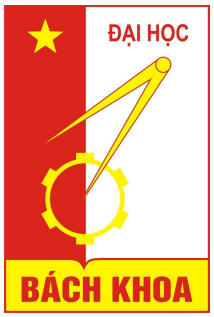
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG




# BÁO CÁO MÔN HỌC
# XỬ LÝ NGÔN NGỮ TỰ NHIÊN

# SPEECH AND LANGUAGE PROCESSING

Giảng viên hướng dẫn: **TS. NGUYỄN KIÊM HIẾU**

Học viên thực hiện: **LÊ HOÀNG NGÂN**

Mã số học viên: **20162886**

**HÀ NỘI, 07/2020**

# Mục lục

# Chương 1

# Biểu thức chính quy, Chuẩn hóa văn bản, Khoảng cách Edit Distance

*User:* *You are like my father in some ways.*
*ELIZA:* *WHAT RESEMBLANCE DO YOU SEE*
*User:* *You are not very aggressive but I think you don't want me to notice that.*
*ELIZA:* *WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE*
*User:* *You don't argue with me.*
*ELIZA:* *WHY DO YOU THINK I DON'T ARGUE WITH YOU*
*User:* *You are afraid of me.*
*ELIZA:* *DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU*

Weizenbaum (1966)

Cuộc đối thoại trên là của ELIZA, một hệ thống xử lý ngôn ngữ tự nhiên có thể thực hiện một cuộc trò chuyện với người dùng bằng cách học theo các phản hồi của nhà tâm lý trị liệu Rogerian (Weixenbaum, 1966). ELIZA mô phỏng một bác sĩ tâm lý với công việc lắng nghe và hỏi kĩ càng về những câu chuyện của bệnh nhân, giúp cho họ có thể dần dần nói ra những điều mà họ không thể chia sẻ. Có thể thấy, ELIZA thực hiện đoạn hội thoại một cách thực sự rất tự nhiên, để làm được điều này, một trong những thuật toán được áp dụng là thuật toán **"pattern matching"**. Để xác định được câu nói của người dùng rơi vào mẫu nào (*pattern*, ELIZA sẽ bắt đầu dựa trên các từ khóa (*keyword*) xuất hiện trong câu nói của người dùng. Điều này đồng nghĩa với việc hệ thống đã phải định sẵn các từ khóa, đồng thời sắp xếp thứ tự ưu tiên của chúng một các hợp lý. Với mỗi mẫu, hệ thống sẽ có sẵn $3-5$ kiểu trả lời để có thể lựa chọn ngẫu nhiên. Nếu không có từ khóa nào được xác định, phản hồi của ELIZA thường sẽ là các mẫu câu: *"PLEASE GO ON"*, *THAT'S VERY INTERESTING*, hoặc *I SEE*. Điều này như một gợi ý để người dùng tiếp tục đưa ra câu hội thoại mới. Nhờ vậy, ELIZA đã thành công đáng kể, nhiều người dùng đã thực sự tin rằng ELIZA hiểu vấn đề của họ. ELIZA là *chatbot* đầu tiên và thành công nhất, là một hệ thống *chatbot*

điển hình, được xem là tiền đề phát triển thị trường *chatbot* như ngày nay.

Chúng ta sẽ bắt đầu tìm hiểu công cụ quan trọng nhất trong việc mô tả các mẫu văn bản: biểu thức chính quy. Biểu thức chính quy (**regular expression**) là một nhóm các kí tự, kí hiệu được sử dụng để tìm kiếm văn bản (*text*). Sau đó, chúng ta tìm hiểu về chuẩn hóa văn bản (**text normalization**), trong đó, biểu thức chính quy là một phần quan trọng. Chuẩn hóa văn bản là đưa văn bản từ các dạng không đồng nhất về cùng một dạng tiêu chuẩn. Trong những quy trình xử lý chuẩn hóa văn bản, tách từ là một quy trình quan trọng. Trong tiếng Việt, dấu cách (*space*) không được sử dụng như một ký hiệu phân cách từ (vì có một số từ ghép, nếu tách bằng dấu cách sẽ mất đi ý nghĩa của từ). Đối với các từ tiếng Anh, thường được tách nhau bằng khoảng trắng, nhưng trong một số trường hợp, tách bằng khoảng trắng sẽ mất đi ý nghĩa của từ (Ví dụ, *New York* hay *rock 'n' roll*). Một phần khác của chuẩn hóa văn bản là chuẩn hóa từ, mục đích là xác định các từ có cùng một gốc (*root*), mặc dù các từ có cấu tạo khác nhau. Ví dụ trong tiếng Anh, từ *sang, sung, sings* đều là các hình thức khác nhau của động từ *sing*, từ *sing* được gọi là **lemma**.Ngoài ra, chuẩn hóa văn bản cũng bao gồm cả phân đoạn câu (**sentence segmentation**), chia một văn bản thành từng câu riêng lẻ, sử dụng một số tín hiệu như dấu chấm hoặc dấu chấm than.

Cuối cùng, trong chương này, chúng ta tìm hiểu các thuật toán xác định khoảng cách **edit distance** để so sánh, đo mức độ tương tự của hai chuỗi dựa trên số các chính sửa (chèn, xóa, thay thế) cần để thay đổi từ chuỗi này thành chuỗi còn lại. **Minimum Edit Distance** là một thuật toán với nhiều ứng dụng áp dụng trong xử lý ngôn ngữ tự nhiên, như sửa lỗi sai chỉnh tả hay nhận dạng giọng nói.

## 1.1 Biểu thức chính quy

### 1.1.1 Một số biểu thức chính quy cơ bản

Mẫu biểu thức chính quy đơn giản nhất là một chuỗi gồm các kí tự. Ví dụ, để tìm kiếm *woodchuck*, biểu thức chính quy là /woodchuck/. Biểu thức /Buttercup/ khớp với bất kì chuỗi nào chứa chuỗi con *Buttercup*, ví dụ chuỗi *I'm called little Buttercup*. Chuỗi tìm kiếm có thể bao gồm các kí tự đặc biệt (/!/) hoặc một chuỗi các kí tự (/urgl/).

| RE | Example Patterns Matched |
|---|---|
| /woodchucks/ | "interesting links to woodchucks and lemurs" |
| /a/ | "Mary Ann stopped by Mona's" |
| /!/ | "You've left the burglar behind again!" said Nori |

Hình 1.1: Một số tìm kiếm regex đơn giản

Trong biểu thức chính quy, ta cần phân biệt chữ thường và chữ in hoa (/s/ khác với /S/). Như vậy, với biểu thức /woodchucks/ sẽ không khớp với chuỗi *Woodchucks*. Để giải quyết vấn đề này, chúng ta sử dụng dấu ngoặc vuông [ ]. Chuỗi kí tự bên trong

ngoặc vuông được chỉ định khớp với bất kì chuỗi nào chứa chuỗi con bên trong dấu ngoặc vuông. Ví dụ, hình 1.2 cho thấy biểu thức /[wW]/ khớp với các chuỗi chứa w hoặc W.

| RE | Match | Example Patterns |
|---|---|---|
| /[wW]oodchuck/ | Woodchuck or woodchuck | "Woodchuck" |
| /[abc]/ | 'a', 'b', *or* 'c' | "In uomini, in soldati" |
| /[1234567890]/ | any digit | "plenty of 7 to 5" |

Hình 1.2: Cách sử dụng dấu ngoặc vuông [ ] chỉ định phân biệt các ký tự

Biểu thức chính quy /[1234567890]/ chỉ định bất kì chữ số nào, biểu thức /[ABCDEFGHỊKLMNOPQRSTUVWXYZ]/ chỉ định bất kì chữ cái viết hoa nào. Có thể thấy cách biểu diễn như vậy có phần bất tiện, khi đó với trường hợp có một chuỗi xác định được liên kết với một bộ ký tự, dấu ngoặc vuông có thể được sử dùng cùng dấu gạch ngang (-) để chỉ định bất kì ký tự nào trong phạm vi. Ví dụ, biểu thức $/[2-5]/$ chỉ định bất kì một trong các số 2, 3, 4, 5, biểu thức /[b-g]/ chỉ định một trong các ký tự *b, c, d, e, f, g*. Một số ví dụ khác được biểu diễn trong hình 1.3

| RE | Match | Example Patterns Matched |
|---|---|---|
| /[A-Z]/ | an upper case letter | "we should call it 'Drenched Blossoms' " |
| /[a-z]/ | a lower case letter | "my beans were impatient to be hoed!" |
| /[0-9]/ | a single digit | "Chapter 1: Down the Rabbit Hole" |

Hình 1.3: Cách sử dụng dấu ngoặc [ ] và dấu gạch ngang - để chỉ định một phạm vi

Nếu dấu mũ ^ nằm ở ngay sau dấu [ của cặp [ ] thì nó mang ý nghĩa tạo ra tập hợp ký tự loại trừ (phủ định). Ví dụ, biểu thức /[^n]hanh/ có nghĩa là bất kì ký tự nào ngoại trừ kí tự *n*, theo sau bởi *h*, tiếp theo bởi *a, n* và *h*. Điều này chỉ đúng khi ^ đứng ngay sau dấu [, nếu sử dụng ở những vị trí khác trong cặp [ ], nó đơn giản là một dấu mũ ^. Hình 1.4 cho thấy một số ví dụ:

| RE | Match (single characters) | Example Patterns Matched |
|---|---|---|
| /[^A-Z]/ | not an upper case letter | "Oyfn pripetchik" |
| /[^Ss]/ | neither 'S' nor 's' | "I have no exquisite reason for't" |
| /[^\.]/ | not a period | "our resident Djinn" |
| /[e^]/ | either 'e' or '^' | "look up ^ now" |
| /a^b/ | the pattern 'a^b' | "look up a^ b now" |

Hình 1.4: Cách sử dụng dấu mũ ^ trong biểu thức chính quy

Vậy làm thế nào để có thể tùy chọn một chuỗi có thể chứa một chuỗi con xác định hay không? Ví như tùy chọn *s* trong *woodchuck* và *woodchucks*, chúng ta sử dụng dấu hỏi **(?)**, tùy chọn có hay không cho kí tự phía trước, tức là cho phép ký tự trước nó lặp lại 0 hoặc 1 lần. Một số ví dụ khác như trong hình 1.5

| RE | Match | Example Patterns Matched |
|---|---|---|
| /woodchucks?/ | woodchuck or woodchucks | "woodchuck" |
| /colou?r/ | color or colour | "colour" |

Hình 1.5: Dấu hỏi ? đánh dấu tùy chọn của biểu thức hoặc ký tự trước nó

Xét cách biểu diễn tiếng kêu của một con cừu:

*baa!*
*baaa!*
*baaaa!*
*baaaaa!*
...

Chuỗi chứa các ký tự *a, b*, theo sau *b* là ít nhất hai ký tự *a*, tiếp đến là dấu chấm than (**!**). Toán tử cho phép chúng ta biểu diễn biểu thức chính quy khớp với các chuỗi như trên là *****, thường được gọi là **Kleene ***, cho phép ký tự hoặc biểu thức chính quy trước nó lặp lại 0 hoặc nhiều lần. Vậy /a*/ khớp với tất cả các chuỗi chứa 0 hoặc nhiều ký tự *a*. Biểu thức chính quy /aa*/ khớp với các chuỗi chứa ít nhất 1 ký tự *a*. Ngoài ra, có một cách dễ dàng hơn để chỉ định ít nhất 1 ký tự, chúng ta sử dụng **Kleene +**.

Một ký tự biểu diễn rất quan trọng là dấu chấm (/./), khớp với bất kỳ ký từ đơn vào ngoại trừ ký tự xuống dòng, ví dụ được biểu diễn trong hình 1.6

| RE | Match | Example Matches |
|---|---|---|
| /beg.n/ | any character between *beg* and *n* | begin, beg'n, begun |

Hình 1.6: Cách sử dụng dấu chấm . để chỉ định bất kỳ ký tự

Ký tự /./ thường được sử dụng cùng **Kleene ***, biểu diễn "bất kì chuỗi ký tự nào". Ví dụ, giả sử chúng ta cần tìm bất kì dòng nào thỏa mãn chuỗi *abab* xuất hiện 2 lần, có thể biểu diễn bằng biểu thức chính quy /abab.*abab/.

**Anchor** là một tập các ký tự đặc biệt gán các biểu thức chính quy đến các vị trí cụ thể trong chuỗi. Các anchor phổ biến nhất là dấu mũ ^ - điểm bắt đầu một dòng và ký hiệu dollar $ - điểm kết thúc một dòng. Dấu mũ ^ kiểm tra sự phụ hợp nếu ký tự đầu tiên của chuỗi khớp mẫu, ví dụ mẫu /The/ khớp với từ *The only*. Như vậy, chúng ta cần phân biệt 3 cách sử dụng của dấu mũ ^: điểm bắt đầu của một dòng, tập hợp ký tự loại trừ khi sử dụng cùng dấu ngoặc vuông [ ] và chỉ đơn thuần là dấu mũ^. Ký hiệu dollar $ cho biết kết thúc dòng phải thỏa mãn mẫu ở phía trước $.

Ngoài ra, 2 anchor khác thường xuyên được sử dụng: \b khớp với ký tự biên, và \B khớp với ký tự không phải ký tự biên. Ký tự biên là một ký tự giả, nó khớp với vị trí mà một ký tự không được theo sau hoặc đứng trước bởi một ký tự khác. Lưu ý rằng một ký tự biên được khớp sẽ không bao gồm trong kết quả so khớp. Nói cách khác, độ dài của một ký tự biên là 0. Ví dụ, mẫu /\b99\b/ khớp với chuỗi 99 trong câu *There are* 99 *bottles of beer on the wall* những sẽ không khớp chuỗi 99 trong câu *There are*

299 *bottles of beer on the wall*, nhưng nó sẽ khớp 99 trong $99 do 99 theo sau ký hiệu dollar $ không phải là một chữ số, kí hiệu gạch dưới hay một ký tự chữ cái.

### 1.1.2 Toán tử thay thế, nhóm, ưu tiên toán tử

Giả sử người dùng cần tìm kiếm các thông tin về vật nuôi, thường sẽ đặc biệt quan tâm 2 vật nuôi là *dog* (chó) và *cat* (mèo). Với trường hợp này ta sử dụng toán tử thay thế, ký hiệu |, khi đó mẫu /cat|dog/ khớp với *cat* trong chuỗi *lovely cat* và *dog* trong chuỗi *lovely dog*.

Đôi khi chúng ta sử dụng toán tử thay thế trong một câu. Ví dụ, giả sử người dừng muốn tìm kiếm thông tin về *pet fish* (cá cảnh). Vậy làm thế nào để chỉ định cả 2 chuỗi *guppy* và *guppies*? Có thể thấy, chuỗi *guppy* được ưu tiên hơn toán tử thay thế |. Để toán tử thay thể chỉ áp dụng cho một chuỗi cụ thể, chúng ta sử dụng toán tử dấu ngoặc đơn ( ). Khi đó, biểu thức chính quy đoán nhận chuỗi là /gupp(y|ies)/. Toán tử dấu ngoặc đơn ( ) cũng được sử dụng cùng toán tử Kleene*. Ví dụ, để biểu diễn một chuỗi *Column 1 Column 2 Column 3*, biểu thức chính quy tương ứng là `/(Column [0 − 9]+ *)*/`.

| | |
|---|---|
| Parenthesis | () |
| Counters | * + ? {} |
| Sequences and anchors | the ^my end$ |
| Disjunction | \| |

Hình 1.7: Bảng thứ tự ưu tiên các toán tử của biểu thức chính quy

**Ví dụ 1.1.1.** *Xây dựng một biểu thức chính quy tìm các bài báo tiếng Anh chứa "the".*

`/the/`

Có thể thấy mẫu này không chính xác vì nó bỏ lỡ trường hợp `The`.

