vì phụ thuộc qui luật phân bố xác suất, ở đây ta có linearly separable mà.

nếu ta tìm được w ổn định (mang tính global) chứ không mang tính địa phương như 2 kq đó thì quá tuyệt vời.  
vậy thôi.

^ | v · Reply · Share ›



Tiep Vu Huu Mod → Ngoc NK • 2 months ago

Xin lỗi bạn, quả thực mình không hiểu hết ý của bạn. Xin trả lời theo những gì mình hiểu.

Với hai lớp dữ liệu, nếu dữ liệu trong hai lớp là 'linearly separable' thì PLA chỉ đảm bảo tìm được đường phân chia giữa hai lớp. Bản thân nó không biết đường nào là tốt nhất, và càng không thể biết các trường hợp đặc biệt trong tập kiểm thử có thỏa mãn hơn không. Xin nhắc lại, PLA đảm bảo việc dữ liệu trong tập huấn luyện được phân chia chính xác bởi đường phân chia.

Nếu bạn muốn một đường phân chia tốt hơn, bạn có thể đọc thêm Bài 19: Support Vector Machine để biết thêm chi tiết.

Nhân tiện, bạn không 'bắt buộc' phải sử dụng phương pháp nào cả. Nếu PLA làm được thì SVM cũng làm được, vì cả hai đều cho bài toán phân lớp. Thực tế bạn cần thử nhiều mô hình khác nhau cho mỗi bài toán để chọn ra mô hình tốt nhất. Không có thuật toán vạn năng nào cho mọi bài toán cả.

Thiết kế PLA là một thiết kế của nhà khoa học của nó không tồn tại mãi mãi trong Mục 6. Tuy nhiên, nó vẫn được dùng để

Mô hình PLA là một mô hình đơn giản, chất lượng của nó không quá cao và hiệu ứng mịn. Tuy nhiên, nó vẫn được ưa chuộng trong Neural Network như là bước đệm để làm quen với các mô hình này.

Hy vọng câu trả lời này giúp ích cho bạn.

[^](#) [v](#) [• Reply](#) [Share](#) >



**Ngoc NK** → Tiep Vu Huu • 2 months ago

cám ơn bạn đã trả lời những thắc mắc đó và đã giúp mình hiểu thêm được nhiều điều từ đây.

[^](#) [v](#) [• Reply](#) [Share](#) >



**Lê Hải Sơn** → Ngoc NK • 2 months ago

Bạn muốn nâng cao và khắc phục thì thử sử dụng boosting đi, cái đoạn PLA bạn chỉ để xác suất phân loại tầm 0.6 , 0.7 => bạn có w1 , sau đó kiểm tra lại w1 và đánh dấu những điểm mà w1 phân loại sai , ở lần PLA thứ 2 ưu tiên phân loại đúng các điểm đã đánh dấu là phân loại sai ở w1 => được w2 ( w2 cũng có xác suất phân loại sai tầm 0.6, 0.7) , lần PLA thứ 3 thì cũng làm tương tự như 2 lần trước rồi kết hợp 3 đường phân loại vào ( w1,w2,w3 ) vào là được một bộ phân loại mạnh hơn, nó là đường gấp khúc có thể phân loại tốt hơn

[^](#) [v](#) [• Reply](#) [Share](#) >



**Ngoc NK** → Lê Hải Sơn • 2 months ago

mình cám ơn bạn đã nhiệt tình chia sẻ,

để mình thử xem sao,

[^](#) [v](#) [• Reply](#) [Share](#) >

Avatar

This comment was deleted.



**Tiep Vu Huu** Mod → Guest • 2 months ago

Cảm ơn bạn đã sửa lại câu hỏi để dễ hiểu hơn.

Thêm CHỈ một phần tử x0 lên trước vector x là chính xác.

Phần X = np.concatenate((np.ones((1, 2\*N)), X), axis = 0) có nghĩa là ghép một vector hàng toàn 1 vào phía trên (chiều axis = 0) của ma trận dữ liệu X ban đầu. Trong ma trận dữ liệu X ban đầu, mỗi CỘT là một điểm dữ liệu. Vậy nên thêm số 1 lên trên nó nghĩa là chỉ thêm MỘT phần tử mà thôi.

Hy vọng câu trả lời hữu ích.

[^](#) [v](#) [• Reply](#) [Share](#) >



**Ngoc NK** → Tiep Vu Huu • 2 months ago

vâng, bạn đúng rồi, mình quên mất đó là X\_bar. sorry bạn.

[^](#) [v](#) [• Reply](#) [Share](#) >



**Tiep Vu Huu** Mod → Guest • 2 months ago

Cảm ơn bạn đã góp ý. Nhưng thực sự mình không hiểu câu "mình thấy là đã thêm vector x0=1 cùng số chiều với vector x lên trước vector x" của bạn.

x0 là một số chứ không phải một vector.

[^](#) [v](#) [• Reply](#) [Share](#) >



**Quoc Tc** • 3 months ago

Có vấn đề em chưa hiểu, hi vọng có bạn nào hiểu hoặc ai Tiệp có thể giúp:

Với mạng neuron network đã tìm ra ở trên (của phần ví dụ)  $w = [[3.6864918], [-4.08776229], [4.1502239]]$ . Thi làm sao biết được "điểm dữ liệu mới" thuộc phần dữ liệu nào? nhìn trên hình vẽ thi mình biết nó nằm bên phải hay bên trái đường phân chia, nhưng dựa vào neuron network thi làm sao mình biết được?

1 [^](#) [v](#) [• Reply](#) [Share](#) >



**Tiep Vu Huu** Mod → Quoc Tc • 3 months ago

Như trong bài anh đã nói. Nếu tìm được w rồi thi class của điểm dữ liệu mới x được xác định bằng  $\text{sgn}(w^T x)$  trong đó sgn là hàm xác định dấu.

[^](#) [v](#) [• Reply](#) [Share](#) >



**Vũ Tiến Dũng** • 3 months ago

Cám ơn bạn về bài viết. Xin góp ý mấy điểm bị sai chính tả để bạn sửa.

Ở section 5, paragraph 6, "Các Neral Networks sau này có thể có nhiều node ở output ...", sửa lại là "Các Neural.."

Ở Hình 6, comment "Hình 6: ... đang Neural Network.", sửa lại là "đang"

1 điểm nữa mình để ý là bạn viết hay bị lẫn cách gọi của số, khi thi dùng số "1, 2" khi thi dùng "một, hai" trong cùng một câu :D

1 [^](#) [v](#) [• Reply](#) [Share](#) >



**Tiep Vu Huu** Mod → Vũ Tiến Dũng • 3 months ago

Cám ơn bạn. Lỗi này mình cũng biết từ lâu mà chưa bao giờ để ý sửa. Mình còn một lỗi nữa là đánh nhầm dấu cho hai từ gần nhau. :D

[^](#) [v](#) [• Reply](#) [Share](#) >

[^](#) [|](#) [v](#) • Reply • Share

Lan Anh NGUYEN • 3 months ago

Trong PLA, mảng các giá trị m được return, có giá trị bằng 0 (tại vị trí 2 và 10) tức là lần lặp đó có số misclassified = 0. Như vậy, thuật toán sẽ phải dừng. Tiệp cho mình hỏi, tại sao thuật toán k dừng luôn thời điểm đó mà tiếp tục cho đến 18? Mình hiểu sai chỗ nào chăng?

Cám ơn Tiệp vì sự đóng góp của bạn cho cộng đồng ML.

[1 ^](#) [|](#) [v](#) • Reply • Share

Tiеп Vu Huu Mod → Lan Anh NGUYEN • 3 months ago

Bạn có thể show đoạn kết quả đó ra được không. Vì đúng như bạn nói, thuật toán sẽ dừng khi số lượng misclassified bằng 0.

[^](#) [|](#) [v](#) • Reply • Share

Chu\_Chính • 4 months ago

Thuật toán này có thể dừng phân loại động vật hay thực vật được không nhỉ ?? anh tiệp

[1 ^](#) [|](#) [v](#) • Reply • Share

Tiếp Vu Huu Mod → Chu\_Chính • 3 months ago

Có thể. Nhưng nó phụ thuộc rất nhiều vào việc dữ liệu em có là gì. Thuật toán tốt mà không có dữ liệu tốt thì cũng được.