`/[tT]he/`

Biểu thức chính quy có thể trả về các chuỗi không chính xác như *other* hoặc *theology*.

`\b[tT]he\b`

Biểu thức bỏ lỡ các trường hợp như *the_* hoặc *the25*, những chuỗi có ký tự số hoặc ký tự đặc biệt ở trước và sau *the*.

`/[^a-zA-Z][tT]he[^a-zA-Z]/`

Không tìm thấy từ *The* (từ viết hoa khi bắt đầu một dòng).

Biểu thức chính quy phù hợp nhất:

`/(^|[^a-zA-Z])[tT]he([^a-zA-Z]|$)/`

Quá trình xây dựng một biểu thức chính quy như ở trên bằng việc sửa lỗi các biểu thức chính quy cơ bản, ở đây chúng ta phân loại thành 2 loại lỗi: dương tính giả (False Positive - FP), âm tính giả (False Nagative). Dương tính giả là việc chỉ định nhầm

những chuỗi không đúng, đối với ví dụ trên, việc chỉ định cả các chuỗi như *other* hoặc *theology* là một trong những dương tính giả. Âm tính giả là việc bỏ lỡ những chuỗi đúng, ví dụ như việc bỏ là chuỗi *The*. Để giảm hai lỗi này trong một ứng dụng xử lý ngôn ngữ tự nhiên, người ta thường sử dụng:

- Tăng **precision** (giảm dương tính giả).
- Tăng **recall** (giảm âm tính giả).

**Ví dụ 1.1.2.** *Xây dựng ứng dụng hộ trợ người dùng mua máy tính trên web với các điều kiện: dung lượng lớn hơn* $6GHz$ *và* $500GB$*, giá dưới* 1000$.

Xây dựng biểu thức chính quy theo từng bước như ví dụ 1.1.1. Đầu tiên ta xây dựng biểu thức chính quy biểu diễn chuỗi yêu cầu về giá cả:
`/\b$[0-9]+(\.[0-9][0-9])?\b/`
Biểu thức chính quy biểu diễn chuỗi thông số tốc độ xử lý:
`/\b$[0-9]+*(GHz|[Gg]igahert)\b/`
Nếu thông số tốc độ xử lý là số thập phân (ví dụ $5.5GHz$):
`/\b$[0-9]+(\.[0-9][0-9])?*(GHz|[Gg]igahert)\b/`

### 1.1.3 Một số toán tử khác

Hình 1.8 cho thấy một số toán tử phổ biến chỉ định cho các phạm vi. Bên cạnh Kleene+ và Kleene*, chúng ta có thể chỉ định cụ thể số lần xuất hiện của mẫu bằng cách sử dụng dấu ngoặc nhọn . `/n/` chỉ định ký tự đứng trước phải xuất hiện $n$ lần, $n$ là một số nguyên dương. Ví dụ, `/a2/` không khớp với *a* trong *candy*, nhưng nó khớp với tất cả ký tự *a* trong *caandy*.

| RE | Expansion | Match | First Matches |
|---|---|---|---|
| \d | [0-9] | any digit | Party␣of␣5 |
| \D | [^0-9] | any non-digit | Blue␣moon |
| \w | [a-zA-Z0-9_] | any alphanumeric/underscore | Daiyu |
| \W | [^\w] | a non-alphanumeric | !!!! |
| \s | [␣\r\t\n\f] | whitespace (space, tab) | |
| \S | [^\s] | Non-whitespace | in␣Concord |

Hình 1.8: Ký hiệu cho các bộ ký tự phổ biến

Ngoài ra, `/n, m/` chỉ định từ $n$ đến $m$ lần xuất hiện của ký tự hoặc biểu thức chính quy trước nó, và `/n,/` chỉ định ít nhất $n$ lần xuất hiện của ký tự hoặc biểu thức chính quy trước đó. Một số toán tử đếm được tóm tắt trong hình 1.9

Cuối cùng, một số ký tự đặc biệt được biểu diễn bằng cách thêm dấu gạch chéo \ trước nó (hình 1.10). Một trong những ký tự phổ biến nhất là ký tự xuống dòng `\n` và ký tự tab `\t`.

| RE | Match |
|---|---|
| * | zero or more occurrences of the previous char or expression |
| + | one or more occurrences of the previous char or expression |
| ? | exactly zero or one occurrence of the previous char or expression |
| {n} | *n* occurrences of the previous char or expression |
| {n,m} | from *n* to *m* occurrences of the previous char or expression |
| {n,} | at least *n* occurrences of the previous char or expression |

Hình 1.9: Toán tử đếm của biểu thức chính quy

| RE | Match | First Patterns Matched |
|---|---|---|
| * | an asterisk "*" | "K*A*P*L*A*N" |
| \. | a period "." | "Dr. Livingston, I presume" |
| \? | a question mark | "Why don't they come and lend a hand?" |
| \n | a newline | |
| \t | a tab | |

Hình 1.10: Một số ký tự đặc biệt

### 1.1.4 Thay thế biểu thức chính quy, Capture Group, ứng dụng ELIZA

Một toán tử quan trọng của biểu thức chính quy là thay thế. Toán tử thay thế `s/regexp1/pattern/` được sử dụng trong Python và Unix như `vim` hay `sed`, cho phép biểu thức chính quy biểu diễn việc thay thế chuỗi này bằng một chuỗi khác, ví dụ: `s/colour/color/`.

Toán tử thường được sử dụng để tham chiếu đến một chuỗi con cụ thể trong một đoạn văn bản. Giả sử chúng ta muốn đặt dấu ngoặc $<$ $>$ quanh tất cả các số nguyên trong một văn bản, ví dụ như đổi chuỗi *the* 35 *boxes* thành *the* $<$35$>$ *boxes*. Biểu thức chính quy tương ứng:

`s/([0 − 9]+)/<\1>/`

Ở đây, \1 được thay thế bằng bất kỳ chuỗi nào khớp với chuỗi trong cặp dấu ngoặc ( ) đầu tiên. Việc sử dụng dấu ngoặc đơn ( ) để lưu trữ một mẫu trong bộ nhớ được gọi là **capture group**. Mỗi khi một capture group được sử dụng, biểu thức chính quy sẽ được lưu trữ và đánh dấu theo thứ tự. Như vậy, nếu ta sử dụng 2 capture group khác nhau, \2 được thay thế bằng bất kỳ chuỗi nào khớp với capture group thứ 2, ví dụ:

`/the (.*)er they (.*), the \1er we \2/`

Biểu thức chính quy trên khớp với chuỗi *the faster they ran, the faster we ran* nhưng không khớp với chuỗi *the faster they ran, the faster we ate*. Tương tự với \3 là \4.

Như ta đã biết, capture group sẽ xác định giá trị trả về cho từng biểu thức con, nhưng đôi lúc ta lại muốn biểu thức con đó không xuất hiện trong kết quả trả về thì làm thế nào? Để giải quyết vấn đề này, ta dùng **non-capturing group** trong biểu thức chính quy, ký hiệu: `(?: pattern)`. Ví dụ:

`/(?:some|a few) (people|cats) like some \1/`

Biểu thức chính quy trên khớp với chuỗi *some cats like some people* nhưng sẽ không khớp *some people like some a few*.

Toán tử thay thế và capture group được ứng dụng nhiều trong việc xây dựng chatbot đơn giản như ELIZA (*Weizenbaum, 1966*). ELIZA mô phỏng một nhà tâm lý trị liệu bằng cách thực hiện các cuộc đối thoại với người dùng như sau:

| | |
|---|---|
| $User_1$: | Men are all alike. |
| $ELIZA_1$: | IN WHAT WAY |
| $User_2$: | They're always bugging us about something or other. |
| $ELIZA_2$: | CAN YOU THINK OF A SPECIFIC EXAMPLE |
| $User_3$: | Well, my boyfriend made me come here. |
| $ELIZA_3$: | YOUR BOYFRIEND MADE YOU COME HERE |
| $User_4$: | He says I'm depressed much of the time. |
| $ELIZA_4$: | I AM SORRY TO HEAR YOU ARE DEPRESSED |

ELIZA hoạt động bằng cách sử dụng các tầng thay thế biểu thức chính quy, mỗi tầng sẽ khớp và thay đổi một phần của các input. Nhóm thế đầu tiên thường sẽ là thay đổi cách xưng hô, như *my* thành *YOUR*, *I am, I'm* thành *YOU ARE*. Dưới đây là một số ví dụ thay thế khác:

```
s/.* I'M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

Trong một câu input có thể sẽ có nhiều chuỗi con thay thế, khi đó ta sử dụng capture group để sắp xếp thứ tự và tham chiếu các chuỗi con cần thay thế.

### 1.1.5 Lookahead

Toán tử **lookahead**, ký hiệu: `(?= pattern)` cho thêm vào để lọc kết quả của biểu thức chính quy, khi đó `/x(?=y)/` chỉ khớp $x$ nếu $x$ được theo sau bởi $y$. Phủ định lookahead, ký hiệu: `?! pattern` nghĩa là lấy kết quả mà đi sau nó không có chuỗi lookahead, khi đó `/x(?! y)/` chỉ khớp $x$ nếu $x$ không được theo sau bởi $y$.

Bên cạnh đó, ta cũng có khái niệm toán tử **lookbehind**, ký hiệu: `(?<= pattern)`, sử dụng để lấy đi các phù hợp mà đi trước là một mẫu cụ thể. Phủ định lookbehind, ký hiệu: `(?<! pattern)`, sử dụng để lấy các phù hợp mà đi trước không có mẫu lookbehind.

## 1.2 Chuẩn hóa văn bản

Với các bài toán xử lý ngôn ngữ tự nhiên, chuẩn hóa văn bản là một nhiệm vụ bắt buộc, một quy trình chuẩn hóa văn bản gồm 3 bước:

- Tách từ
- Chuẩn hóa từ (chuẩn hóa từ về dạng cơ sở, phân tích hình thái từ)
- Tách câu

Trong các phần tiếp theo chúng ta sẽ đi qua từng nhiệm vụ.

### 1.2.1 Tách từ tiếng Việt

Tách từ là một quá trình xử lý nhằm mục đích xác định ranh giới của các từ trong câu văn, cũng có thể hiểu đơn giản rằng tách từ là quá trình xác định các từ đơn, từ ghép,... có trong câu. Đối với xử lý ngôn ngữ tự nhiên, để có thể xác định cấu trúc ngữ pháp của câu, xác định từ loại của một từ trong câu, yêu cầu nhất thiết đặt ra là phải xác định được đâu là từ trong câu. Vấn đề này tưởng chừng đơn giản nhưng đối với máy tính, đây là bài toán rất khó giải quyết. Chính vì lý do đó, tách từ được xem là bước xử lý quan trọng đối với các hệ thống xử lý ngôn ngữ tự nhiên, đặc biệt là đối với các ngôn ngữ thuộc vùng Đông An theo loại hình ngôn ngữ đơn lập, một từ có thể có một hoặc nhiều âm tiết, ví dụ: tiếng Trung, Nhật, Thái và tiếng Việt. Với các ngôn ngữ thuộc loại hình này, ranh giới từ không chỉ đơn giản là khoảng trắng mà có sự liên hệ chặt chẽ giữa các tiếng với nhau, một từ có thể cấu tạo với một hoặc nhiều tiếng. Vì vậy với những ngôn ngữ như này, vấn đề của bài toán tách từ là khử được sự nhập nhằng trong ranh giới từ.

Tiếng Việt là ngôn ngữ không biến hình, từ điển từ tiếng Việt (Vietlex) có khoảng hơn 40000 từ. Trong đó, từ đơn là từ dùng 1 âm tiết làm một từ, từ ghép là tổ hợp các âm tiết, giữa các âm tiết đó có quan hệ về nghĩa với nhau, từ ghép được chia thành 2 loại:

- Từ ghép đẳng lập: các thành phần cấu tạo có quan hệ bình đẳng với nhau về nghĩa (*chợ búa, bếp núc,...*)
- Từ ghép chính phụ: các thành phần cấu tạo có quan hệ phụ thuộc nhau. Thành tố phụ có vai trò phân loại, chuyên biệt hóa và sắc thái hóa cho thành tố chính (*tàu hỏa, đường sắt,...*)

Ngoài ra còn có các từ diễn tả gồm nhiều từ (*bởi vì* cũng được coi là 1 từ), tên riêng gồm tên người và địa điểm cũng được coi là 1 đơn vị từ vựng, và các mẫu số, thời gian.

Một cách tổng quát có thể thấy rằng bài toán tách từ tiếng Việt có 3 phương pháp tiếp cận chính:

- Tiếp cận dựa vào từ điển cố định.
- Tiếp cận dựa vào thống kê thuần túy.
- Tiếp cận dựa trên cả hai phương pháp trên.

Trong đó, một số phương pháp thường được sử dụng như so khớp từ dài nhất (Longest Matching), so khớp cực đại (Maximum matching), mô hình Markov ẩn,.... Trong bài báo cáo này, tôi giới thiệu cách tiếp cận bài toán dựa vào từ điển bằng thuật toán so khớp từ dài nhất. Xây dựng từ điển bằng cách mỗi mục từ lưu thông tin về từ, từ loại, nghĩa loại, tổ chức sao cho tốn ít bộ nhớ và thuận tiện trong việc tìm kiếm. Từ loại và nghĩa loại kiểu byte được lưu dưới dạng một ký tự. Đầu vào của bài toán là từ điển và chuỗi đầu vào đã tách các dấu câu và âm tiết, tư tưởng thuật toán là duyệt từ trái sang phải hoặc từ phải sang trái, lấy các từ dài nhất có thể, dừng lại khi duyệt hết. Có thể thấy, tư tưởng của thuật toán này là thuật toán tham lam.

---

KHỞI TẠO

1. Cho chuỗi đầu vào $[w_0 w_1 \dots w_{n-1}]$
2. words $\leftarrow []$
3. $s \leftarrow 0$

LẶP

4. $e \leftarrow n$
5. Khi $[w_s \dots w_e]$ chưa là một từ: $e \leftarrow e - 1$
6. words $\leftarrow$ words + $[w_s \dots w_e]$
7. $s \leftarrow e + 1$
8. Nếu $e < n$: quay lại bước 4

KẾT THÚC

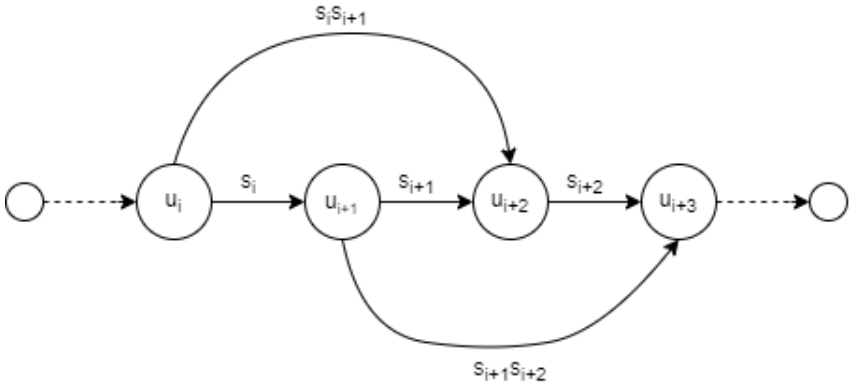
9. Lấy ra chuỗi đã tách từ words

---

Độ phức tạp thuật toán là $O(n, V)$ trong đó $n$ là số âm tiết trong chuỗi, $V$ là số từ trong từ điển. Ưu điểm của thuật toán là cài đặt đơn giản, độ phức tạp tính toán hợp lý, không yêu cầu tập dữ liệu huấn luyện. Nhược điểm là phụ thuộc vào từ điển, chưa giải quyết được vấn đề nhập nhằng trong tiếng Việt.

Một phương pháp tách từ khác là sử dụng biểu thức chính quy, một khuôn mẫu được so sánh với một chuỗi. Biểu thức chính quy thường được sử dụng đặc biệt nhiều trong phân tích cú pháp, xác nhận tính hợp lệ của dữ liệu, xử lý chuỗi, trích rút thông tin,... Đầu tiên, ta sẽ biểu diễn đoạn văn bản bằng chuỗi các âm tiết $s_1 s_2 \dots s_n$, trường hợp nhập nhằng thường xuyên nhất là 3 từ liền nhau $s_1 s_2 s_3$ trong đó $s_1 s_2$ và $s_2 s_3$ đều là một từ.

Biểu diễn biểu thức chính quy bằng một đoạn đồ thị có hướng tuyến tính $G = (V, E)$, trong đó $V = u_1, \dots, u_n, u_{n+1}$. Các âm tiết $s_i, \dots, s_j$ tạo thành một từ thì trong $G$ có cạnh $(u_i, u_j)$. Như vậy để giải quyết bài toán tách từ, ta cần tìm các đường đi ngắn nhất từ $u_0$ đến $u_{n+1}$.




Thuật toán xây dựng đồ thị cho chuỗi $s_1 s_2 \dots s_n$

---

1. $V \leftarrow \emptyset$
2. `for` $i = 0$ `to` $n + 1$ `do`
3. $V \leftarrow V \bigcup u_i$
4. `end for`
5. `for` $i = 0$ `to` $n$ `do`
6. `for` $j = i$ `to` $n$ `do`
7. `if (accept(`$A_w, s_i \dots s_j$`)) then // automat` $A_w$ `nhận xâu vào` $s_i \dots s_j$
8. $E \leftarrow E \bigcup (u_i, u_{j+1})$
9. `end if`
10. `end for`
11. `end for`
12. `return` $G = (V, E)$

---

Ngoài ra, hiện nay có một số công cụ tách từ như CRF, VnTokenizer, SVM, Pyvi,...

```
from  pyvi import ViTokenizer
str1="Nam phi công người Anh (BN91) được công bố khỏi bệnh: Đến 18h ngày 6/7, theo báo cáo của Tiểu " \
    "ban Điều trị Ban chỉ đạo Quốc gia phòng, chống dịch COVID-19, đã có thêm 1 bệnh nhân được công bố khỏi bệnh: " \
    "BN 91(43 tuổi, nam, phi công người Anh) tại Bệnh viện Chợ Rẫy. Căn cứ kết luận Hội chẩn Quốc gia ngày 3/7, " \
    "BN91 được chính thức công bố khỏi bệnh COVID-19, bệnh nhân có thể ra viện và không cần cách ly. " \
    "Tuy nhiên bệnh nhân còn tiếp tục ở lại Bệnh viện Chợ Rẫy điều trị phục hồi chức năng vận động để có thể hồi hương " \
    "trên chuyến bay thương mại ngày 12/7."

if __name__ == '__main__':
    print(ViTokenizer.tokenize(str1))

```

```
Nam phi_công người Anh ( BN91 ) được công_bố khỏi bệnh : Đến 18h ngày 6 / 7 , theo báo_cáo của Tiểu_ban Điều_trị Ban chỉ
_đạo Quốc_gia phòng , chống dịch COVID - 19 , đã có thêm 1 bệnh_nhân được công_bố khỏi bệnh : BN 91 ( 43 tuổi , nam , ph
i_công người Anh ) tại Bệnh_viện Chợ_Rẫy . Căn_cứ kết_luận Hội_chẩn Quốc_gia ngày 3 / 7 , BN91 được chính_thức công_bố k
hỏi bệnh COVID - 19 , bệnh_nhân có_thể ra viện và không cần cách_ly . Tuy_nhiên bệnh_nhân còn tiếp_tục ở lại Bệnh_viện C
hợ_Rẫy điều_trị phục_hồi chức_năng vận_động để có_thể hồi_hương trên chuyến bay thương_mại ngày 12 / 7 .
```

### 1.2.2 Phân tích hình thái từ

Hình thái từ (morphology) nghiên cứu cách từ được tạo ra từ các đơn vị nhỏ hơn gọi là hình vị (morpheme), hình vị là đơn vị ngôn ngữ nhỏ nhất mang ý nghĩa, ví dụ: *disadvantages = dis + advantage + s*. Có 2 lớp hình thái từ: hình thái học biến thể (inflectional morphology) và hình thái học dẫn xuất (derivational morphology). Với hình thái học biến thể, một từ được tạo nên từ gốc từ và hình vị ngữ pháp. Các từ sẽ cùng lớp với từ gốc và liên quan đến ngữ pháp của câu. Ví dụ: *he hits the ball* và *we hit the ball*, trong đó *hit* là từ gốc của *hits*. Ngoài ra cần phân biệt giữa dạng số nhiều và dạng sở hữu, ví dụ: *cats* và *cat's*. Với hình thái học biến thể, từ cũng được tạo nên từ gốc từ và hình vị ngữ pháp nhưng khác lớp, ví dụ *transmit* (Verb) và *transmission* (Noun).

**Vấn đề:** Ta cần xây dựng bộ phân tích hình thái từ như thế nào? Giải pháp đầu tiên là sử dụng từ điển, nhưng có một vấn đề giải pháp này không thực tế, một số ngôn ngữ kết nối các từ đơn với nhau để tạo ra từ có nghĩa. Đặc biết trong tiếng Trung, gần 3000 từ được dùng để ghép nối để tạo ra hàng chục nghìn từ mới. Giải pháp tiếp theo là phân tích từng hình vị từ, ví dụ: *misinterpretations = MIS + INTERPRET + noun form + plural*. Giải pháp này có nhược điểm là không thể tìm thấy tất cả các phần trong từ điển, ví dụ: *cities* $\neq$ *citie + s* $\neq$ *citi + es*. Để xử lý bài toán ta cần xác định rõ những vấn đề sau:

- Các hậu tố nào theo sau từ gốc và theo trật tự như nào: *cat/cats* - biến thể, *dog/dogged* - dẫn xuất.
- Mỗi hậu tố chỉ được kết hợp với một vài từ: *do + er* nhưng không có *be + er*
- Một số dạng từ đặc biệt thay đổi cấu tạo từ khi thêm hậu tố: *get + er* $\rightarrow$ 2 chữ *t* $\rightarrow$ *getter*.

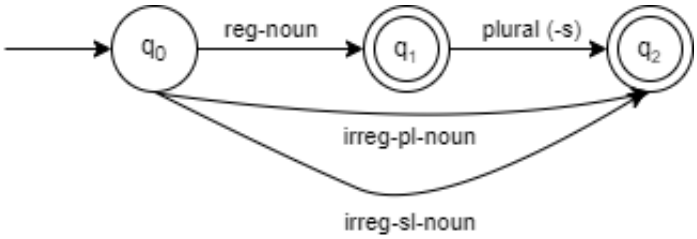
Một số định nghĩa ta cần lưu ý, **gốc từ** (stem) là đơn vị mang nghĩa gốc của từ (morpheme), **phụ tố** (affixes) là các đơn vị kết hợp với gốc từ để biến đổi ý nghĩa và chức năng ngữ pháp: tiền tố (prefix), hậu tố (suffix), trung tố (infix).

Tiếp theo, chúng ta giới thiệu cách sử dụng automata hữu hạn trạng thái (Finite-State Automata - FSA) để xây dựng bộ phân tích các hình vị từ. Một automata hữu hạn trạng thái (đơn định) là một bộ 5 $A = (A, \sum, \delta, q_0, F)$ trong đó:

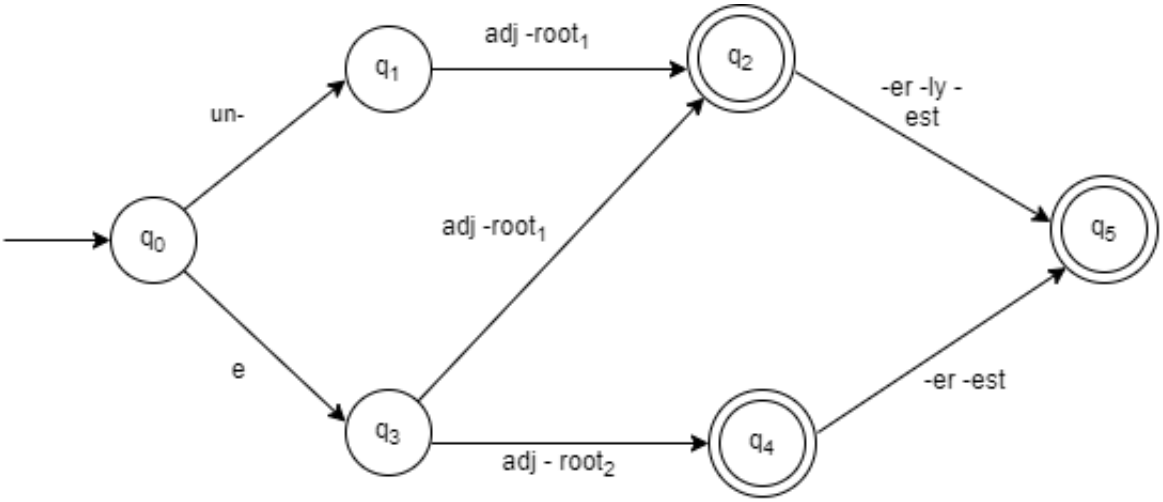
- $Q$: tập hữu hạn các trạng thái.
- $\sum$: tập hữu hạn bảng chữ cái, được gọi là bảng chữ vào (input alphabet).
- $\delta : Q \times \sum \rightarrow Q$: hàm chuyển trạng thái.
- $q_0 \in Q$: trạng thái bắt đầu.
- $F \subseteq Q$: tập các trạng thái kết thúc.

Các bước xây dựng bộ phân tích các hình vị từ dùng FSA:

1. Bước 1: Xây dựng FSA nhận dạng các dạng biến thể của từ.
2. Bước 2: Xây dựng FSA để nhận dạng các dẫn xuất của từ.
3. Bước 3: Sử dụng các FSA để nhận dạng từ.
4. Bước 4: Biến đổi FSA thành bộ chuyển đổi hữu hạn trạng thái để phân tích hình thái từ.
5. Bước 5: Bổ sung các quy luật biến đổi đặc biệt.




Hình 1.11: FSA cho các dạng biến thể của từ




Hình 1.12: đề xuất FSA cho các dạng dẫn xuất của từ

Với hình thái học dẫn xuất, việc xây dựng FSA sẽ phức tạp hơn hình thái học biến thể.

Câu hỏi đặt ra, ta cần xác định gốc từ như thế nào? Có một số luật như: hậu tố *SSES* $\rightarrow$ *SS (caresses* $\rightarrow$ *caress), IES* $\rightarrow$ *I (ponies* $\rightarrow$ *poni), S* $\rightarrow \emptyset$ *(cats* $\rightarrow$ *cat)*,... Các luật sử dụng phép đo 1 từ, kiểm tra số lượng âm tiết có đủ dài không sau khi cắt. Có 2 loại lấy gốc từ: *Stemmer* và *Lemmatizer*, stemmer sử dụng ít các luật ngôn ngữ hơn, lemmatizer cần bộ từ vựng đầy đủ và phân tích hình thái từ để lấy gốc từ. Cả stemmer

và lemmatizer đều không cải thiện kết quả tìm kiếm: tốt trong vài trường hợp, nhưng tệ trong nhiều trường hợp khác. Stemmer tăng độ phủ nhưng giảm độ chính xác. Để lấy gốc từ tốt, đó là vấn đề thực chứng (pragmatic) với mỗi dạng hơn là phân tích hình thái từ.

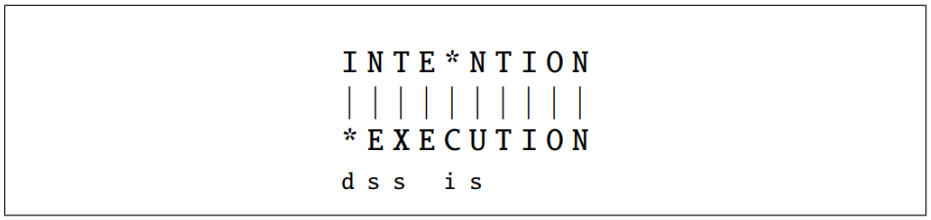
### 1.2.3 Tách câu

Tách câu là một bước quan trọng khác trong các bài toán xử lý văn bản. Dấu hiệu để phân đoạn một văn bản thành các câu thường là dấu chấm câu, dấu hỏi, dấu chấm than. Trong đó dấu hỏi và dấu chấm than tương đối rõ ràng trong việc phân tách ranh giới câu. Mặt khác, dấu chấm còn rất nhiều cách sử dụng khác ngoài việc kết thúc một câu, ví dụ như thường đi kèm các chữ viết tắt như *Mr.* hay *Inc.*, hoặc như dấu ba chấm (. . . ). Vì lý do này, việc tách câu và tách từ trong một đoạn văn bản thường được giải quyết cùng lúc.

## 1.3 Khoảng cách chỉnh sửa tối thiểu (Minimum Edit Distance)

Phần lớn các bài toán xử lý ngôn ngữ tự nhiên đều liên quan đến độ đo sự tương đồng giữa các chuỗi. Ví dụ, đối với bài toán sửa lỗi sai chính tả, người dùng gõ một chuỗi có lỗi sai, *let's say graffe*, và chúng ta cần phải xác định ý muốn của người dùng là gì. Nhận biết ý định người dùng chúng ta cần xác định được một từ tương đồng với *graffe* nhất.

Cho 2 chuỗi $A$ và $B$ có chiều dài lần lượt là $m$ và $n$. Khoảng cách chỉnh sửa (**edit distance**) là số lượng các thao tác (chèn, xóa, thay thế) cần thiết để biến chuỗi này thành chuỗi kia. Ví dụ, khoảng cách giữa *intention* và *execution* là 5: xóa $i$ thay $e$ bằng $n$, thay $x$ bằng $t$, chèn $c$, thay $u$ bằng $n$ (Hình 1.13)

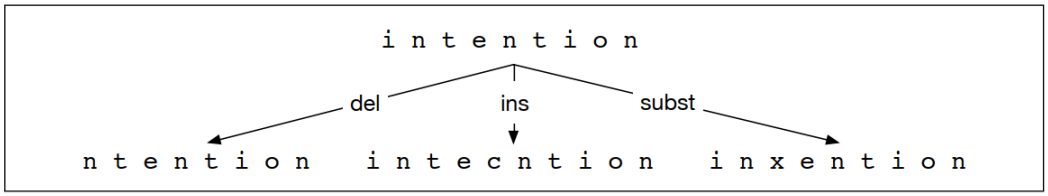



Hình 1.13: Biểu thị minimum edit distance giữa 2 chuỗi dưới dạng căn chỉnh, trong đó $d$ - xóa, $s$ - thay thế, $i$ - chèn

Khoảng cách **Levenshtein** thể hiện khoảng cách khác biệt giữa 2 chuỗi với việc đặt chi phí cho mỗi thao tác để biến chuỗi này thành chuỗi kia bằng 1. Do vậy, với 2 chuỗi *intention* và *execution*, khoảng cách Levenshtein bằng 5. Ngoài ra, cũng có một số phiên bản khác của khoảng cách Levenshtein như đặt chi phí cho thao tác thay thế

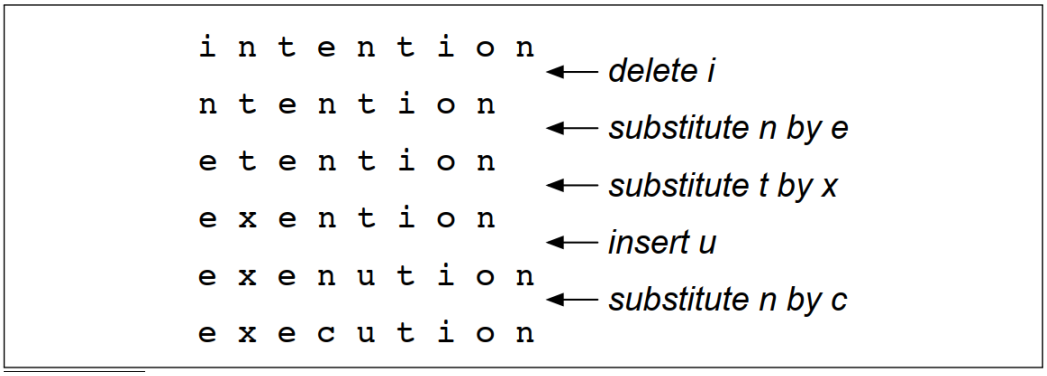
bằng 2, chèn và xóa bằng 1. Với phiên bản này, khoảng cách Levenshtein giữa 2 chuỗi *intention* và *execution* bằng 8.

### 1.3.1 Thuật toán minimun edit distance




Câu hỏi đặt ra là làm thế nào để xác định minimun edit distance? Chúng ta không thể xác định edit distance một cách ngây thơ. Tuy nhiên, nếu coi các thao tác chỉnh sửa là 1 đường dẫn từ chuỗi này đến chuỗi kia, chúng ta cần tìm đường đi ngắn nhất đến mỗi thao tác chỉnh sửa. Chúng ta có thể sử dụng quy hoạch động để xác định minimum edit distance.

Cấu trúc của giải thuật quy hoạch động là một quy trình gồm 3 bước: chia bài toán thành các bài toán con nhỏ hơn, giải các bài toán này một cách tối ưu bằng cách sử dụng đệ quy, và sử dụng các kết quả tối ưu đó xây dựn. Với bài toán này, xác định đường dẫn ngắn nhất của các từ được chỉnh sửa để biểu diễn minimum edit distance giữa 2 chuỗi *intention* và *execution* (hình 1.14)




Hình 1.14: Đường dẫn chỉnh sửa từ *intention* đến *execution*

Thuật toán minimum edit distance được đặt tên bởi *Wagner, Fischer (1974)* và đồng thời cũng được rất nhiều người khác phát hiện độc lập. Xét 2 chuỗi $X$ có độ dài $n$, $Y$ có độ dài $m$, chúng ta xác định $D(i,j)$ là khoảng cách chỉnh sửa giữa $X[i \dots i]$ và $Y[1 \dots j]$. Khoảng cách chỉnh sữa giữa $X$ và $Y$ là $D(n,m)$. Sử dụng quy hoạch động để xác định $D(n,m)$. Công thức đệ quy:

$$D[i,j] = \begin{cases} D[i-1,j] + del - cost(source[i]) \\ D[i,j-1] + ins - cost(source[j]) \\ D[i-1,j-1] + sub - cost(source[i], target[j]) \end{cases} \tag{1.1}$$

Với khoảng cách Levenshein, thao tác chèn và xóa có chi phí là 1 ($ins - cost(.) = del - cost(.) = 1$), thao tác thay thế có chi phí là 2, khi đó việc tính toán cho $D(i, j)$:

$$D[i,j] = \begin{cases} D[i-1,j] + 1 \\ D[i,j-1] + 1 \\ D[i-1,j-1] + \begin{cases} 2 & source[i] \neq target[j] \\ 0 & source[i] = target[j] \end{cases} \end{cases} \tag{1.2}$$