[^](#) [|](#) [v](#) • Reply • Share

Ngan Nguyen • 4 months ago

Cảm ơn anh rất nhiều.

[1 ^](#) [|](#) [v](#) • Reply • Share

Nguyễn Văn Đức • 4 days ago

Bài viết rất hay ạ :D

[^](#) [|](#) [v](#) • Reply • Share

Sang Thái • 3 months ago

Em có một chỗ thắc mắc ở phần xây dựng hàm mất mát: "Vậy nếu  $w_0$  nhỏ gần với 0 và số vòng lặp đủ lớn, ta có thể coi như learning rate  $\eta = 1$ ". Anh có thể giải thích thêm được không ạ, em cảm ơn anh nhiều :)

[^](#) [|](#) [v](#) • Reply • Share

đức nam đoàn • 4 months ago

anh cho em hỏi là tại sao ậy nếu  $w_0$  nhỏ gần với 0 và số vòng lặp đủ lớn, ta có thể coi như learning rate  $\eta = 1$ . Theo em hiểu thì với mọi  $w_0$  chứ nhỉ?

[^](#) [|](#) [v](#) • Reply • Share

Phương Nguyễn • 5 months ago

Hi anh !

Đọc các bài viết của anh em đều có thể hiểu và nắm được ý đồ của thuật toán tuy nhiên các phần giải thích và chứng minh bằng toán thì em lại hơi rối, anh có lời khuyên nào để em có thể cải thiện được phần này không ah. Cảm ơn anh.

[^](#) [|](#) [v](#) • Reply • Share

Tiếp Vu Huu Mod → Phương Nguyễn • 5 months ago

Lời khuyên duy nhất của anh là 'rối ở đâu, gỡ ở đó'. Phần nào em chưa hiểu kỹ thì tra cứu thêm. Nếu vẫn chưa hiểu thì có thể lên Forum của blog để hỏi thêm. Câu hỏi càng cụ thể càng tốt.

[^](#) [|](#) [v](#) • Reply • Share

ALSO ON TIEPVU

**ebook Machine Learning cơ bản**

26 comments • 3 months ago

Xuân Xuân — Anh ơi, cái email trường em cấp ý, em test nó không cho các mail ngoài gửi vào: "Hi. This is the qmail-send program at vnu.edu.vn.I'm ...

**Machine Learning cơ bản**

11 comments • 8 months ago

CTVR — Em cũng thấy yêu cầu 64 bit trên win. Chắc em chạy với linux vậy. P/s: Giờ em mới đọc xong loạt bài về convex trên file pdf. Hồi mới vào ...

[✉ Subscribe](#) [>Add Disqus to your site](#)[Add Disqus](#) [🔒 Privacy](#)

## Bài 10: Logistic Regression

Neural-nets (/tags#Neural-nets) Supervised-learning (/tags#Supervised-learning) Regression (/tags#Regression) Classification (/tags#Classification) GD (/tags#GD)

Jan 27, 2017

Trong trang này:

- 1. Giới thiệu
  - Nhắc lại hai mô hình tuyến tính
  - Một ví dụ nhỏ
  - Mô hình Logistic Regression
  - Sigmoid function
- 2. Hàm mất mát và phương pháp tối ưu
  - Xây dựng hàm mất mát
  - Tối ưu hàm mất mát
  - Công thức cập nhật cho logistic sigmoid regression
- 3. Ví dụ với Python
  - Ví dụ với dữ liệu 1 chiều
  - Các hàm cần thiết cho logistic sigmoid regression
  - Ví dụ với dữ liệu 2 chiều
- 4. Một vài tính chất của Logistic Regression
  - Logistic Regression thực ra được sử dụng nhiều trong các bài toán Classification.
  - Boundary tạo bởi Logistic Regression có dạng tuyến tính
- 5. Thảo luận
- 6. Tài liệu tham khảo

### 1. Giới thiệu

#### Nhắc lại hai mô hình tuyến tính

Hai mô hình tuyến tính (linear models) Linear Regression (/2016/12/28/linearregression/) và Perceptron Learning Algorithm (/2017/01/21/perceptron/) (PLA) chúng ta đã biết đều có chung một dạng:

$$y = f(\mathbf{w}^T \mathbf{x})$$

trong đó  $f()$  được gọi là *activation function*, và  $\mathbf{x}$  được hiểu là dữ liệu mở rộng với  $x_0 = 1$  được thêm vào để thuận tiện cho việc tính toán. Với linear regression thì  $f(s) = s$ , với PLA thì  $f(s) = \text{sgn}(s)$ . Trong linear regression, tích vô hướng  $\mathbf{w}^T \mathbf{x}$  được trực tiếp sử dụng để dự đoán output  $y$ , loại này phù hợp nếu chúng ta cần dự đoán một giá trị thực của đầu ra không bị chặn trên và dưới. Trong PLA, đầu ra chỉ nhận một trong hai giá trị 1 hoặc -1, phù hợp với các bài toán *binary classification*.

Trong bài này, tôi sẽ giới thiệu mô hình thứ ba với một activation khác, được sử dụng cho các bài toán *flexible* hơn. Trong dạng này, đầu ra có thể được thể hiện dưới dạng xác suất (probability). Ví dụ: xác suất thi đỗ nếu biết thời gian ôn thi, xác suất ngày mai có mưa dựa trên những thông tin đó được trong ngày hôm nay,... Mô hình mới này của chúng ta có tên là *logistic regression*. Mô hình này giống với linear regression ở khía cạnh đầu ra là số thực, và giống với PLA ở việc đầu ra bị chặn (trong đoạn [0, 1]). Mặc dù trong tên có chứa từ *regression*, logistic regression thường được sử dụng nhiều hơn cho các bài toán *classification*.

#### Một ví dụ nhỏ

Tôi xin được sử dụng một ví dụ trên Wikipedia ([https://en.wikipedia.org/wiki/Logistic\\_regression](https://en.wikipedia.org/wiki/Logistic_regression)):

Một nhóm 20 sinh viên dành thời gian trong khoảng từ 0 đến 6 giờ cho việc ôn thi. Thời gian ôn thi này ảnh hưởng đến xác suất sinh viên vượt qua kỳ thi như thế nào?

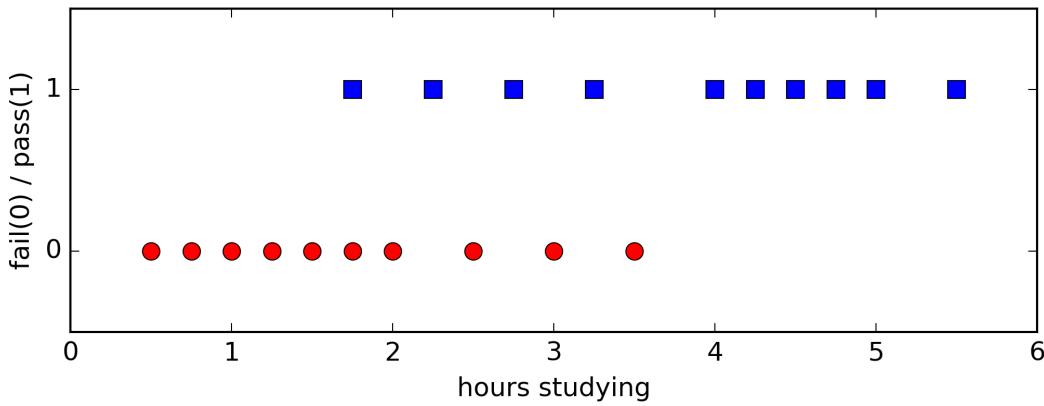
Kết quả thu được như sau:

Hours	Pass	Hours	Pass
.5	0	2.75	1
.75	0	3	0
1	0	3.25	1

Hours	Pass	Hours	Pass
1.25	0	3.5	0
1.5	0	4	1
1.75	0	4.25	1
1.75	1	4.5	1
2	0	4.75	1
2.25	1	5	1
2.5	0	5.5	1

Mặc dù có một chút *bất công* khi học 3.5 giờ thì trượt, còn học 1.75 giờ thì lại đỗ, nhìn chung, học càng nhiều thì khả năng đỗ càng cao. PLA không thể áp dụng được cho bài toán này vì không thể nói một người học bao nhiêu giờ thì 100% trượt hay đỗ, và thực tế là dữ liệu này cũng không *linearly separable* (điều kiện để PLA có thể làm việc). Chú ý rằng các điểm màu đỏ và xanh được vẽ ở hai tung độ khác nhau để tiện cho việc minh họa. Các điểm này được vẽ dùng cả dữ liệu đầu vào  $x$  và đầu ra  $y$ . Khi ta nói *linearly separable* là khi ta chỉ dùng dữ liệu đầu vào  $x$ .