```
function MIN-EDIT-DISTANCE(source, target) returns min-distance

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix distance[n+1,m+1]

  # Initialization: the zeroth row and column is the distance from the empty string
      D[0,0] = 0
      for each row i from 1 to n do
          D[i,0] ← D[i-1,0] + del-cost(source[i])
      for each column j from 1 to m do
          D[0,j] ← D[0,j-1] + ins-cost(target[j])

  # Recurrence relation:
  for each row i from 1 to n do
      for each column j from 1 to m do
          D[i,j] ← MIN( D[i−1,j] + del-cost(source[i]),
                        D[i−1,j−1] + sub-cost(source[i], target[j]),
                        D[i,j−1] + ins-cost(target[j]))
  # Termination
  return D[n,m]
```

Hình 1.15: Giả mã quy hoạch động cho thuật toán minimum edit distance

| Src\Tar | # | e | x | e | c | u | t | i | o | n |
|---|---|---|---|---|---|---|---|---|---|---|
| # | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 6 | 7 | 8 |
| n | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 7 | 8 | 7 |
| t | 3 | 4 | 5 | 6 | 7 | 8 | 7 | 8 | 9 | 8 |
| e | 4 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 9 |
| n | 5 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 10 |
| t | 6 | 5 | 6 | 7 | 8 | 9 | 8 | 9 | 10 | 11 |
| i | 7 | 6 | 7 | 8 | 9 | 10 | 9 | 8 | 9 | 10 |
| o | 8 | 7 | 8 | 9 | 10 | 11 | 10 | 9 | 8 | 9 |
| n | 9 | 8 | 9 | 10 | 11 | 12 | 11 | 10 | 9 | 8 |

Hình 1.16: Tính toán minimum edit distance giữa 2 chuỗi *intention* đến *execution* bằng giả mã hình 1.15 sử dụng khoảng cách Levenshtein

Biết minimum edit distance được áp dụng hữu ích cho các bài toán như tìm lỗi sai chính tả. Trong nhận dạng giọng nói, nó được sử dụng để xác định tỷ lệ lỗi từ. Ngoài ra, nó còn đóng một vai trò quan trọng trọng dịch máy.

Để mở rộng thuật toán minimum edit distance, chúng ta có thể backtrace các đường dẫn chỉnh sửa (hình 1.17) bằng cách đặt:

$$ptr(i,j) = \begin{cases} LEFT \leftarrow & \text{chèn} \\ UP \uparrow & \text{xóa} \\ DIAG \nwarrow & \text{thay thế} \end{cases} \tag{1.3}$$

| | # | e | x | e | c | u | t | i | o | n |
|---|---|---|---|---|---|---|---|---|---|---|
| # | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| i | **1** | ↖←↑ 2 | ↖←↑ 3 | ↖←↑ 4 | ↖←↑ 5 | ↖←↑ 6 | ↖←↑ 7 | ↖ 6 | ← 7 | ← 8 |
| n | 2 | ↖←↑ **3** | ↖←↑ 4 | ↖←↑ 5 | ↖←↑ 6 | ↖←↑ 7 | ↖←↑ 8 | ↑ 7 | ↖←↑ 8 | ↖ 7 |
| t | 3 | ↖←↑ 4 | ↖←↑ **5** | ↖←↑ 6 | ↖←↑ 7 | ↖←↑ 8 | ↖ 7 | ←↑ 8 | ↖←↑ 9 | ↑ 8 |
| e | 4 | ↖ 3 | ← 4 | ↖← **5** | ← **6** | ← 7 | ←↑ 8 | ↖←↑ 9 | ↖←↑ 10 | ↑ 9 |
| n | 5 | ↑ 4 | ↖←↑ 5 | ↖←↑ 6 | ↖←↑ 7 | ↖←↑ **8** | ↖←↑ 9 | ↖←↑ 10 | ↖←↑ 11 | ↖↑ 10 |
| t | 6 | ↑ 5 | ↖←↑ 6 | ↖←↑ 7 | ↖←↑ 8 | ↖←↑ 9 | ↖ **8** | ← 9 | ← 10 | ←↑ 11 |
| i | 7 | ↑ 6 | ↖←↑ 7 | ↖←↑ 8 | ↖←↑ 9 | ↖←↑ 10 | ↑ 9 | ↖ **8** | ← 9 | ← 10 |
| o | 8 | ↑ 7 | ↖←↑ 8 | ↖←↑ 9 | ↖←↑ 10 | ↖←↑ 11 | ↑ 10 | ↑ 9 | ↖ **8** | ← 9 |
| n | 9 | ↑ 8 | ↖←↑ 9 | ↖←↑ 10 | ↖←↑ 11 | ↖←↑ 12 | ↑ 11 | ↑ 10 | ↑ 9 | ↖ **8** |

Hình 1.17: Sử dụng backtrace bắt đầu từ ô số 8 góc dưới bên phải và theo quy tắc đánh số mũi tên theo quy tắc trên. Chuỗi trong các ô thể hiện chi phí tối thiểu để căn chỉnh giữa 2 chuỗi

Độ phức tạp thuật toán:

- Time: $O(nm)$
- Space: $O(nm)$
- Backtrace: $O(n + m)$

# Chương 2

# Mô hình ngôn ngữ N-grams

Ngôn ngữ tự nhiên là những ngôn ngữ được con người sử dụng trong các giao tiếp hàng ngày: nghe, nói đọc, viết. Mặc dù con người có thể dễ dàng hiểu được và học các ngôn ngữ tự nhiên nhưng việc làm cho máy hiểu được ngôn ngữ tự nhiên không phải là chuyện dễ dàng. Sở dĩ có khó khăn là do ngôn ngữ tự nhiên có các bộ luật, cấu trúc ngữ pháp phong phú hơn nhiều các ngôn ngữ máy tính, hơn nữa để hiểu đúng nội dung các giao tiếp, văn bản trong ngôn ngữ tự nhiên cần phải nắm được ngữ cảnh của nội dung đó. Các phương pháp xử lý ngôn ngữ tự nhiên dựa trên thống kê không nhắm tới việc con người tự xây dựng mô hình ngữ pháp mà lập chương trình cho máy tính có thể "học" nhờ vào việc thống kê các từ và cụm từ có trong văn bản. Cốt lõi nhất của phương pháp xử lý ngôn ngữ tự nhiên dựa trên thống kê chính là việc xây dựng mô hình ngôn ngữ.

Mô hình ngôn ngữ là một phân bố xác suất trên các tập văn bản. Nói một cách đơn giản, mo hình ngôn ngữ có thể cho biết xác suất một câu (hoặc một cụm từ) thuộc một ngôn ngữ là có xác suất sinh là bao nhiêu. Ví dụ, khi áp dụng mô hình ngôn ngữ cho tiếng việt:

$P(\textit{hôm nay là thứ hai}) = 0.02$

$P(\textit{nay hai là hôm thứ}) = 0$

Mô hình ngôn ngữ được áp dụng trong rất nhiều lĩnh vực của xử lý ngôn ngữ tự nhiên như: sửa lỗi chính tả, dịch máy, tách từ,... Chính vì vậy, nghiên cứu mo hình ngôn ngữ chính là tiền đề nghiên cứu các lĩnh vực tiếp theo. Mô hình ngôn ngữ có nhiều hướng tiếp cân, nhưng chủ yếu được xây dựng theo mô hình N-grams.

## 2.1 N-grams

Mô hình ngôn ngữ thống kê cho phép gán (ước lượng) xác suất cho một chuỗi $n$ phần tử (thường là từ) $P(w_1 w_2 \dots w_n)$, cho phép dự đoán khả năng một chuỗi từ xuất hiện trong ngôn ngữ đó. Theo công thức Bayes:

$$P(AB) = P(B|A) * P(A)$$

Trong đó:

- $P(A)$ : Xác suất xảy ra sự kiện A
- $P(B)$ : Xác suất xảy ra sự kiện B
- $P(B|A)$ : Xác xuất (có điều kiện) xảy ra sự kiện $B$ nếu biết rằng sự kiện $A$ đã xảy ra

Với chuỗi $n$ từ $w_1, w_2, \ldots, w_n$, đặt là $w_1^n$. Khi đó:

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1}) \tag{2.1}$$

$$= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \tag{2.2}$$

Giả thuyết Markov: giả định rằng xác suất của một từ chỉ phụ thuộc vào một số từ trước đó. Mô hình Markov là mô hình ngẫu nhiên giả định thuộc tính Markov. Một mô hình ngẫu nhiên mô hình hóa quá trình trong đó trạng thái thụ thuộc vào các trạng thái trước đó. Một quy trình ngẫu nhiên có thuộc tính Markov nếu phân phối xác suất có điều kiện của các trạng thái tương lai trong quy trình. Khi đó N-gram là mô hình ngôn ngữ tính xác suất của từ dựa vào $N-1$ từ trước đó. Ví dụ, bigram - xét 1 từ trước đó, trigram - 2 từ trước đó,... Với $N$ càng lớn thì số trường hợp càng lớn nên thường người ta chỉ sử dụng với $N = 1, 2$, đôi lúc là 3. Phương trình tổng quán ước lượng N-gram của xác suất có điều kiện:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1}) \tag{2.3}$$

Áp dụng với mô hình Bigram, công thức xác suất có điều kiện được ước lượng:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1}) \tag{2.4}$$

Thay công thức 2.4 vào công thức 2.2, ta có:

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}) \tag{2.5}$$

Chúng ta ước lượng các xác suất N-gram bằng ước lượng Maximum Likelihood (ML) - ước lượng hợp lý cực đại. Ví dụ, để tính xác suất Bigram cụ thể cho từ $y$ và các từ cho trước $x$, ta sẽ lấy tổng số cụm từ $xy$ trong dữ liệu là $C(xy)$ và chia cho tổng tất cả các Bigram có chung từ cho trước $x$:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \tag{2.6}$$

Tương đương với:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{2.7}$$

Xét ví dụ sau: Cho 3 câu, trong đó ký hiệu `<s>` để bắt đầu một câu và, `</s>` để kết thúc một câu.

`<s> I am Sam </s>`
`<s> Sam I am </s>`
`<s> I do not like green eggs and ham </s>`

Khi đó, thực hiện một số tính toán:

$$P(I| < s >) = \frac{2}{3} = 0.67 \quad P(Sam| < s >) = \frac{1}{3} = 0.33 \quad P(am|I) = \frac{2}{3} = 0.67$$

$$P(< /s > |Sam) = \frac{1}{2} = 0.5 \quad P(Sam|am) = \frac{1}{2} = 0.5 \quad P(do|I) = \frac{1}{3} = 0.33$$

Công thức ước lượng ML cho xác suất N-gram:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n_N+1}^{n-1} w_n)}{C(w_{n_N+1}^{n-1})} \tag{2.8}$$

## 2.2 Đánh giá mô hình ngôn ngữ N-gram

### 2.2.1 Phân bố không đều

Xét một ví dụ với một văn bản, ta sử dụng dữ liệu từ Berkeley Restaurant Project, một hệ thống đối thoại trả lời các câu hỏi về database của các nhà hàng ở Berkeley, California (Jurafsky et al. 1994). Dưới đây là một số truy vấn của người dùng:

```
can you tell me about any good cantonese restaurants close by
mid priced thai food is what i'm looking for
tell me about chez panisse
can you give me a listing of the kinds of food that are available
i'm looking for a good place to eat breakfast
when is caffe venezia open during the day
```

Hình 2.1 cho thấy số lượng từ Bigram trong Berkeley Restaurant Project. Dễ thấy rằng, phân lớn các giá trị bằng 0, ma trận được sinh ra là một ma trận thưa

| | i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|---|
| **i** | 5 | 827 | 0 | 9 | 0 | 0 | 0 | 2 |
| **want** | 2 | 0 | 608 | 1 | 6 | 6 | 5 | 1 |
| **to** | 2 | 0 | 4 | 686 | 2 | 0 | 6 | 211 |
| **eat** | 0 | 0 | 2 | 0 | 16 | 2 | 42 | 0 |
| **chinese** | 1 | 0 | 0 | 0 | 0 | 82 | 1 | 0 |
| **food** | 15 | 0 | 15 | 0 | 1 | 4 | 0 | 0 |
| **lunch** | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **spend** | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

Hình 2.1: Số lượng các cụm Bigram của 8 từ

Có thể thấy, khi sử dụng mô hình N-gram, sự phân bố không đều trong tập văn bản huấn luyện có thể dẫn đến các ước lượng không chính xác. Khi các N-gram phân bố

thưa, nhiều cụm N-gram không xuất hiên hoặc chỉ có số lần xuất hiện nhỏ, việc ước lượng các câu có chứa cụm N-gram này sẽ có kết quả không tốt. Với $V$ là kích thước bộ từ vựng, ta sẽ có $V^N$ cụm N-gram có thể sinh từ bộ từ vựng này. Tuy hiện, thực tế thì số cụm N-gram có nghĩa và thường gặp chỉ chiếm rất ít.

Ví dụ, đối với tiếng Việt có khoảng hơn 5000 âm tiết khác nhau, ta có tổng số cụm 3-gram có thể có là $5000^3$, tuy nhiên, số cụm 3-gram được thống kê chỉ xấp xỉ 1.500.000.

Khi tính toán xác suất một câu, có rất nhiều trường hợp sẽ gặp cụm N-gram chưa xuất hiện trong dữ liệu huấn luyện bao giờ, điều này làm xác suất của cả câu bằng 0, trong khi câu đó có thể là một câu hoàn toàn đúng về mặt ngữ pháp và ngữ nghĩa.

### 2.2.2 Kích thước bộ nhớ của mô hình ngôn ngữ

Khi kích thước tập văn bản huấn luyện lớn, số lượng các cụm N-gram và kích thước của mô hình ngôn ngữ cũng rất lớn. Nó không những gây khó khăn trong việc lưu trữ mà còn làm tốc độ xử lý của mô hình ngôn ngữ giảm xuống do bộ nhớ của máy tính là hạn chế. Để xây dựng mô hình ngôn ngữ hiệu quả, chúng ta phải giảm kích thước của mô hình mà vẫn đảm bảo độ chính xác.

### 2.2.3 Độ hỗn độn (Perplexity)

Một trong những phương pháp phổ biến để đánh giá chất lượng của một mô hình ngôn ngữ là perplexity (độ hỗn độn), ký hiệu: $PP$. Giả sử ta có câu test: $W = w_1 w_2 \dots w_N$ có độ dài $N$, khi đó, chất lượng mô hình ngôn ngữ được tính như sau:

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \tag{2.9}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \tag{2.10}$$

Sử dụng công thức xác suất đồng thời, ta có:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \dots w_{i-1})}} \tag{2.11}$$

Với mô hình Bigram:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}} \tag{2.12}$$

Từ công thức 2.11, ta thấy khi xác suất càng cao, sự hỗn độn càng thấp, chất lượng của mô hình càng tốt. Vì vậy, giảm thiểu sự hỗn độn tương đương với việc tối đa hóa xác suất tập huấn luyện trong mô hình ngôn ngữ.

Tác giả huấn luyện các mô hình unigram, bigram và trigram trên 38 triệu từ (bao gồm các ký tự bắt đầu câu) từ Tạp chí Phố Wall, sử dụng 19.979 từ. Sau đó, tính toán mức độ hỗn độn của từng mô hình này trên bộ kiểm tra 1,5 triệu từ bằng công thức 2.12. Bảng dưới đây cho thấy sự hỗn độn của một tập huấn luyện:

| | **Unigram** | **Bigram** | **Trigram** |
|---|---|---|---|
| **Perplexity** | 962 | 170 | 109 |

Có thể thấy, mô hình Trigram cho ta đánh giá mô hình ngôn ngữ tốt hơn Bigram và Unigram.

## 2.3 Các phương pháp làm mịn

Để khắc phục tình trạng các cụm N-gram phân bố không đều, người ta đã đưa ra các phương pháp làm mịn các kết quả thống kê nhằm đánh giá chính xác hơn (mịn hơn) xác suất của các cụm N-gram. Các phương pháp làm mịn đánh giá lại xác suất của các cụm N-gram bằng cách:

- Gán cho các cụm N-gram có xác suất bằng 0 (không xuất hiện trong tập huấn luyện) một giá trị khác 0.
- Thay đổi lại giá trị xác suất của các cụm N-gram có xác suất khác 0 khác (có xuất hiện khi thống kê) thành một giá trị phù hợp (tổng xác suất của tất cả các khả năng N-gram khác nhau phải đảm bảo là không đổi, với giá trị là 100%)

Các phương pháp làm mịn có thể được chia ra thành một số loại như sau:

- Chiết khấu (Discouting): giảm (lượng nhỏ) xác suất của các cụm N-gram có xác suất lớn hơn 0 để bù cho các cụm N-gram không xuất hiện trong tập huấn luyện.
- Truy hồi (Back-off): Tính toán xác suất các cụm N-gram không xuất hiện trong tập huấn luyện dựa vào các cụm N-gram thành phần có độ dài ngắn hơn và có xác suất lớn hơn 0.
- Nội suy (Interpolation): Tính toán xác suất của tất cả các cụm N-gram dựa vào xác suất của các cụm N-gram ngắn hơn.

Trong phần này, ta sẽ giới thiệu một số phương pháp làm mịn cụ thể: add-1 smoothing (laplace), add-k smoothing, Stupid backoff, và Kneser-Ney smoothing.

### 2.3.1 Add-1 smoothing (Laplace smoothing)

Phương pháp này sẽ cộng thêm vào số lần xuất hiện của mỗi cụm N-gram lên 1, khi đó xác suất của cụm N-gram sẽ được tính lại là:

$$P = \frac{c+1}{N+V} \tag{2.13}$$

Trong đó $c$ là sô lần xuất hiện của cụm N-gram trong tập dữ liệu mẫu, $N$ là số cụm N-gram, $V$ là kích thước của toàn bộ từ vựng. Ở đây, $\sum c = N$, vì vậy, sau khi thêm 1 vào tần suất xuất hiện mỗi cụm N-gram, tổng này sẽ trở thành $\sum(c+1) = N+V$.

Với Unigram, ta có thể viết lại công thức 2.13 như sau:

$$P_1 = \frac{c_1+1}{N_1+V} \tag{2.14}$$

Ta có, $c^* = (c_1+1)\frac{N_1}{N_1+V}$ là tần suất của Bigram, trong đó $c_1$ là số lần xuất hiện của Unigram trước khi làm mịn bằng phương pháp Add-1.

Với cụm N-gram, $w = w_1w_2\dots w_n$, ta có:

$$\sum_w C(w_1w_2\dots w_{n-1}w) = C(w_1w_2\dots w_{n-1})$$

Do đó:

$$P(w_n|w_1w_2\dots w_{n-1}) = \frac{C(w_1w_2\dots w_n)+1}{C(w_1w_2\dots w_{n-1})+V}$$

### 2.3.2 Add-$k$ smoothing

Để ý thấy rằng, có rất nhiều cụm N-gram không nhìn thấy (bậc thấp) so với những N-gram nhìn thấy (bậc cao). Trong khi đó, có những cụm N-gram có nghĩa (cần thiết) bị giảm đi còn những cụm N-gram tối nghĩa lại có xác suất lớn. Để hạn chế điều này, người ta đưa thêm hệ số $k$ thay vì cộng 1 nhằm cân đối lại xác suất. Phương pháp này được gọi là làm mịn add-$k$. Công thức:

$$P = \frac{c+k}{V+kV} \tag{2.15}$$

Trong đó, $k < 1$, đặc biệt, khi $k = \frac{1}{2}$, được gọi là phương pháp Jeffreys - Perks.

Đặt $M = kV$, khi đó:

$$P(w_n|w_1w_2\dots w_{n-1}) = \frac{C(w_1w_2\dots w_n)+M(\frac{1}{V})}{C(w_1w_2\dots w_{n-1})+M} \tag{2.16}$$

Dễ thấy với một Unigram, tỷ số $\frac{1}{V}$ chính là xác suất xảy ra của mỗi Unigram, hay:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)+MP(w_i)}{C(w_{i-1}+M} \tag{2.17}$$

### 2.3.3 Good-Turing smoothing

Để tránh một cụm N-gram có nghĩa nhưng lại có tần suất xuất hiện nhỏ, người ta sử dụng phương pháp làm mịn Good-Turing dựa trên việc tính toán $N_c$, với $N_c$ là số cụm N-gram xuất hiện $c$ lần.

$$N_c = \sum_{w:count(w)} = c$$

Như vậy, $N_0$ là số cụm N-gram có tần số 0, $N_1$ là số cụm N-gram có tần số 1, ...

Khi đó, thuật toán Good-Turing sẽ thay thế tần số $c$ bằng một tần số $c^*$ theo công thức:

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

Xác suất của một cụm N-gram với tần số là $c$ được tính lại theo công thức sau:

$$P = (c+1)\frac{N_{c+1}}{N_c.N}$$

Trong đó $N$ là tổng số cụm N-gram.

### 2.3.4 Backoff và Interpolation

Trong các phương pháp Add-1 hoặc Add-$k$, nếu cụm $w_1 \dots w_{k-1}$ và cụm $w_1 \dots w_k$ đều không xuất hiện trong tập huấn luyện thì sau khi làm mịn xác suất vẫn bằng 0. Với phương pháp truy hồi (Backoff) ta sẽ giải quyết được vấn đề này.

Backoff là một phương pháp mà xác suất được tính từ điều kiện của các cụm N-gram trước đó. Nó được tính toán bằng cách "backing-off" với cụm N-gram ngắn hơn đã có trong những điều kiện nhất định. Bằng cách đó, backoff có xác suất tin cậy đã được tính để tạo ra các kết quả tốt hơn. Ngược lại, với phương pháp nội suy (Interpolation), ta kết hợp các ước lượng xác suất từ các mô hình N-gram khác nhau, cụ thể chúng ta kết hợp, tính trọng số của Trigram, Biram và Unigram.

Trong phép nội suy tuyến tính đơn giản, ta kết hợp các cụm N-gram khác nhau. Do đó, ta ước lượng xác suất Trigram $P(w_n|w_{n-2}w_{n-1})$ bằng cách kết hợp các xác suất Trigram, Bigram và Unigram, tương ứng với các trọng số $\lambda_1, \lambda_2, \lambda_3$:

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n) \tag{2.18}$$

Với điều kiện:

$$\sum_i \lambda_i = 1 \tag{2.19}$$

Với phép nội suy tuyến tính phức tạp hơn, mỗi trọng số $\lambda$ được tính dựa trên điều kiện của các bối cảnh. Nếu xác định số lượng chính xác số cụm Bigram, chúng ta giả định rằng số lượng cụm Trigram dựa trên số cụm Bigram sẽ là một số đáng tin cậy. Phương trình **??** là phương trình nội duy với các trọng số dựa trên điều kiện theo ngữ cảnh:

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) + \lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) + \lambda_3(w_{n-2}^{n-1})P(w_n) \tag{2.20}$$

Trong mô hình N-gram backoff, công thức xác suất $P_{BO}$ được tính như sau:

$$P_{BO}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}) & \text{nếu } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{BO}(w_n|w_{n-N+2}^{n-1}) & \text{otherwise} \end{cases} \tag{2.21}$$

Trong phương pháp Backoff, sự chính xác của mô hình phụ thuộc nhiều vào tham số $\alpha$. Có một vài cách để chọn $\alpha$ tùy theo tập huấn luyện và mô hình N-gram. Một cách đơn giản, ta có thể chọn $\alpha$ là một hằng số bất kỳ, tuy nhiên rất khó để chọn được một hằng số để tổng xác suất các cụm N-gram không đổi. Việc chọn hằng số không chính xác sẽ ảnh hưởng lớn đến độ chính xác của mô hình. Ta có thể chon $\alpha$ như một hàm của N-gram theo $(w_{k-1}w_k)$. Tuy nhiên, để mô hình có độ chính xác cao, người ta thường kết hợp thêm một số phương pháp như Good-Turing để làm giảm xác suất của các cụm N-gram đã xuất hiện.

### 2.3.5 Kneser-Ney Smoothing

Kneser-Ney là phương pháp chủ yếu được sử dụng để tính toán phân bố xác suất của N-gram trong một tài liệu dựa trên những tính toán đã có. Kneser-Ney được coi là phương pháp hiệu quả nhất để làm mịn do việc sử dụng các chiết khấu tuyệt đối bằng cách trừ đi một giá trị cố định từ điều kiện về tần số bậc thấp của xác suất để bỏ qua N-gram với tần số thấp hơn. Đây là phương pháp hiệu quả với Bigram và các mô hình N-gram cao hơn.

Ta xét một ví dụ, trong một Bigram "Bắc Kạn", từ "Kạn" chỉ đi sau từ "Bắc", trong khi đó, từ "Bắc" có số lần đếm nhiều hơn rất nhiều so với từ "Kạn"(có thể không nhìn thấy), xác suất của Bigram "Bắc Kạn" bằng 0. Vì thế, ý tưởng của phương pháp này là giảm đi số lần đếm được của những N-gram nhìn thấy (tất nhiên không thể giảm về 0), bù cho những N-gram không nhìn thấy với hệ số chiết khấu cố định $d$ (chiết khấu tuyệt đối - Absolute Discounting). Phương trình xác suất chiết khấu tuyệt đối nội suy áp dụng cho Bigram:

$$P_{AD}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{\sum_v C(w_{i-1}v)} + \lambda(w_{i-1})P(w_i) \tag{2.22}$$

Trong đó $\lambda$ là trọng số nội suy.

$P_{Continuation}(w)$ là xác suất khả năng từ $w$ xuất hiện tiếp theo dựa vào các từ cho trước. Các tính toán Kneser-Ney dựa trên xác suất này. Công thức $P_{Continuation}(w)$ với mô hình Bigram được cho bởi phương trình:

$$P_{Continuation} = \frac{|v : C(vw) > 0|}{\sum_{w'} |v : C(vw') > 0|} \tag{2.23}$$

Trong đó, $|v : C(vw) > 0|$ là số lượng các Bigram khác nhau kết thúc bởi $w$.

Cuối cùng, sử dụng nội suy để thực hiện các bước tính toán cuối cùng cho phương pháp làm mịn Kneser-Ney với mô hình Bigram:

$$P_{KN}(w_i|w_{i-1}) = \frac{\max\left(C(w_{i-1}w_i) - d, 0\right)}{C(w_{i-1})} + \lambda(w_{i-1})P_{Continuation}(w_i) \tag{2.24}$$

Trong đó, hằng số $\lambda$ được chuẩn hóa bằng công thức:

$$\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}w) > 0}|w : C(w_{i-1}w) > 0| \tag{2.25}$$

Công thức KN với trường hợp N-gram tổng quát:

$$P_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{\sum_v c_{KN}(w_{i-n+1}^{i-1}v)} + \lambda(w_{i-n+1}^{i-1})P_{KN}(w_i|w_{i-n+2}^{i-1}) \tag{2.26}$$

Trong đó $c_{KN}$ phụ thuộc vào bậc cụm N-gram cao nhất mà ta đã xét (ví dụ, là Trigram nếu chúng ta nội suy Trigram, Bigram, Unigram).

# Chương 3

# Sửa lỗi chính tả và kênh gây nhiễu

Sửa lỗi chính tả thường được xem xét từ 2 trường hợp. Sửa lỗi không từ (**Non-word spelling correction**) phát hiện và sữa lỗi chính tả mà đó không phải là một từ (Ví dụ *graffe* và *giraffe*). Ngược lại, sửa lỗi thực từ (**real word spelling correction**) là phát hiện và sửa lỗi chính tả mà tình cờ từ đó cũng là một từ trong tiếng Anh (**real-word errors**). Điều này có thể xảy ra do lỗi đánh máy và vô tình tạo ra một từ thực sự, hoặc do lỗi nhận thức (**cognitive errors**) khi người viết thay thế nhầm bằng một từ đồng âm hoặc gần đồng âm (Ví dụ *dessert* thay cho *desert*, *piece* thay cho *peace*).

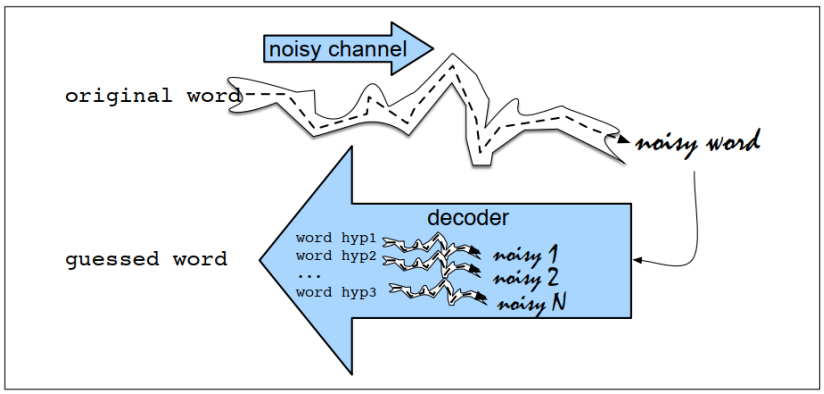