Chúng ta biểu diễn các điểm này trên đồ thị để thấy rõ hơn:



Hình 1: Ví dụ về kết quả thi dựa trên số giờ ôn tập.

Nhận thấy rằng cả linear regression và PLA đều không phù hợp với bài toán này, chúng ta cần một mô hình *flexible* hơn.

## Mô hình Logistic Regression

Đầu ra dự đoán của:

- Linear Regression:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

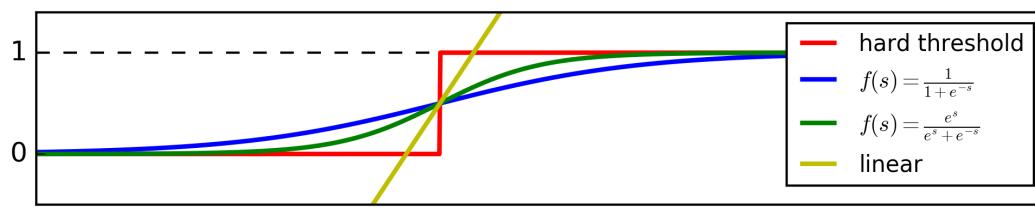
- PLA:

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \mathbf{x})$$

Đầu ra dự đoán của logistic regression thường được viết chung dưới dạng:

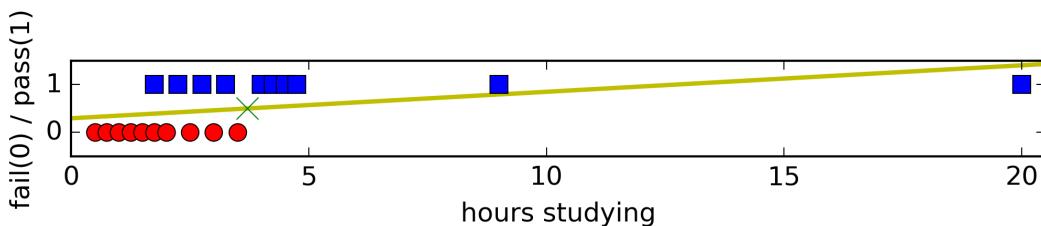
$$f(\mathbf{x}) = \theta(\mathbf{w}^T \mathbf{x})$$

Trong đó  $\theta$  được gọi là logistic function. Một số activation cho mô hình tuyến tính được cho trong hình dưới đây:



Hình 2: Các activation function khác nhau.

- Đường màu vàng biểu diễn linear regression. Đường này không bị chặn nên không phù hợp cho bài toán này. Có một *trick* nhỏ để đưa nó về dạng bị chặn: cắt phần nhỏ hơn 0 bằng cách cho chúng bằng 0, cắt các phần lớn hơn 1 bằng cách cho chúng bằng 1. Sau đó lấy điểm trên đường thẳng này có tung độ bằng 0.5 làm điểm phân chia hai class, đây cũng không phải là một lựa chọn tốt. Giả sử có thêm vài bạn sinh viên tiêu biểu ôn tập đến 20 giờ và, tất nhiên, thi đỗ. Khi áp dụng mô hình linear regression như hình dưới đây và lấy mốc 0.5 để phân lớp, toàn bộ sinh viên thi trượt vẫn được dự đoán là trượt, nhưng rất nhiều sinh viên thi đỗ cũng được dự đoán là trượt (nếu ta coi điểm x màu xanh lục là *ngưỡng cứng* để đưa ra kết luận). Rõ ràng đây là một mô hình không tốt. Anh chàng sinh viên tiêu biểu này đã kéo theo rất nhiều bạn khác bị trượt.



Hình 3: Tại sao Linear Regression không phù hợp?

- Đường màu đỏ (chỉ khác với activation function của PLA ở chỗ hai class là 0 và 1 thay vì -1 và 1) cũng thuộc dạng *ngưỡng cứng* (hard threshold). PLA không hoạt động trong bài toán này vì dữ liệu đã cho không *linearly separable*.
- Các đường màu xanh lam và xanh lục phù hợp với bài toán của chúng ta hơn. Chúng có một vài tính chất quan trọng sau:
  - Là hàm số liên tục nhận giá trị thực, bị chặn trong khoảng  $(0, 1)$ .
  - Nếu coi điểm có tung độ là  $1/2$  làm điểm phân chia thì các điểm càng xa điểm này về phía bên trái có giá trị càng gần 0. Ngược lại, các điểm càng xa điểm này về phía phải có giá trị càng gần 1. Điều này *khớp* với nhận xét rằng học càng nhiều thì xác suất đỗ càng cao và ngược lại.
  - Mượt* (smooth) nên có đạo hàm mọi nơi, có thể được lợi trong việc tối ưu.

## Sigmoid function

Trong số các hàm số có 3 tính chất nói trên thì hàm *sigmoid*:

$$f(s) = \frac{1}{1 + e^{-s}} \triangleq \sigma(s)$$

được sử dụng nhiều nhất, vì nó bị chặn trong khoảng  $(0, 1)$ . Thêm nữa:

$$\lim_{s \rightarrow -\infty} \sigma(s) = 0; \quad \lim_{s \rightarrow +\infty} \sigma(s) = 1$$

Đặc biệt hơn nữa:

$$\begin{aligned} \sigma'(s) &= \frac{e^{-s}}{(1 + e^{-s})^2} \\ &= \frac{1}{1 + e^{-s}} \frac{e^{-s}}{1 + e^{-s}} \\ &= \sigma(s)(1 - \sigma(s)) \end{aligned}$$

Công thức đạo hàm đơn giản này giúp hàm số này được sử dụng rộng rãi. Ở phần sau, tôi sẽ lý giải việc *người ta đã tìm ra hàm số đặc biệt này như thế nào*.

Ngoài ra, hàm *tanh* cũng hay được sử dụng:

$$\tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Hàm số này nhận giá trị trong khoảng  $(-1, 1)$  nhưng có thể dễ dàng đưa nó về khoảng  $(0, 1)$ . Bạn đọc có thể chứng minh được:

$$\tanh(s) = 2\sigma(2s) - 1$$

## 2. Hàm mất mát và phương pháp tối ưu

### Xây dựng hàm mất mát

Với mô hình như trên (các activation màu xanh lam và lục), ta có thể giả sử rằng xác suất để một điểm dữ liệu  $\mathbf{x}$  rơi vào class 1 là  $f(\mathbf{w}^T \mathbf{x})$  và rơi vào class 0 là  $1 - f(\mathbf{w}^T \mathbf{x})$ . Với mô hình được giả sử như vậy, với các điểm dữ liệu training (đã biết đầu ra  $y$ ), ta có thể viết như sau:

$$\begin{aligned} P(y_i = 1 | \mathbf{x}_i; \mathbf{w}) &= f(\mathbf{w}^T \mathbf{x}_i) \quad (1) \\ P(y_i = 0 | \mathbf{x}_i; \mathbf{w}) &= 1 - f(\mathbf{w}^T \mathbf{x}_i) \quad (2) \end{aligned}$$

trong đó  $P(y_i = 1 | \mathbf{x}_i; \mathbf{w})$  được hiểu là xác suất xảy ra sự kiện đầu ra  $y_i = 1$  khi biết tham số mô hình  $\mathbf{w}$  và dữ liệu đầu vào  $\mathbf{x}_i$ . Bạn đọc có thể đọc thêm Xác suất có điều kiện ([https://vi.wikipedia.org/wiki/Xác\\_suất\\_có điều\\_kiện](https://vi.wikipedia.org/wiki/Xác_suất_có điều_kiện)). Mục đích của chúng ta là tìm các hệ số  $\mathbf{w}$  sao cho  $f(\mathbf{w}^T \mathbf{x}_i)$  càng gần với 1 càng tốt với các điểm dữ liệu thuộc class 1 và càng gần với 0 càng tốt với những điểm thuộc class 0.