Sửa lỗi không từ được phát hiện bằng cách tìm tất cả các từ không có trong từ điển. Ví dụ từ bị sai chính tả *graffe* sẽ không được tìm thấy trong từ điển. Từ điển càng lớn càng tốt, các hệ thống hiện đại thường sử dụng những từ điển khổng lồ có nguồn gốc từ web. Để sửa lỗi không từ, đầu tiên ta tạo ra các **ứng viên (candidates)**: là những từ thực có một chuỗi chữ cái giống với từ lỗi. Các ứng viên cho lỗi chính tả *graffe* có thể là *giraffe*, *graf*, *gaffe*, *grail*, hay *craft*. Sau đó ta xếp hạng các ứng viên sử dụng một **khoảng cách metric** giữa từ gốc và từ lỗi. Ta muốn một metric có khoảng trực giác như chúng ta cho rằng *giraffe* giống với từ gốc hơn là *grail* khi xét từ gốc là *graffe*. Bởi vì *giraffe* có cách đánh vần giống với *graffe* hơn là *grail* so với *graffe*. Thuật toán cực tiểu hóa khoảng cách ở chương 2 sẽ đóng một vai trò ở đây. Nhưng ta muốn ưu tiên những cách sửa lỗi mà các từ xuất hiện thường xuyên hơn, hoặc có khả năng xuất hiện nhiều hơn trong ngữ cảnh của lỗi đó. Mô hình kênh nhiễu được giới thiệu trong phần tiếp theo cung cấp một cách để hình thức hóa trực giác này.

Phát hiện lỗi thực từ là một công việc khó hơn rất nhiều, do bất kỳ từ nào trong chuỗi đầu vào cũng có thể là một lỗi. Nhưng ta vẫn có thể sử dụng mô hình kênh nhiễu để tìm những ứng viên cho mỗi từ $w$ được gõ bởi người dùng, và xếp hạng cách sửa lỗi là từ mà khả năng lớn chính là ý định ban đầu của người dùng.

## 3.1 Mô hình kênh nhiễu (The Noisy Channel Model)

Trong phần này ta giới thiệu về mô hình kênh nhiễu và chỉ ra cách ứng dụng nó vào việc phát hiện và sửa lỗi chính tả. Mô hình kênh nhiễu được ứng dụng vào việc sửa lỗi

chính tả và khoảng thời gian mà những nhà nghiên cứu tại phòng thí nghiệm AT&T Bell và (Kernighan et al. 1990, Church and Gale 1991) và IBM Watson Research (Mays et al., 1991).




Hình 3.1: Trong mô hình kênh nhiễu, ta tưởng tượng rằng bề mặt mà chúng ta nhìn thấy thực ra là một dạng méo mó từ một từ gốc được truyền vào thông qua một kênh nhiễu. Bộ giải mã truyền từng giả thiết thông qua một mô hình của kênh này và lựa chọn từ phù hợp nhất với bề mặt từ nhiễu.

Về trực quan, model kênh nhiễu (Hình 3.1) xử lý từ bị sai chính tả như một từ đúng chính tả đã bị "làm méo" bằng việc được truyền qua một kênh nhiễu.

Kênh này tạo ra những sự nhiễu ở dạng thay thế hay những dạng biến đổi khác của các chữ cái, khiến cho nó khó bị nhận là một từ đúng. Mục đích của chúng ta là xây dựng một mô hình của kênh này. Khi đưa ra được mô hình này, chúng ta sẽ tìm được từ đúng bằng việc truyền tất cả các từ của ngôn ngữ đang xét qua mô hình kênh nhiễu của chúng ta và xem cái nào gần với từ bị sai chính tả nhất.

Mô hình kênh nhiễu này chính là một loại của suy luận Bayes (**Bayesian inference**). Ta xem xét những quan sát $x$ (từ bị sai chính tả) và việc của chúng ta là tìm ra từ $w$ đã tạo ra từ bị sai đó. Trong tất cả các từ có thể trong từ vựng $V$ ta cần tìm từ $w$ sao cho $P(w|x)$ là lớn nhất. Ta sử dụng ký hiệu $\hat{}$ với nghĩa "ước lượng của từ đúng".

$$\hat{w} = \arg\max_{w \in V} P(w|x) \tag{3.1}$$

Hàm $\arg\max_x f(X)$ có nghĩa "$x$ sao cho $f(x)$ đạt cực đại". Do đó, phương trình 3.1 có nghĩa là trong tất cả các từ, ta cần một từ cụ thể để cực đại vế phải $P(w|x)$.

Cốt lõi của phân loại Bayes là sử dụng quy tắc Bayes để biến đổi phương trình 3.1 thành một tập hợp các xác suất khác. Quy tắc Bayes được trình bày trong biểu thức 3.2 cho chúng ta một cách để tách xác suất có điều kiện bất kỳ $P(a|b)$ thành 3 xác suất khác:

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)} \tag{3.2}$$

Ta có thể thay thế biểu thức 3.2 vào 3.1 để được biểu thức 3.3:

$$\hat{w} = \arg\max_{w \in V} \frac{P(x|w)P(w)}{P(x)} \tag{3.3}$$

Ta có thể rút gọn biểu thức 3.3 bằng cách rút gọn mẫu số $P(x)$. Do ta đang chọn một từ chỉnh sửa có tiềm năng trong tất cả các từ, ta sẽ tính $\frac{P(x|w)P(w)}{P(x)}$ với mỗi từ. Nhưng $P(x)$ không đổi đối với mỗi từ; ta quan sát cùng một từ đang bị lỗi $x$, do đó có cùng xác suất $P(x)$. Vì thế, ta có thể chọn từ để cực đại hóa công thức đơn giản hơn như sau:

$$\hat{w} = \arg\max_{w \in V} P(x|w)P(w) \tag{3.4}$$

Tóm lại, mô hình kênh nhiễu nói rằng chúng ta có một số từ cơ bản $w$ là đúng, và chúng ta có một mô hình kênh nhiễu sửa đổi từ đó thành một số dạng sai chính tả có thể. **likelihood** hay **channel model** của kênh nhiễu tạo ra một chuỗi quan sát bất kỳ $x$ được cho bởi $P(x|w)$. Xác suất tiên nghiệm (**prior probability**) của một từ ẩn được cho bởi $P(w)$. Ta có thể tính toán từ có khả năng nhất $\hat{w}$ bằng cách nhân $P(w)$ với $P(x|w)$ và chọn từ nào có tích này là lớn nhất.

Ta áp dụng kênh hây nhiễu để tiếp cận việc sửa lỗi non–word bằng cách lấy một từ bất kỳ trong từ điển, tạo ra tất cả các từ ứng viên, xếp hạng chúng theo 3.4, và chọn ra từ có hạng cao nhất. Ta có thể biến đổi biểu thức 3.4 để chỉ xét danh sách các từ ứng viên thay vì toàn bộ các từ như sau:

$$\hat{w} = \arg\max_{w \in C} \overbrace{P(x|w)}^{\text{channel model}} \overbrace{P(w)}^{\text{prior}} \tag{3.5}$$

Thuật toán kênh gây nhiễu được trình bày ở hình 3.2.

**function** NOISY CHANNEL SPELLING(*word x*, *dict D*, lm, *editprob*) **returns** *correction*

**if** $x \notin D$
  candidates, edits ← All strings at edit distance 1 from *x* that are $\in D$, and their edit
  **for** each *c*, *e* in candidates, edits
    channel ← editprob(e)
    prior ← lm(x)
    *score[c]* = log *channel* + log *prior*
  **return** argmax$_c$ *score*[*c*]

Hình 3.2: Mô hình kênh nhiễu sửa lỗi chính tả

Để thấy cách tính toán của likelihood và prior, hãy xét một ví dụ. Áp dụng thuật toán cho ví dụ từ bị sai là *acress*. Bước đầu tiên của thuật toán để xuất những từ có

cách đánh vần tương đồng với từ input. Phân tích chỉ ra rằng phần lớn lỗi chính tả là sự sai khác một chữ cái, vì thế để đơn giản, ta giả sử rằng những từ ứng viên này cách từ bị lỗi 1 lần sửa đổi. Để tìm danh sách những từ ứng viên, ta sẽ sử dụng thuật toán **minimum edit distance** được giới thiệu ở chương 2, nhưng sẽ mở rộng thêm thao tác thứ 4 là đổi vị trí (transpositions), là phép toán mà 2 chữ cái sẽ đổi chỗ cho nhau, cùng với 3 thao tác là chèn (insertions), xóa (deletions), thay thế (substitutions). Phiên bản này được gọi là **Damerau-Levenshtein edit distance**. Áp dụng vào từ *acress* ta có được danh sách các từ ứng viên được cho ở hình 3.3

| | | Transformation | | | |
|---|---|---|---|---|---|
| **Error** | **Correction** | **Correct Letter** | **Error Letter** | **Position (Letter #)** | **Type** |
| acress | actress | t | — | 2 | deletion |
| acress | cress | — | a | 0 | insertion |
| acress | caress | ca | ac | 0 | transposition |
| acress | access | c | r | 2 | substitution |
| acress | across | o | e | 3 | substitution |
| acress | acres | — | s | 5 | insertion |
| acress | acres | — | s | 4 | insertion |

Hình 3.3: Những từ ứng viên cho lỗi chính tả *acress* và các phép biến đổi có thể gây ra lỗi. "—" mang ý nghĩa ký tự null

Khi ta có danh sách các từ ứng viên, ta dùng 3.5 để đánh giá từng ứng viên, ta cần tính xác suất tiên nghiệm (prior) và kênh nhiễu (channel model).

Xác suất tiên nghiệm $P(w)$ cho mỗi từ là xác suất của một từ $w$ xác suất **language model** của từ $w$ trong ngữ cảnh, được tính bằng cách sử dụng **language model**, từ unigram đến trigram hay 4-gram. Ví dụ unigram language model như trong bảng sau. Ta tính language model từ 404,253,213 từ trong Corpus of Contemporary English (COCA).

| **w** | **count(w)** | **p(w)** |
|---|---|---|
| actress | 9,321 | .0000231 |
| cress | 220 | .000000544 |
| caress | 686 | .00000170 |
| access | 37,038 | .0000916 |
| across | 120,844 | .000299 |
| acres | 12,874 | .0000318 |

Làm cách nào ta có thể tính $P(x|w)$? Một model hoàn hảo của xác suất một từ bị sai sẽ dựa vào tất cả các yếu tố: người đánh máy là ai, người đánh máy thuận tay trái hay tay phải, v..v... May mắn thay, ta có thể ước lượng một cách khá hợp lý $P(x|w)$ chỉ bằng cách xem xét ngữ cảnh hiện tại. Ví dụ, chữ cái $m$ và $n$ thường bị nhầm với nhau, đây là một phần của thực tế: 2 chữ cái $m$ và $n$ phát âm tương tự nhau và chúng nằm cạnh nhau trên bàn phím.

Một mô hình đơn giản, ví dụ $p(acress|across)$ chỉ cần dựa vào số lần chữ $e$ được thay thế với chữ $o$ trong một số lượng lỗi lớn. Trong trường hợp này ta sử dụng một **confusion matrix** chứa số lượng các lỗi. Tổng quát hơn, **confusion matrix** liệt kê ra số lần mà một thứ bị nhầm lẫn bởi một thứ khác. Ví dụ trong trường hợp trên sẽ là một ma trận kích thước $26 \times 26$ (hoặc tổng quát hơn là ma trận kích thước $|A| \times |A|$, với $A$ là một bảng chữ cái), ma trận này thể hiện số lần mà một chữ cái bị sử dụng sai thay vì một chữ cái khác. Ta sẽ dùng 4 **confusion matrix** là:

del[x, y]: count(xy được viết thành x)

ins[x, y]: count(x được viết thành xy)

sub[x, y]: count(x được viết thành y)

trans[x, y]: count(xy được viết thành yx)

Ta lấy những **confusion matrix** này ở đâu? Có một cách là trích xuất chúng từ danh sách các lỗi chính tả như sau:

**additional**: addional, additonal

**environments**: enviornments, enviorments, enviroments

**preceded**: preceeded

Khi ta đã có các **confusion matrix**, ta có thể ước lược được giá trị $P(x|w)$ như sau: ($w_i$ là ký tự thứ $i$ của từ đúng $w$) và $x_i$ là ký tự thứ $i$ của từ $x$:

$$P(x|w) = \begin{cases} \dfrac{del[x_{i-1}, w_i]}{count[x_{i-1w_i}]}, \text{if deletion} \\ \dfrac{ins[x_{i-1}, w_i]}{count[w_{i-1}]}, \text{if insertion} \\ \dfrac{sub[x_i, w_i]}{count[w_{i-1}]}, \text{if substitution} \\ \dfrac{trans[w_i, w_{i+1}]}{count[w_i w_{i+1}]}, \text{if transposition} \end{cases} \tag{3.6}$$

Sử dụng tính toán từ Kernighan et al. (1990), cho kết quả cho từ *acress* như hình 3.4

| Candidate Correction | Correct Letter | Error Letter | x\|w | P(x\|w) |
|---|---|---|---|---|
| actress | t | - | c\|ct | .000117 |
| cress | - | a | a\|# | .00000144 |
| caress | ca | ac | ac\|ca | .00000164 |
| access | c | r | r\|c | .000000209 |
| across | o | e | e\|o | .0000093 |
| acres | - | s | es\|e | .0000321 |
| acres | - | s | ss\|s | .0000342 |

Hình 3.4: Channel model cho *acress*, các xác suất được lấy từ các confusion matric *del[], ins[], subs[], trans[]* như trong Kernighan et al. (1990)

Hình 3.5 cho ta thấy những xác suất cuối cùng cho mỗi trường hợp. **unigram prior** được nhân bởi **likelihood** (Được tính bằng công thức 3.6 và các **confusion matrix**). Cột cuối cùng chính là kết quả, được nhân với $10^9$ cho dễ đọc.

| Candidate Correction | Correct Letter | Error Letter | x\|w | P(x\|w) | P(w) | $10^9$*P(x\|w)P(w) |
|---|---|---|---|---|---|---|
| actress | t | - | c\|ct | .000117 | .0000231 | 2.7 |
| cress | - | a | a\|# | .00000144 | .000000544 | 0.00078 |
| caress | ca | ac | ac\|ca | .00000164 | .00000170 | 0.0028 |
| access | c | r | r\|c | .000000209 | .0000916 | 0.019 |
| across | o | e | e\|o | .0000093 | .000299 | 2.8 |
| acres | - | s | es\|e | .0000321 | .0000318 | 1.0 |
| acres | - | s | ss\|s | .0000342 | .0000318 | 1.0 |

Hình 3.5: Xếp hạng các cách sửa lỗi ứng viên, sử dụng language model được nêu ở trên và error model từ hình 3.4. Điểm xếp hạng (score) cuối cùng được nhân với $10^9$ cho dễ đọc

Những kết quả trong hình 3.5 cho thấy việc triển khai mô hình kênh nhiễu chọn từ *across* là từ tốt nhất, và *actress* là từ tốt thứ 2.

Thật không may, thuật toán có vấn đề ở một chỗ: ý định của người viết khá rõ ràng nếu xét về ngữ cảnh: *. . . was called a "stellar and versatile acress whose combination of sass and glamour has defined her. . . "*. Ngữ cảnh này chỉ ra rằng từ *actress* mới là từ hợp lý hơn chứ không phải là *across*.

Vì lý do này, việc sử dụng một language model lớn hơn unigram là rất quan trọng. Ví dụ, nếu chúng ta sử dụng tiếng Anh Mỹ đương đại (Corpus of Contemporary American English) để tính các xác suất bigram cho từ *acress* và *across* trong ngữ cảnh của chúng sử dụng **add-one smoothing**, ta sẽ được các kết quả như sau:

$$P(actressjversatile) = .000021$$

$$P(across jversatile) = .000021$$

$$P(whose jactress) = .0010$$

$$P(whose jacross) = .000006$$

Nhân các kết quả này với nhau ta được language model ước lượng cho 2 ứng viên:

$$P(\text{"}versatileactresswhose\text{"}) = .000021 * .0010 = 210 \times 10^{-10}$$

$$P(\text{"}versatileacrosswhose\text{"}) = .000021 * .000006 = 1 \times 10^{-10}$$

Kết hợp với language model được cho bởi hình 3.5, mô hình kênh nhiễu bigram chọn từ để sửa lỗi là *actress*.

Việc đánh giá các thuật toán sửa lỗi chính tả thường được thực hiện bằng cách tổ chức một tập luyện, phát triển và kiểm tra từ những danh sách các lỗi.

## 3.2 Lỗi chính tả thực từ (Real-word spelling errors)

Cách tiếp cận kênh nhiễu cũng có thể được áp dụng để phát hiện và sửa lỗi real–word. Hãy bắt đầu với một phiên bản của mô hình kênh nhiễu được giới thiệu bởi Mays et al. (1991). Thuật toán của họ là lấy câu input $X = \{x_1, x_2, \ldots, x_k, \ldots, x_n\}$, tạo ra một tập gồm các câu có khả năng là đúng $C(X)$, sau đó chọn ra câu có xác suất language model cao nhất.

Để tìm $C(X)$, ta bắt đầu bằng việc tạo ra một tập các từ ứng viên với mỗi từ input $x_i$. Các $C(x_i)$ bao gồm tất cả các từ tiếng anh với **edit distance** nhỏ so với $x_i$. Với **edit distance** là 1 và từ bị lỗi là *thew*, tập các từ ứng viên có thể là $C(thew) = \{the, thaw, threw, them, thwe\}$. Ta giả sử rằng mỗi câu chỉ có một lỗi chính tả, khi đó $C(X)$ với $X =$ `Only two of thew apples` có thể là:

```
only two of thew apples
oily two of thew apples
only too of thew apples
only to of thew apples
only tao of the apples
only two on thew apples
only two off thew apples
only two of the apples
only two of threw apples
only two of thew applies
only two of thew dapples
...
```

Mỗi câu trên được xếp hạng bởi kênh nhiễu:

$$\hat{W} = \arg\max_{W \in C(X)} P(X|W)P(W) \tag{3.7}$$

Với $P(W)$, ta có thể sử dụng xác suất trigram.

Vậy còn mô hình kênh gây nhiễu? Do đây đều là những từ thật sự, ta cần phải xét khả năng mà từ đầu vào không phải là một lỗi. Gọi xác suất mà để một từ được viết đúng, $P(w|w)$ là $\alpha$. Mays et al. (1991) đưa ra một model đơn giản: cho một từ $x$, $P(x|w) = \alpha$ khi $x = w$, và chia đều $1 - \alpha$ cho các phần tử khác trong $C(x)$:

$$P(x|w) = \begin{cases} \alpha & \text{if } x = w \\ \dfrac{1-\alpha}{|C(x)|} & \text{if } x \in C(x) \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

Xét một ví dụ về mô hình này. Giả sử ta có chuỗi `two of thew`. Người viết có thể thực sự muốn viết từ `thew`. Nhưng `thew` ở đây có thể là `the` hay một từ nào đó. Trong ví dụ này, xét **edit distance** là 1, và chỉ có 5 từ ứng viên là `the`, `thaw`, `threw`, `thwe` và từ được viết `thew`. Ta lấy **edit probabilities** từ Norvig's (2009) cho ví dụ này. Với **language model probabilities**, ta sử dụng Stupid Backoff model (Section 4.6)

$$\begin{aligned} P(\text{the}|\text{two of}) &= 0.476012 \\ P(\text{thew}|\text{two of}) &= 9.95051 \times 10^{-8} \\ P(\text{thaw}|\text{two of}) &= 2.09267 \times 10^{-7} \\ P(\text{threw}|\text{two of}) &= 8.9064 \times 10^{-7} \\ P(\text{them}|\text{two of}) &= 0.00144488 \\ P(\text{thwe}|\text{two of}) &= 5.18681 \times 10^{-9} \end{aligned}$$

Ở đây ta chỉ áp dụng cho một cụm đơn giản *two of thew*, nhưng model này có thể áp dụng cho tất cả các câu, vì thế nếu ví dụ là *two of thew people*, ta sẽ cần phải nhân với các xác suất $P(\text{people}|\text{of the})$, $P(\text{people}|\text{of thew})$, $P(\text{people}|\text{of threw})$,...

Theo Norvig (2009), ta giả sử rằng xác suất để một từ là lỗi đánh máy là 0.05, tức là $\alpha = P(w|w) = 0.95$. Hình 3.6 cho ta kết quả.

| x | w | x\|w | $P(x\|w)$ | $P(w\|w_{i-2}, w_{i-1})$ | $10^8 P(x\|w)P(w\|w_{i-2}, w_{i-1})$ |
|---|---|---|---|---|---|
| thew | the | ew\|e | 0.000007 | 0.48 | 333 |
| thew | thew | | $\alpha$=0.95 | $9.95 \times 10^{-8}$ | 9.45 |
| thew | thaw | e\|a | 0.001 | $2.1 \times 10^{-7}$ | 0.0209 |
| thew | threw | h\|hr | 0.000008 | $8.9 \times 10^{-7}$ | 0.000713 |
| thew | thwe | ew\|we | 0.000003 | $5.2 \times 10^{-9}$ | 0.00000156 |

Hình 3.6: Mô hình kênh nhiễu trên 5 khả năng của `thew`, với Stupid Backoff trigram language model được tính từ Google N-gram và error model từ Norvig (2009).

Với cụm từ lỗi *two of thew*, mô hình sửa lỗi chọn từ để sửa là *the*. Nhưng lưu ý rằng nếu tỷ lệ lỗi thấp hơn, kết quả có thể sẽ thay đổi. Khi xác suất một từ là lỗi đủ thấp ($\alpha$ đủ cao), mô hình có thể chọn từ *thew* chính là từ đúng.

## 3.3 Mô hình kênh nhiễu

Một số mở rộng về những mô hình đơn giản được trình bày ở trên.

Đầu tiên, thay vì giả thiết chuỗi đầu vào chỉ có một lỗi duy nhất, các hệ thống hiện đại duyệt qua từng từ một và sử dụng kênh nhiễu để đưa ra quyết định cho từ đó. Nhưng nếu chỉ đơn thuần chạy hệ thống kênh nhiễu cơ bản như trên, nó sẽ dễ dàng bị quá tải, thay thế các từ đúng nhưng ít được sử dụng bằng các từ thường được sử dụng hơn. Do đó, các thuật toán hiện đại cần tăng cường kênh nhiễu để phát hiện xem một từ có nên sửa hay không. Ví dụ hệ thống của Google (Whitelaw et al., 2009) sử dụng một danh sách đen, ngăn một số loại ký tự (ví dụ số, chấm câu, các từ chỉ có 1 chữ cái) khỏi việc bị sửa đổi. Những hệ thống như thế cũng thận trong hơn trong việc tin tưởng các cách sửa lỗi. Thay vì việc chỉ chọn cách sửa có $P(w|x)$ cao hơn chính nó, những hệ thống này cẩn thận hơn, chỉ sửa từ $w$ khi sự khác biệt về xác suất là đủ lớn. Cách sửa $w$ chỉ được chọn khi và chỉ khi:

$$\log P(w|x) - \log P(x|x) > \theta$$

Dựa vào sự khác biệt một cách sửa lỗi có đủ tốt hay không, trong các ứng dụng cụ thể, có thể có tính năng tự động sửa lỗi hoặc chỉ đưa ra một số gợi ý. (Các thuật toán về vấn đề này được giới thiệu ở chương sau).

Các hệ thống hiện đại sử dụng những bộ từ điển lớn hơn rất nhiều so với các hệ thống ra đời sớm hơn.

Ta có thể cải tiến mô hình kênh nhiễu bằng cách thay đổi cách kết hợp giữa **prior** và **likelihood**. Trong mô hình cũ, chúng chỉ nhân lại với nhau. Nhưng thường thì những xác suất này không có một sự tương xứng. Ta sử dụng một cách kết hợp tốt hơn bằng cách lũy thừa một yếu tố với $\lambda$

$$\hat{w} = \arg\max_{v \in V} P(x|w)P(w)^{\lambda} \tag{3.9}$$

hay

$$\hat{w} = \arg\max_{w \in W} \log P(x|w) + \lambda \log P(w) \tag{3.10}$$

Sau đó ta tìm cách để điều chỉnh $\lambda$

Cuối cùng, nếu mục đích của chúng ta là chỉ sửa lỗi cho những **confusion sets** cụ thể như *peace/piece, affect/effect, weather/whether,* hay thậm chí là sửa lỗi ngữ pháp như *among/between*, ta có thể phân lớp bằng luyện có giám sát để rút ra những ngữ cảnh và đưa ra lựa chọn giữa 2 từ ứng viên.

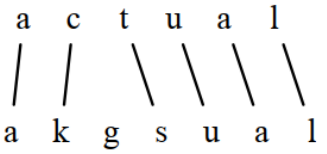
### 3.3.1 Cải tiến Edit Models: Partitions and Pronunciation

Các nghiên cứu gần đây chủ yếu tập trung vào việc cải tiến $P(t|c)$. Có một mở rộng quan trọng là khả năng tính xác suất cho phép biến đổi nhiều chữ cái. Ví dụ Brill and Moore (2000) đề xuất một channel model (không chính thức) mô hình một lỗi được tạo ra bởi đánh máy, bằng việc đầu tiên chọn một từ, sau đó chọn một cách chia từ đó thành các phân vùng gồm các chữ cái, sau đó gõ từng phân vùng một và có thể gõ sai những phân vùng đó. Ví dụ một người đang gõ chữ `physical`, chọn cách chia phân vùng `ph y s i c al`. Gõ từng phân vùng một. Ví dụ xác suất để người đó gõ thành `fisikle` với cách chia `f i s i k le` sẽ là $p(\texttt{f}|\texttt{ph}) * p(\texttt{i}|\texttt{y}) * p(\texttt{s}|\texttt{s}) * p(\texttt{i}|\texttt{i}) * p(\texttt{k}|\texttt{k}) * p(\texttt{le}|\texttt{al})$. Hơn nữa, mỗi cách sửa còn tùy theo vị trí của từ đó (beginning, middle, end) vì thế thay vì $P(\texttt{f}|\texttt{ph})$, mô hình ước lượng $P(\texttt{f}|\texttt{ph, beginning})$.

Hình thức hóa, cho $R$ là một cách phân vùng của chuỗi $x$ thành cách chuỗi con liền kề nhau (có thể rỗng), và $T$ là một cách phân vùng của chuỗi ứng viên. Brill and Moore xấp xỉ $P(x|w)$ với xác suất tương ứng với phân vùng tốt nhất:

$$P(x|w) \approx \max_{R,T,s.t.|T|=|R|} \sum_{i=1}^{|R|} P(T_i|R_i, position) \tag{3.11}$$

Xác suất của mỗi phép biến đổi $P(T_i|R_i)$ có thể được học bởi tập luyện với bộ ba: lỗi, chuỗi đúng, số lần lỗi xảy ra. Ví dụ cho một cặp chuỗi training `akgsual/actual`, tạo ra căn chỉnh:




Căn chỉnh này tương ứng với chuỗi các bước chỉnh sửa:

`a` → `a`, `c` → `k`, $\epsilon$ → `g`, `t` → `s`, `u` → `u`, `a` → `a`, `l` → `l`

Với mỗi sự thay thế không phù hợp sẽ được mở rộng thêm $N$ chỉnh sửa bổ sung. Ví dụ $N = 2$, ta có thể mở rộng `c` *rightarrow*`k` thành:

`ac` → `ak` `c` → `cg` `ac` → `akg` `ct` → `kgs`

Xác suất cho mỗi chỉnh sửa $\alpha \rightarrow \beta$ được ước lượng từ số lượng (count) trong tập train là $\dfrac{count(\alpha \rightarrow \beta)}{count(\alpha)}$

Một nghiên cứu khác là sử dụng phát âm (**pronunciation**) bên cạnh chính tả. Phát âm là một phần quan trọng trong các thuật toán kênh không nhiễu (non-noisy-channel) như thuật toán GNU aspell, lợi dụng các quy tắc metaphone. Metaphone là một chuỗi các quy tắc ánh xạ một từ thành một đại diện cho cách phát âm của nó. Một số ví dụ:

- Bỏ các chữ cái liền kề và trùng lặp, ngoại trừ ký tự C

- Nếu từ được bắt đầu bởi 'KN', 'GN', 'PN', 'AE', 'WR', bỏ chữ cái đầu tiên.
- Bỏ chữ B nếu nó sau chữ M và nếu nó ở cuối từ.

Phát âm cũng có thể kết hợp trực tiếp với mô hình kênh nhiễu. Ví dụ Toutanova and Moore (2002) model, thêm vào 2 mô hình kênh, một dựa vào chính tả và một dựa vào phát âm. Mô hình phát âm dựa trên mô hình **letter-to-sound** để dịch một từ đầu vào hoặc một từ trong từ điển thành một chuỗi đại diện cho cách phát âm của từ đó. Ví dụ `actress` và `aktress` đều có thể ánh xạ thành chuỗi `ae k t r ix s`.

MỘt số hàm khoảng cách được để xuất để xử lý một số vấn đề về tên. Chúng chủ yếu được sử dụng cho nhiệm vụ trùng lặp (quyết định xem 2 tên trong một danh sách nào đó có giống nhau hay không).

Thuật toán Soundex là một phương pháp lâu đời hơn. Nó có lợi thế là các tên hơi sai chính tả được thể hiện bằng những tên đúng chính tả (ví dụ Jurafsky, Jarofsky, Jarovsky, và Jarovski đều được ánh xạ thành J612). Thuật toán được trình bày ở hình 3.7

**function** SOUNDEX(*name*) **returns** *soundex form*

1. Keep the first letter of *name*
2. Drop all occurrences of non-initial a, e, h, i, o, u, w, y.
3. Replace the remaining letters with the following numbers:
   b, f, p, v → 1
   c, g, j, k, q, s, x, z → 2
   d, t → 3
   l → 4
   m, n → 5
   r → 6
4. Replace any sequences of identical numbers, only if they derive from two or more letters that were *adjacent* in the original name, with a single number (e.g., 666 → 6).
5. Convert to the form `Letter Digit Digit Digit` by dropping digits past the third (if necessary) or padding with trailing zeros (if necessary).

Hình 3.7: Thuật toán Soundex

Hoặc thay vì Soundex, các công trình gần đây sử dụng khoảng cách Jaro-Winkler (Winkler, 2006)

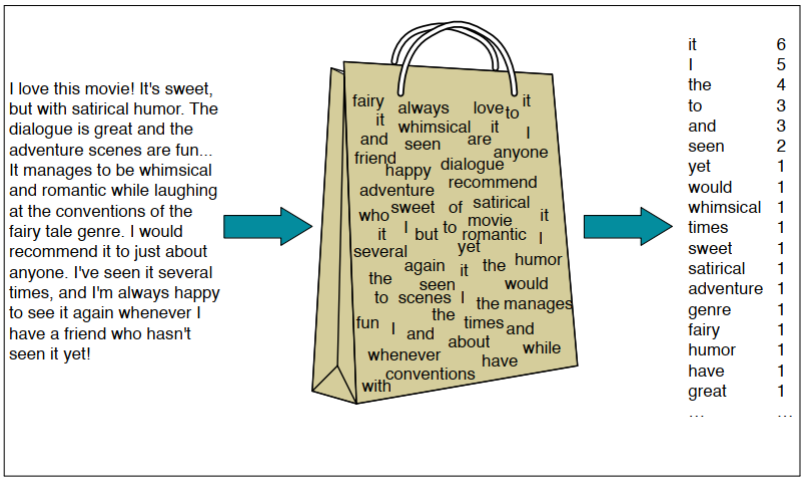
# Chương 4

# Naive Bayes và bài toán phân loại cảm xúc

## 4.1 Thuật toán phân loại Naive Bayes

Trong phần này ta sẽ giới thiệu thuật toán phân loại đa thức Bayes. Sở dĩ gọi là "Naive" vì thuật toán này hoạt động dựa trên các giả thiết làm đơn giản hoá các đặc trưng và sự tương tác giữa chúng.

Dữ liệu đầu vào của thuật toán được thể hiện trong hình 6.1. Ta coi văn bản như là một bag-of-words, tức là, một tập không sắp thứ tự của các từ, không quan tâm đến vị trí của chúng mà chỉ quan tâm đến tần suất xuất hiện của chúng trong văn bản. Trong ví dụ, thay vì xét các từ và thứ tự của chúng trong câu như "I like this movie" và "I would recommend it", ta chỉ đơn giản xét từ "I" xuất hiện 5 lần trong toàn bộ văn bản, từ "it" xuất hiện 6 lần, các từ "love", "recommend", và "movie" xuất hiện 1 lần và tiếp tục.