Ký hiệu  $z_i = f(\mathbf{w}^T \mathbf{x}_i)$  và viết gộp lại hai biểu thức bên trên ta có:

$$P(y_i | \mathbf{x}_i; \mathbf{w}) = z_i^{y_i} (1 - z_i)^{1-y_i}$$

Biểu thức này là tương đương với hai biểu thức (1) và (2) ở trên vì khi  $y_i = 1$ , phần thứ hai của vế phải sẽ triệt tiêu, khi  $y_i = 0$ , phần thứ nhất sẽ bị triệt tiêu! Chúng ta muốn mô hình gần với dữ liệu đã cho nhất, tức xác suất này đạt giá trị cao nhất.

Xét toàn bộ training set với  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N] \in \mathbb{R}^{d \times N}$  và  $\mathbf{y} = [y_1, y_2, \dots, y_N]$ , chúng ta cần tìm  $\mathbf{w}$  để biểu thức sau đây đạt giá trị lớn nhất:

$$P(\mathbf{y}|\mathbf{X}; \mathbf{w})$$

ở đây, ta cũng ký hiệu  $\mathbf{X}, \mathbf{y}$  như các biến ngẫu nhiên ([https://vi.wikipedia.org/wiki/Biến\\_ngẫu\\_nhiên](https://vi.wikipedia.org/wiki/Biến_ngẫu_nhiên)) (random variables). Nói cách khác:

$$\mathbf{w} = \arg \max_{\mathbf{w}} P(\mathbf{y}|\mathbf{X}; \mathbf{w})$$

Bài toán tìm tham số để mô hình gần với dữ liệu nhất trên đây có tên gọi chung là bài toán *maximum likelihood estimation* ([https://en.wikipedia.org/wiki/Maximum\\_likelihood\\_estimation](https://en.wikipedia.org/wiki/Maximum_likelihood_estimation)) với hàm số phía sau arg max được gọi là *likelihood function*. Khi làm việc với các bài toán Machine Learning sử dụng các mô hình xác suất thống kê, chúng ta sẽ gặp lại các bài toán thuộc dạng này, hoặc *maximum a posteriori estimation* ([https://en.wikipedia.org/wiki/Maximum\\_a\\_posteriori\\_estimation](https://en.wikipedia.org/wiki/Maximum_a_posteriori_estimation)), rất nhiều. Tôi sẽ dành 1 bài khác để nói về hai dạng bài toán này.

Giả sử thêm rằng các điểm dữ liệu được sinh ra một cách ngẫu nhiên độc lập với nhau (independent), ta có thể viết:

$$\begin{aligned} P(\mathbf{y}|\mathbf{X}; \mathbf{w}) &= \prod_{i=1}^N P(y_i|\mathbf{x}_i; \mathbf{w}) \\ &= \prod_{i=1}^N z_i^{y_i} (1 - z_i)^{1-y_i} \end{aligned}$$

với  $\prod$  là ký hiệu của tích. Bạn đọc có thể muốn đọc thêm về *Độc lập thống kê* ([https://vi.wikipedia.org/wiki/%C4%90%C3%ACc\\_l%C3%A0p\\_th%C3%B9ng\\_k%C3%A9](https://vi.wikipedia.org/wiki/%C4%90%C3%ACc_l%C3%A0p_th%C3%B9ng_k%C3%A9)).

Trực tiếp tối ưu hàm số này theo  $\mathbf{w}$  nhìn qua không đơn giản! Hơn nữa, khi  $N$  lớn, tích của  $N$  số nhỏ hơn 1 có thể dẫn tới sai số trong tính toán (numerical error) vì tích là một số quá nhỏ. Một phương pháp thường được sử dụng đó là lấy logarit tự nhiên (cơ số  $e$ ) của *likelihood function* biến phép nhân thành phép cộng và để tránh việc số quá nhỏ. Sau đó lấy ngược dấu để được một hàm và coi nó là hàm mất mát. Lúc này bài toán tìm giá trị lớn nhất (maximum likelihood) trở thành bài toán tìm giá trị nhỏ nhất của hàm mất mát (hàm này còn được gọi là negative log likelihood):

$$\begin{aligned} J(\mathbf{w}) &= -\log P(\mathbf{y}|\mathbf{X}; \mathbf{w}) \\ &= -\sum_{i=1}^N (y_i \log z_i + (1 - y_i) \log(1 - z_i)) \end{aligned}$$

với chú ý rằng  $z_i$  là một hàm số của  $\mathbf{w}$ . Bạn đọc tạm nhớ biểu thức vế phải có tên gọi là *cross entropy*, thường được sử dụng để đo *khoảng cách* giữa hai phân phối (distributions). Trong bài toán đang xét, một phân phối là dữ liệu được cho, với xác suất chỉ là 0 hoặc 1; phân phối còn lại được tính theo mô hình logistic regression. *Khoảng cách* giữa hai phân phối nhỏ đồng nghĩa với việc (có *về hiến nhiên* là) hai phân phối đó rất gần nhau. Tính chất cụ thể của hàm số này sẽ được đề cập trong một bài khác mà tầm quan trọng của khoảng cách giữa hai phân phối là lớn hơn.

**Chú ý:** Trong machine learning, logarit thập phân ít được dùng, vì vậy log thường được dùng để ký hiệu logarit tự nhiên.

## Tối ưu hàm mất mát

Chúng ta lại sử dụng phương pháp Stochastic Gradient Descent (/2017/01/16/gradientdescent2/#-stochastic-gradient-descent) (SGD) ở đây (Bạn đọc được khuyến khích đọc SGD trước khi đọc phần này). Hàm mất mát với chỉ một điểm dữ liệu  $(\mathbf{x}_i, y_i)$  là:

$$J(\mathbf{w}; \mathbf{x}_i, y_i) = -(y_i \log z_i + (1 - y_i) \log(1 - z_i))$$

Với đạo hàm:

$$\begin{aligned} \frac{\partial J(\mathbf{w}; \mathbf{x}_i, y_i)}{\partial \mathbf{w}} &= -\left(\frac{y_i}{z_i} - \frac{1 - y_i}{1 - z_i}\right) \frac{\partial z_i}{\partial \mathbf{w}} \\ &= \frac{z_i - y_i}{z_i(1 - z_i)} \frac{\partial z_i}{\partial \mathbf{w}} \quad (3) \end{aligned}$$

Để cho biểu thức này trở nên gọn và đẹp hơn, chúng ta sẽ tìm hàm  $z = f(\mathbf{w}^T \mathbf{x})$  sao cho mẫu số bị triệt tiêu. Nếu đặt  $s = \mathbf{w}^T \mathbf{x}$ , chúng ta sẽ có:

$$\frac{\partial z_i}{\partial \mathbf{w}} = \frac{\partial z_i}{\partial s} \frac{\partial s}{\partial \mathbf{w}} = \frac{\partial z_i}{\partial s} \mathbf{x}$$

Một cách trực quan nhất, ta sẽ tìm hàm số  $z = f(s)$  sao cho:

$$\frac{\partial z}{\partial s} = z(1 - z) \quad (4)$$

để triệt tiêu mẫu số trong biểu thức (3). Chúng ta cùng khởi động một chút với phương trình vi phân đơn giản này. Phương trình (4) tương đương với:

$$\begin{aligned}
 & \frac{\partial z}{z(1-z)} = \partial s \\
 \Leftrightarrow & \left(\frac{1}{z} + \frac{1}{1-z}\right)\partial z = \partial s \\
 \Leftrightarrow & \log z - \log(1-z) = s \\
 \Leftrightarrow & \log \frac{z}{1-z} = s \\
 \Leftrightarrow & \frac{z}{1-z} = e^s \\
 \Leftrightarrow & z = e^s(1-z) \\
 \Leftrightarrow & z = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}} = \sigma(s)
 \end{aligned}$$

Đến đây, tôi hy vọng các bạn đã hiểu hàm số *sigmoid* được tạo ra như thế nào.