Hình 4.1: Bag-of-word

Naive Bayes là một thuật toán phân loại xác suất, tức là đối với mỗi văn bản $d$,

trong tất cả các lớp $c \in C$ thuật toán phân loại trả về lớp $\hat{c}$ có xác suất hậu nghiệm là cao nhất đối với văn bản đã cho. Trong phương trình 4.1 ta dùng ký hiệu mũ ˆ cho ước lượng của lớp chính xác

$$\hat{c} = \arg\max_{c \in C} P(c|d) \tag{4.1}$$

Ý tưởng của thuật toán phân loại Bayes là sử dụng công thức xác suất đầy đủ để đưa phương trình 4.1 về các xác suất khác có các tính chất hữu dụng. Công thức xác suất đầy đủ Bayes được thể hiện trong phương trình 4.2:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{4.2}$$

Thế phương trình 4.2 vào 4.1 ta được 4.3

$$\hat{c} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(x|d)P(c)}{P(d)} \tag{4.3}$$

Ta có thể loại bỏ $P(d)$ ra khỏi phương trình 4.3. Ta làm được điều này vì ta sẽ tính $\frac{P(d|c)P(c)}{P(d)}$ cho từng lớp. Nhưng $P(d)$ lại không thay đổi đối với mỗi lớp, và phải có cùng một xác suất $P(d)$. Vì vậy, ta có thể chọn lớp làm cực đại công thức đơn giản hơn

$$\hat{c} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} P(d|c)P(c) \tag{4.4}$$

Ta sau đó đi tìm lớp có xác suất cao nhất đối với văn bản $d$ bằng việc chọn lớp có tích cao nhất giữa hai xác suất: Xác suất tiên nghiệm của lớp $P(c)$ (prior) và xác suất $P(d|c)$ (likelihood):

$$\hat{c} = \arg\max_{c \in C} = \overbrace{P(d|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}} \tag{4.5}$$

Không làm mất tính tổng quát, ta có thể đại diện văn bản $d$ dưới dạng một tập các đặc trưng $f_1, f_2, ..., f_n$:

$$\hat{c} = \arg\max_{c \in C} = \overbrace{P(f_1, f_2, ..., f_n|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}} \tag{4.6}$$

Không may, phương trình 4.6 vẫn quá phức tạp để có thể tính trực tiếp: nếu không có các giả thiết đơn giản hoá, ước lượng xác suất của tất cả các tổ hợp các đặc trưng (ví dụ như tất cả các tập hợp từ và vị trí của chúng) sẽ yêu cầu khối lượng tham số rất lớn và một tập luyện khổng lồ. Thuật toán phân loại Bayes do đó đưa ra 2 giả thiết.

Đầu tiên là giả thiết bag-of-word đã đề cập ở trên: Ta giả sử vị trí các từ không quan trọng, và từ "love" có cùng ảnh hưởng đến quá trình phân loại dù nó có đứng

đầu tiên hay thứ 20 hay ở cuối văn bản. Vì vậy mà ta giả sử các đặc trưng $f_1, f_2, ..., f_n$ chỉ chứa các từ mà không chứa vị trí của chúng.

Giả thiết thứ hai thường được gọi là giả thiết naive Bayes: Đây là điều kiện các xác suất $P(f_i|c)$ là độc lập xác suất với lớp $c$ cho trước và khi đó có thể viết thành:

$$P(f_1, f_2, ..., f_n) = P(f_1|c).P(f_2|c)....P(f_n|c) \tag{4.7}$$

Phương trình cuối cùng cho lớp mà thuật toán phân loại naive Bayes trả về là:

$$c_{NB} = \arg\max_{c \in C} P(c) \prod_{f \in F} P(f|c) \tag{4.8}$$

Để áp dụng thuật toán phân loại naive Bayes cho văn bản, ta phải xét vị trí của các từ, bằng việc đánh index cho mọi vị trí các từ trong văn bản:

$$\text{index} \longleftarrow \text{tất cả các vị trí từ trong văn bản}$$

$$c_{NB} = \arg\max_{i \in C} P(c) \prod_{i \in index} P(w_i|c) \tag{4.9}$$

Tính toán naive Bayes được thực hiện trên không gian log để tránh tràn số và tăng tốc độ. Vì vậy phương trình 4.9 thường được viết dưới dạng

$$c_{NB} = \arg\max_{c \in C} \log P(c) + \sum_{i \in index} \log P(w_i|c) \tag{4.10}$$

Bằng việc xét các đặc trưng trên không gian log, phương trình 4.10 tính lớp dự đoán như một hàm tuyến tính của các đặc trưng đầu vào. Các thuật toán phân loại sử dụng tổ hợp tuyến tính của các input để đưa ra quyết định phân loại - như phân loại Bayes và hồi quy tuyến tính - được gọi là các thuật toán phân loại tuyến tính.

## 4.2 Huấn luyện cho thuật toán phân loại naive Bayes

Ta cần phải tính các xác suất $P(c)$ và $P(f_i|c)$. Xét ước lượng hợp lý cực đại, ta sẽ sử dụng các tần số trong dữ liệu. Đối với P(c) ta tính phần trăm số các văn bản trong tập luyện thuộc về lớp $c$. Gọi $N_c$ là số văn bản trong tập luyện thuộc về lớp $c$ và $N_{doc}$ là tổng số toàn bộ văn bản. Khi đó:

$$\hat{P}(c) = \frac{N_c}{N_{doc}} \tag{4.11}$$

Để tính các xác suất $P(f_i|c)$, ta sẽ giả sử rằng mỗi đặc trưng tương ứng với sự tồn tại của một từ trong bag-of-words, vì vậy ta cần $P(w_i|c)$, và ta chỉ cần tính tỷ số số lần từ $w_i$ xuất hiện trong toàn bộ các từ của các văn bản thuộc về lớp $c$. Đầu tiên ta hợp nhất toàn bộ các văn bản thuộc lớp $c$ vào một văn bản lớn. Rồi sau đó ta dùng tấn suất từ $w_i$ trong văn bản lớn này làm ước lượng hợp lý cực đại cho xác suất:

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)} \tag{4.12}$$

Trong đó từ điển $V$ chứa tất cả các từ ở các lớp, không chỉ ở trong một lớp $c$.

Tuy nhiên, ta gặp phải vấn đề khi huấn luyện ước lượng hợp lý cực đại. Giả sử ta cần ước lượng xác suất cho từ "fantastic" thuộc về lớp *positive*, nhưng lại không có văn bản nào vừa chứa từ "fantastic" vừa thuộc lớp *positive*. Có thể từ "fantastic" lại nằm trong một văn bản thuộc lớp "negative" ý nghĩa mỉa mai. Trong trường hợp đó, xác suất cho đặc trưng này sẽ bằng không:

$$\hat{P}(\text{"fantastic"}|\text{positive}) = \frac{count(\text{"fantastic"}, \text{positive})}{\sum_{w \in V} count(w, \text{positive})} \tag{4.13}$$

Nhưng vì thuật toán nhân các xác suất đặc trưng với nhau nên chỉ cần một ước lượng nào đó bằng không thì xác suất của cả lớp đó bằng không.

Một giải pháp đơn giản là sử dụng phương pháp làm mịn add-one (Laplace) đã giới thiệu trong chương 2.

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V}(count(w, c) + 1)} = \frac{count(w, c) + 1}{(\sum_{w \in V} count(w, c)) + |V|} \tag{4.14}$$

Bây giờ chúng ta sẽ phải làm gì với những từ xuất hiện trong tập test mà không xuất hiện trong tập luyện? Thông thường ta sẽ bỏ qua những từ chưa biết đó và loại bỏ chúng ra khỏi văn bản test và không thêm các xác suất của chúng.

Cuối cùng, một số hệ thống hoàn toàn bỏ qua một lớp các từ: **stop word**. Đó là các từ xuất hiện nhiều lần như *the* và *a*. Điều này có thể làm bằng việc sắp xếp từ điển theo tần suất xuất hiện trong tập luyện, tìm ra 10-100 từ đầu tiên là stop word, rồi loại bỏ chúng ra khỏi các văn bản. Tuy nhiên ở trong hầu hết các ứng dụng phân loại văn bản, việc loại bỏ các stop word không cải thiện hoạt động, vì vậy nên thường sử dụng toàn bộ từ điển thay vì loại bỏ các stop word.

## 4.3 Ví dụ

Chúng ta xem xét một ví dụ về việc huấn luyện và kiểm thử Naive Bayes với phương pháp làm mịn Add-1 với bài toán phân tích cảm xúc, bài toán được chia thành 2 lớp: tích cực (+) và tiêu cực (-).

| | Cat | Documents |
|---|---|---|
| Training | - | just plain boring |
| | - | entirely predictable and lacks energy |
| | - | no surprises and very few laughs |
| | + | very powerful |
| | + | the most fun film of the summer |
| Test | ? | predictable with no fun |

Xác suất tiên nghiệm $P(c)$ cho 2 lớp dạng được tính theo công thức 4.11

**function** TRAIN NAIVE BAYES(D, C) **returns** log $P(c)$ and log $P(w|c)$

**for each** class $c \in C$ # Calculate $P(c)$ terms
  $N_{doc}$ = number of documents in D
  $N_c$ = number of documents from D in class c
  $logprior[c] \leftarrow \log \frac{N_c}{N_{doc}}$
  $V \leftarrow$ vocabulary of D
  $bigdoc[c] \leftarrow$ **append**(d) **for** d $\in$ D **with** class $c$
  **for each** word $w$ in V # Calculate $P(w|c)$ terms
    $count(w,c) \leftarrow$ # of occurrences of $w$ in $bigdoc[c]$
    $loglikelihood[w,c] \leftarrow \log \frac{count(w,c) + 1}{\sum_{w' \text{ in } V} (count(w',c) + 1)}$
**return** $logprior$, $loglikelihood$, $V$

**function** TEST NAIVE BAYES($testdoc$, $logprior$, $loglikelihood$, C, V) **returns** best $c$

**for each** class $c \in C$
  $sum[c] \leftarrow logprior[c]$
  **for each** position $i$ in $testdoc$
    $word \leftarrow testdoc[i]$
    **if** $word \in V$
      $sum[c] \leftarrow sum[c] + loglikelihood[word,c]$
**return** $\text{argmax}_c\ sum[c]$

Hình 4.2: Thuật toán Naive Bayes sử dụng phương pháp làm mịn Add-1. Để sử dụng Add-$\alpha$, thay 1 bằng $\alpha$

Khi đó:

$$P(-) = \frac{3}{5} \qquad P(+) = \frac{2}{5}$$

Áp dụng công thức 4.14:

$$P(\text{"predictable"}|-) = \frac{1+1}{14+20}$$
$$P(\text{"predictable"}|+) = \frac{0+1}{9+20}$$
$$P(\text{"no"}|-) = \frac{1+1}{14+20}$$
$$P(\text{"no"}|+) = \frac{0+1}{9+20}$$
$$P(\text{"fun"}|-) = \frac{0+1}{14+20}$$
$$P(\text{"fun"}|+) = \frac{1+1}{9+20}$$

Với câu kiểm tra S="*predictable with no fun*", sau khi xóa từ *with*, lớp được chọn thông qua biểu thức 4.9 được tính như sau:

$$P(-)P(S|-) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5}$$
$$P(+)P(S|+) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5}$$

Do $P(-)P(S|-) > P(+)P(S|+)$, câu S thuộc lớp tiêu cực (-)

## 4.4 Tối ưu hóa cho bài toán phân tích cảm xúc

Trong phần này, ta sẽ giới thiệu một biến thể của Naive Bayes, được gọi là binary multinominal naive Bayes (binary NB), tương tự như thuật toán naive Bayes, ta sử dụng công thức 4.10, ngoài ra, mỗi văn bản, chúng ta xóa các từ trùng lặp và giữ nguyên thứ tự của các từ còn lại trong văn bản. Hình 4.3 cho thấy một ví dụ gồm 4 câu (đã được rút ngắn và chuẩn hóa).

**Four original documents:**

- − it was pathetic the worst part was the boxing scenes
- − no plot twists or great scenes
- \+ and satire and great plot twists
- \+ great scenes great film

**After per-document binarization:**

- − it was pathetic the worst part boxing scenes
- − no plot twists or great scenes
- \+ and satire great plot twists
- \+ great scenes film

| | NB Counts | | Binary Counts | |
|---|---|---|---|---|
| | + | − | + | − |
| and | 2 | 0 | 1 | 0 |
| boxing | 0 | 1 | 0 | 1 |
| film | 1 | 0 | 1 | 0 |
| great | 3 | 1 | 2 | 1 |
| it | 0 | 1 | 0 | 1 |
| no | 0 | 1 | 0 | 1 |
| or | 0 | 1 | 0 | 1 |
| part | 0 | 1 | 0 | 1 |
| pathetic | 0 | 1 | 0 | 1 |
| plot | 1 | 1 | 1 | 1 |
| satire | 1 | 