*Chú ý: Trong việc giải phương trình vi phân ở trên, tôi đã bỏ qua hằng số khi lấy nguyên hàm hai vế. Tuy vậy, việc này không ảnh hưởng nhiều tới kết quả.*

## Công thức cập nhật cho logistic sigmoid regression

Tới đây, bạn đọc có thể kiểm tra rằng:

$$\frac{\partial J(\mathbf{w}; \mathbf{x}_i, y_i)}{\partial \mathbf{w}} = (z_i - y_i)\mathbf{x}_i$$

Quá đẹp!

Và công thức cập nhật (theo thuật toán SGD (/2017/01/16/gradientdescent2/#-stochastic-gradient-descent)) cho logistic regression là:

$$\mathbf{w} = \mathbf{w} + \eta(y_i - z_i)\mathbf{x}_i$$

Khá đơn giản! Và, như thường lệ, chúng ta sẽ có vài ví dụ với Python.

## 3. Ví dụ với Python

### Ví dụ với dữ liệu 1 chiều

Quay trở lại với ví dụ nêu ở phần Giới thiệu. Trước tiên ta cần khai báo vài thư viện và dữ liệu:

```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(2)

X = np.array([[0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 1.75, 2.00, 2.25, 2.50,
              2.75, 3.00, 3.25, 3.50, 4.00, 4.25, 4.50, 4.75, 5.00, 5.50]])
y = np.array([0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1])

# extended data
X = np.concatenate((np.ones((1, X.shape[1])), X), axis = 0)
```

### Các hàm cần thiết cho logistic sigmoid regression

```

def sigmoid(s):
    return 1/(1 + np.exp(-s))

def logistic_sigmoid_regression(X, y, w_init, eta, tol = 1e-4, max_count = 10000):
    w = [w_init]
    it = 0
    N = X.shape[1]
    d = X.shape[0]
    count = 0
    check_w_after = 20
    while count < max_count:
        # mix data
        mix_id = np.random.permutation(N)
        for i in mix_id:
            xi = X[:, i].reshape(d, 1)
            yi = y[i]
            zi = sigmoid(np.dot(w[-1].T, xi))
            w_new = w[-1] + eta*(yi - zi)*xi
            count += 1
        # stopping criteria
        if count%check_w_after == 0:
            if np.linalg.norm(w_new - w[-check_w_after]) < tol:
                return w
        w.append(w_new)
    return w
eta = .05
d = X.shape[0]
w_init = np.random.randn(d, 1)

w = logistic_sigmoid_regression(X, y, w_init, eta)
print(w[-1])

```

```

[[ -4.092695 ]
 [ 1.55277242]]

```

Với kết quả tìm được, đầu ra  $y$  có thể được dự đoán theo công thức:  $y = \text{sigmoid}(-4.1 + 1.55*x)$ . Với dữ liệu trong tập training, kết quả là:

```

print(sigmoid(np.dot(w[-1].T, X)))

```

```

[[ 0.03281144  0.04694533  0.06674738  0.09407764  0.13102736  0.17961209
  0.17961209  0.24121129  0.31580406  0.40126557  0.49318368  0.58556493
  0.67229611  0.74866712  0.86263755  0.90117058  0.92977426  0.95055357
  0.96541314  0.98329067]]

```

Biểu diễn kết quả này trên đồ thị ta có:

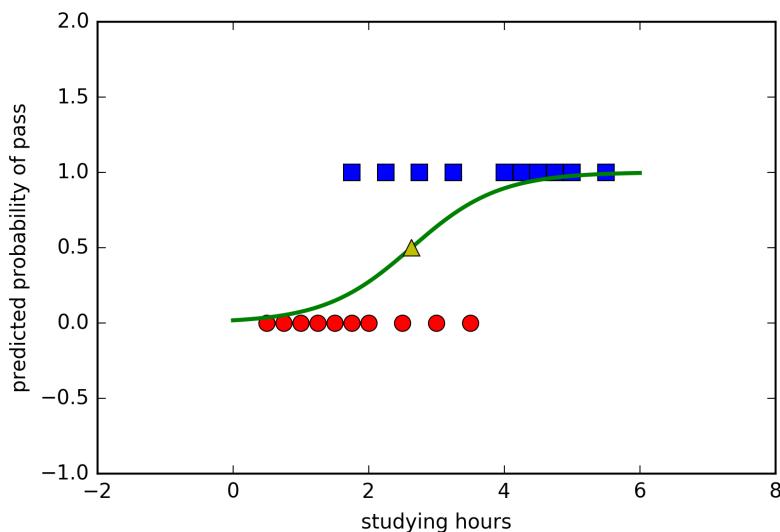
```

X0 = X[1, np.where(y == 0)][0]
y0 = y[np.where(y == 0)]
X1 = X[1, np.where(y == 1)][0]
y1 = y[np.where(y == 1)]

plt.plot(X0, y0, 'ro', markersize = 8)
plt.plot(X1, y1, 'bs', markersize = 8)

xx = np.linspace(0, 6, 1000)
w0 = w[-1][0][0]
w1 = w[-1][1][0]
threshold = -w0/w1
yy = sigmoid(w0 + w1*xx)
plt.axis([-2, 8, -1, 2])
plt.plot(xx, yy, 'g-', linewidth = 2)
plt.plot(threshold, .5, 'y^', markersize = 8)
plt.xlabel('studying hours')
plt.ylabel('predicted probability of pass')
plt.show()

```

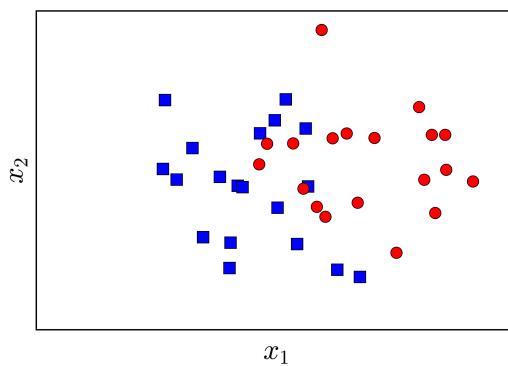


Hình 4: Dữ liệu và hàm sigmoid tìm được.

Nếu như chỉ có hai output là 'fail' hoặc 'pass', điểm trên đồ thị của hàm sigmoid tương ứng với xác suất 0.5 được chọn làm *hard threshold* (ngưỡng cứng). Việc này có thể chứng minh khá dễ dàng (tôi sẽ bàn ở phần dưới).

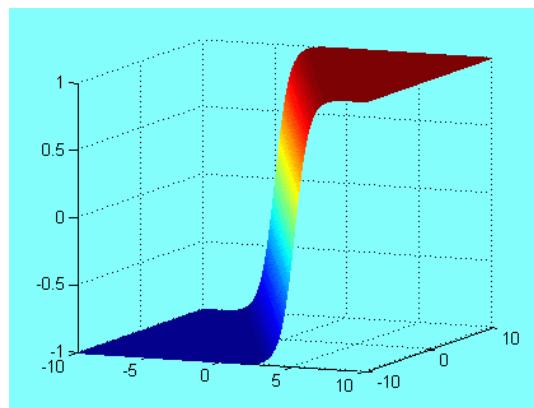
### Ví dụ với dữ liệu 2 chiều

Chúng ta xét thêm một ví dụ nhỏ nữa trong không gian hai chiều. Giả sử chúng ta có hai class xanh-đỏ với dữ liệu được phân bố như hình dưới.

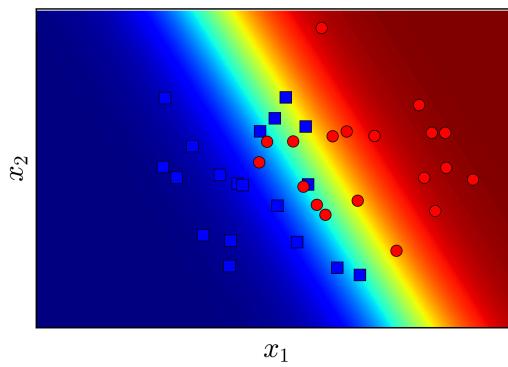


Hình 5: Hai class với dữ liệu hai chiều.

Với dữ liệu đầu vào nằm trong không gian hai chiều, hàm sigmoid có dạng như thác nước dưới đây:

Hình 6: Hàm sigmoid với dữ liệu có chiều là 2. (Nguồn: Biased and non biased neurons ([http://galaxy.agh.edu.pl/~vlsi/AI/bias/bias\\_eng.html](http://galaxy.agh.edu.pl/~vlsi/AI/bias/bias_eng.html)))

Kết quả tìm được khi áp dụng mô hình logistic regression được minh họa như hình dưới với màu nền khác nhau thể hiện xác suất điểm đó thuộc class đỏ. Đỏ hơn tức gần 1 hơn, xanh hơn tức gần 0 hơn.



Hình 7: Logistic Regression với dữ liệu hai chiều.

Nếu phải lựa chọn một *ngưỡng cứng* (chứ không chấp nhận xác suất) để phân chia hai class, chúng ta quan sát thấy đường thẳng nằm nằm trong khu vực xanh lục là một lựa chọn hợp lý. Tôi sẽ chứng minh ở phần dưới rằng, đường phân chia giữa hai class tìm được bởi logistic regression có dạng một đường phẳng, tức vẫn là linear.

## 4. Một vài tính chất của Logistic Regression

Logistic Regression thực ra được sử dụng nhiều trong các bài toán Classification.

Mặc dù có tên là Regression, tức một mô hình cho fitting, Logistic Regression lại được sử dụng nhiều trong các bài toán Classification. Sau khi tìm được mô hình, việc xác định class \$y\$ cho một điểm dữ liệu \$x\$ được xác định bằng việc so sánh hai biểu thức xác suất:

$$P(y = 1|x; \mathbf{w}); \quad P(y = 0|x; \mathbf{w})$$

Nếu biểu thức thứ nhất lớn hơn thì ta kết luận điểm dữ liệu thuộc class 1, ngược lại thì nó thuộc class 0. Vì tổng hai biểu thức này luôn bằng 1 nên một cách gọn hơn, ta chỉ cần xác định xem \$P(y = 1|x; \mathbf{w})\$ lớn hơn 0.5 hay không. Nếu có, class 1. Nếu không, class 0.

### Boundary tạo bởi Logistic Regression có dạng tuyến tính

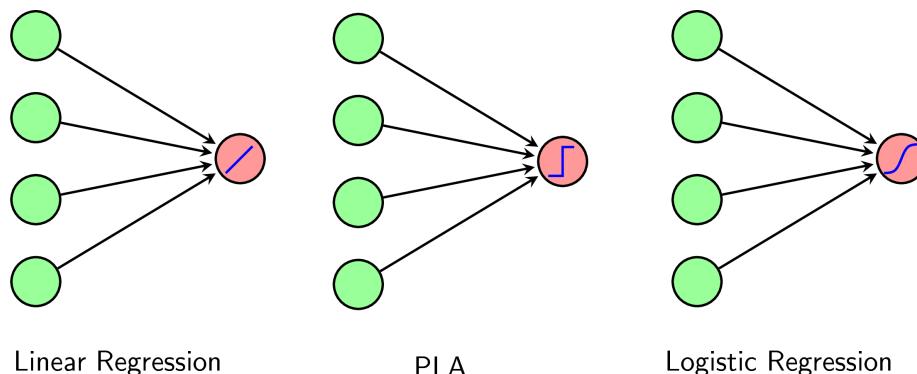
Thật vậy, theo lập luận ở phần trên thì chúng ta cần kiểm tra:

$$\begin{aligned} P(y = 1|x; \mathbf{w}) &> 0.5 \\ \Leftrightarrow \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}} &> 0.5 \\ \Leftrightarrow e^{-\mathbf{w}^T \mathbf{x}} &< 1 \\ \Leftrightarrow \mathbf{w}^T \mathbf{x} &> 0 \end{aligned}$$

Nói cách khác, boundary giữa hai class là đường có phương trình \$\mathbf{w}^T \mathbf{x}\$. Đây chính là phương trình của một siêu mặt phẳng. Vậy Logistic Regression tạo ra boundary có dạng tuyến tính.

## 5. Thảo luận

- Một điểm cộng cho Logistic Regression so với PLA là nó không cần có giả thiết dữ liệu hai class là linearly separable. Tuy nhiên, boundary tìm được vẫn có dạng tuyến tính. Vậy nên mô hình này chỉ phù hợp với loại dữ liệu mà hai class là gần với linearly separable. Một kiểu dữ liệu mà Logistic Regression không làm việc được là dữ liệu mà một class chứa các điểm nằm trong 1 vòng tròn, class kia chứa các điểm bên ngoài đường tròn đó. Kiểu dữ liệu này được gọi là phi tuyến (non-linear). Sau một vài bài nữa, tôi sẽ giới thiệu với các bạn các mô hình khác phù hợp hơn với loại dữ liệu này hơn.
- Một hạn chế nữa của Logistic Regression là nó yêu cầu các điểm dữ liệu được tạo ra một cách *độc lập* với nhau. Trên thực tế, các điểm dữ liệu có thể bị *Ảnh hưởng* bởi nhau. Ví dụ: có một nhóm ôn tập với nhau trong 4 giờ, cả nhóm đều thi đỗ (giả sử các bạn này học rất tập trung), nhưng có một sinh viên học một mình cũng trong 4 giờ thì xác suất thi đỗ thấp hơn. Mặc dù vậy, để cho đơn giản, khi xây dựng mô hình, người ta vẫn thường giả sử các điểm dữ liệu là độc lập với nhau.
- Khi biểu diễn theo Neural Networks, Linear Regression, PLA, và Logistic Regression có dạng như sau:



Hình 8: Biểu diễn Linear Regression, PLA, và Logistic Regression theo Neural network.

- Nếu hàm mất mát của Logistic Regression được viết dưới dạng:

$$J(\mathbf{w}) = \sum_{i=1}^N (y_i - z_i)^2$$

thì khó khăn gì sẽ xảy ra? Các bạn hãy coi đây như một bài tập nhỏ.

- Source code cho các ví dụ trong bài này có thể tìm thấy ở đây ([https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/LogisticRegression/LogisticRegression\\_post.ipynb](https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/LogisticRegression/LogisticRegression_post.ipynb)).

## 6. Tài liệu tham khảo

- [1] Cox, David R. "The regression analysis of binary sequences." Journal of the Royal Statistical Society. Series B (Methodological) (1958): 215-242.
- [2] Cramer, Jan Salomon. "The origins of logistic regression." (2002).
- [3] Abu-Mostafa, Yaser S., Malik Magdon-Ismail, and Hsuan-Tien Lin. Learning from data. Vol. 4. New York, NY, USA:: AMLBook, 2012. (link to course (<http://work.caltech.edu/telecourse.html>))
- [4] Bishop, Christopher M. "Pattern recognition and Machine Learning.", Springer (2006). (book (<http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf>))
- [5] Duda, Richard O., Peter E. Hart, and David G. Stork. Pattern classification. John Wiley & Sons, 2012.
- [6] Andrew Ng. CS229 Lecture notes. Part II: Classification and logistic regression ([https://datajobs.com/data-science-repo/Generalized-Linear-Models-\[Andrew-Ng\].pdf](https://datajobs.com/data-science-repo/Generalized-Linear-Models-[Andrew-Ng].pdf))
- [7] Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie. The Elements of Statistical Learning (<https://statweb.stanford.edu/~tibs/>).

Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (<https://www.facebook.com/groups/257768141347267/>) để nhận được câu trả lời sớm hơn.

Bạn đọc có thể ủng hộ blog qua 'Buy me a coffee' (/buymeacoffee/) ở góc trên bên trái của blog.

Tôi đang trong quá trình viết cuốn sách 'Machine Learning cơ bản I', các bạn có thể đặt trước tại đây (/ebook/). Cảm ơn bạn.

« Bài 9: Perceptron Learning Algorithm (/2017/01/21/perceptron/)

Blog và các bài viết được tạo như thế nào » (/2017/02/02/howdolcreatethisblog/)

39 Comments tiepvu

Login

♥ Recommend 3 Share

Sort by Best

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS (?)

Name



Shay Patrick Cormac • 2 months ago

Anhơi sao em thấy nó đỡ lỗi rãnh nhỏ thì kết quả nó vẫn rãnh sai

Xin lỗi, sau khi thay đổi tên của chúng tôi thì kết quả ngày càng tệ

mà eta càng lớn thì kết đường sigmoid nó càng cứng

[1 ^](#) | [v](#) • Reply • Share >



Tiep Vu Huu Mod → Shay Patrick Cormac • 2 months ago

Một trong những việc chính của người làm thực nghiệm là phải chọn ra eta phù hợp :).

[^](#) | [v](#) • Reply • Share >



Shay Patrick Cormac → Tiep Vu Huu • 2 months ago

vậy có thuật toán optimization nào mà không cần phải chọn eta không anh

[^](#) | [v](#) • Reply • Share >



Tiep Vu Huu Mod → Shay Patrick Cormac • 2 months ago

Có thể có, nhưng trong hiểu biết của anh, với các bài Neural Network thì các thuật toán đều dựa trên Gradient Descent. Và trong Gradient Descent thì kiểu gì cũng phải chọn learning rate (eta).

[^](#) | [v](#) • Reply • Share >



Shay Patrick Cormac → Tiep Vu Huu • 2 months ago

..

[^](#) | [v](#) • Reply • Share >



Thang Vu • 3 months ago

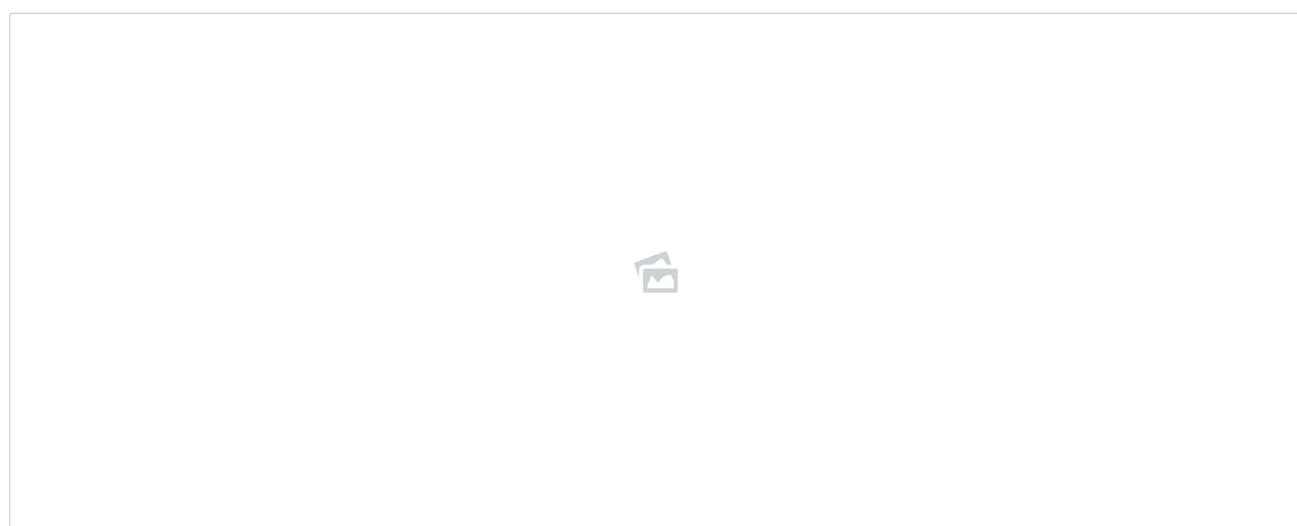
Theo em hàm cost J(w) là hàm hợp của hàm tuyến tính, hàm sigmoid, và cross entropy. Trong đó hàm sigmoid ko phải convex. Trong bài có áp dụng SGD, vậy em hỏi chút là J có phải là convex theo w ko. Và nếu có thì có cách nào bằng toán học chứng minh nó là convex theo w không ngoài việc plot.

[1 ^](#) | [v](#) • Reply • Share >



Thang Vu → Thang Vu • 3 months ago

Đây là hình plot 1 ví dụ đơn giản. Nhiều khả năng nó là convex. Còn chứng minh như nào e chưa rõ. :D



[^](#) | [v](#) • Reply • Share >



Tiep Vu Huu Mod → Thang Vu • 3 months ago

Anh chưa đặt bút xuống thử. Em thử xem chứng minh này xem:

<http://mathgotchas.blogspot...>

[^](#) | [v](#) • Reply • Share >



Thang Vu → Tiep Vu Huu • 2 months ago

vâng. em cảm ơn anh :D

[^](#) | [v](#) • Reply • Share >



Hattieu • 18 days ago

Anh cho em hỏi là Logistic Regression có thể dùng trong bài toán hồi quy ko ạ?

[^](#) | [v](#) • Reply • Share >



Tiep Vu Huu Mod → Hattieu • 18 days ago

Có, nếu đầu ra y có dạng xác suất, tức là một số thực trong khoảng (0,1). Nhưng thường thì Logistic Regression được dùng nhiều trong classification.

[^](#) | [v](#) • Reply • Share >



Dung Lam • 2 months ago



Dũng Lâm • a month ago  
Chào anh em có một câu hỏi. Có phải trong thực tế thì người ta sử dụng neural network hơn là những cách khác là vì nó cho phép mình không cần phải tự feature engineering mà vẫn có thể cho ra được kết quả tốt nhưng bù lại là computation thì cao? Vậy thì theo anh cái computation cao như vậy có phải là vấn đề hay không vì theo e nghĩ CPU hiện nay có thể đạt đến sức mạnh "vô song" rồi.

^ | v • Reply • Share ›



Tiep Vu Huu Mod → Dũng Lâm • a month ago

Em xem mọi người thảo luận ở đây nhé:  
<https://www.facebook.com/groups/machinelearningcoba.../1506088700233200>

^ | v • Reply • Share ›



Trung Thanh Nguyen • 2 months ago

cho em hỏi với dữ liệu nonelinear và với mô hình hồi quy ( tức dữ liệu có thể là xếp thành 1 đường cong chứ k phải đường thẳng ) thì có thể áp dụng Logistics Regression được không ạ , nếu có thì như thế nào và nếu không thì nên dùng cái gì ạ ?

^ | v • Reply • Share ›



Tiep Vu Huu Mod → Trung Thanh Nguyen • 2 months ago

Logistic Regression được dùng chủ yếu trong các bài toán classification.

Nếu em biết dạng của đường cong thì Linear Regression có thể là một lựa chọn. Em đọc thêm phần này xem:  
<https://machinelearningcoba...>

^ | v • Reply • Share ›



Trung Thanh Nguyen → Tiep Vu Huu • 2 months ago

e mới học machine learning nên còn nhiều điều k biết :) , nếu hỏi ngu quá a cũng đừng trách ạ

^ | v • Reply • Share ›



Tiep Vu Huu Mod → Trung Thanh Nguyen • 2 months ago

Không sao em, ai ban đầu học cũng mông lung.

Nếu em có câu hỏi, em có thể đưa lên Forum cho mọi người cùng thảo luận. Comment ở đây ít người để ý. Cảm ơn em.

^ | v • Reply • Share ›



Trung Thanh Nguyen → Tiep Vu Huu • 2 months ago

em đã đọc trước rồi ạ , nhưng em thắc mắc là dữ liệu thì có nhiều chiều , k thể thể hiện data trên đồ thị 2 chiều được nên ta không thể biết data có tuyến tính hay không , nếu không tuyến tính thì làm cách nào để biết được dạng của đường / mặt phẳng đó ? VD :  $y = w_1x_1 + w_2x_2 + w_3x_2^2 + w_4\sin(x_3)$  ? làm cách nào để biết được rằng trong hàm số có  $x_1x_2$  ,  $x_2^2$  ,  $\sin$  ,  $\cos$  ... ạ ,

^ | v • Reply • Share ›



Tiep Vu Huu Mod → Trung Thanh Nguyen • 2 months ago

Nếu có nhiều dữ liệu em có thể thử với Neural networks. Về cơ bản thì Neural Nets có thể cho ra một hàm bất kỳ nếu em có đủ dữ liệu (Xem Bài 14). Thế nào là đủ thì phải thử xem mô hình có chạy không.

Ngoài ra em có thể đọc thêm về Kernel Methods (Bài 21 có đề cập một chút).

^ | v • Reply • Share ›



Albert Anh Thành • 2 months ago

em có thắc mắc là nếu càng nhiều feature thì càng nhiều local minimum phải không anh ?

^ | v • Reply • Share ›



Tiep Vu Huu Mod → Albert Anh Thành • 2 months ago

Em cần nói rõ local minimum của hàm số nào. Như comment ở trên có đưa, hàm mất mát của logistic regression là hàm lồi, nên local minimum cũng là global minimum. Nếu 1 hàm là hàm lồi chặt thì nó chỉ có 1 local minimum và cũng là global minimum.

Nên cần phải đặt nó trong văn cảnh.

^ | v • Reply • Share ›



Albert Anh Thành → Tiep Vu Huu • 2 months ago

thực sự em vẫn chưa hiểu lắm, có cách nào để plot J(w) so với w1 và w2 trên không gian ba chiều khi w chỉ có 2 giá trị không anh. em nghĩ plot thì sẽ hiểu được một cách trực quan hơn

^ | v • Reply • Share ›



CTVR • 3 months ago

Bài toán trong phần thảo luận mà a nói. Em làm thì thấy không có khó khăn gì lớn ngoài hàm  $z = f(s)$  lấy như nào hợp lý hay lấy như nào cho nó đẹp nó gọn. Hiện tại em lấy theo trong bài thì ra phương trình bậc 3 theo z. Không biết có sai không?

Kq:  $(z-y)^*z^*(1-z)$

^ | v • Reply • Share ›



Kim Nguyen • 3 months ago



Chào bạn,

Trong bài viết bạn có nói là logistic regression ko làm việc dc trong trường hợp phi tuyến tính, các điểm của 1 class nằm trong một vòng tròn chẵng hạn. Tuy nhiên, theo mình biết linear regression chỉ linear với các hệ số w, còn đối với x vẫn có thể là hàm mũ (lúc này ta coi  $x(i)^2$  là một biến độc lập chẵng hạn). Vì vậy, nếu ta chọn hàm  $w^T x$  phù hợp, logistic regression hoàn toàn có thể phân loại dữ liệu trong vòng tròn. Bạn nghĩ sao?

Mình thấy trang của bạn cực kỳ hay, rất cảm ơn bạn!

[^](#) [v](#) [Reply](#) [Share](#)



Tiếp Võ Huu Mod → Kim Nguyen • 3 months ago

Chắc chắn bạn. Nếu bạn thay đổi  $x$  đi thì ta lại được cách phân lớp phi tuyến. Bạn có thể đọc thêm bài Kernel SVM: <https://machinelearningcoba...>

Để có cái nhìn rộng hơn :)

[^](#) [v](#) [Reply](#) [Share](#)



Kim Nguyen → Kim Nguyen • 3 months ago

By the way, as I really enjoyed reading your blog, I am very pleased to buy you a coffee, via Paypal :) Let enjoy it and keep working on the good work for our community!!!

[^](#) [v](#) [Reply](#) [Share](#)



Tiếp Võ Huu Mod → Kim Nguyen • 3 months ago

Cảm ơn bạn đã quan tâm và ủng hộ blog ^^.

[^](#) [v](#) [Reply](#) [Share](#)



Mạnh Quyền Nguyễn • 3 months ago

" Khi ta nói linearly separable là khi ta chỉ dùng dữ liệu đầu vào  $x$  " chỗ này là sao ạ em thấy ở dưới có sử dụng cả  $y$  mà

[^](#) [v](#) [Reply](#) [Share](#)



Tiếp Võ Huu Mod → Mạnh Quyền Nguyễn • 3 months ago

Việc dùng  $y$  là để minh họa thêm thôi bạn. Đến lúc vẽ cái đường phân chia ra thì chỉ vẽ trên không gian của  $X$  thôi.

[^](#) [v](#) [Reply](#) [Share](#)



QUOC HUY NGUYEN • 4 months ago

Cho mình hỏi là tại sao ta phải mix\_id bằng mix\_id = np.random.permutation(N). Minh thay bằng range(N) thì kết quả không đúng lắm

[^](#) [v](#) [Reply](#) [Share](#)



Tiếp Võ Huu Mod → QUOC HUY NGUYEN • 4 months ago

Chỗ đó là để dữ liệu ngẫu nhiên sau mỗi vòng lặp. Đây cũng là lý do mà phương pháp tối ưu được gọi là Stochastic Gradient Descent. Bạn có thể đọc ở Bài 8.

[^](#) [v](#) [Reply](#) [Share](#)



QUOC HUY NGUYEN → Tiếp Võ Huu • 4 months ago

Cảm ơn bạn. Minh học được rất nhiều từ blog của bạn

1 [^](#) [v](#) [Reply](#) [Share](#)



vanchung1995 • 7 months ago

Nếu dùng hàm mất mát là tổng bình phương, thì không áp dụng được xác suất vì nó không nằm trong đoạn [0,1] và lợi ích của hàm sigmoid, nó không liên tục nên không tìm nghiệm tối ưu chính xác được phải không anh?

[^](#) [v](#) [Reply](#) [Share](#)



Tiếp Võ Huu Mod → vanchung1995 • 7 months ago

Bạn đọc tiếp Bài 13 về cross-entropy sẽ thấy rõ.

[^](#) [v](#) [Reply](#) [Share](#)



vanchung1995 • 7 months ago

Phương pháp này hay thế! Tự nhiên nghĩ ra cái xác suất!

[^](#) [v](#) [Reply](#) [Share](#)



Thanh Nguyen • 8 months ago

nếu hàm mất mát của logistic regression viết dưới dạng  $J(\mathbf{w}) = \sum_{i=1}^N (y_i - z_i)^2$  thì cái hàm  $f(s) = s$ , lại chuyển về giống với linear regression đúng ko ạ?

[^](#) [v](#) [Reply](#) [Share](#)



Tiếp Võ Huu Mod → Thanh Nguyen • 4 months ago

Nhìn qua hàm mất mát thì có vẻ giống, nhưng đạo hàm theo  $w$  lại khác hoàn toàn.

[^](#) [v](#) [Reply](#) [Share](#)



DaoTuan • 8 months ago

với dữ liệu 1 chiều sao ta lại phải mở rộng  $X$  hả anh?

```
# extened data
X = np.concatenate((np.ones((1, X.shape[1])), X), axis = 0)
^ | v • Reply • Share >
```



**Tiep Vu Huu** Mod → DaoTuan • 8 months ago

Với dữ liệu bao nhiêu chiều thì mô hình tuyến tính cũng có dạng:

$w_0 + w_1 x_1 + \dots + w_d x_d$ .

Hệ số tự do  $w_0$  chính là ứng với  $x_0 = 1$ , vậy nên ta cần mở rộng  $x$  bằng cách thêm phần tử 1 về phía trước.

^ | v • Reply • Share >

ALSO ON TIEPVU

## Machine Learning cơ bản

11 comments • 8 months ago

**CTVR** — Em cũng thấy yêu cầu 64 bit trên win. Chắc em chạy với linux vậy.  
P/s: Giờ em mới đọc xong loạt bài về convex trên file pdf. Hồi mới vào ...

[Subscribe](#) [Add Disqus to your site](#) [Add Disqus](#) [Privacy](#)

## ebook Machine Learning cơ bản

26 comments • 3 months ago

**Xuân Xuân** — Anh ơi, cái email trường em cấp ý, em test nó không cho các mail ngoài gửi vào:"Hi. This is the qmail-send program at vnu.edu.vn.I'm ...

Total visits: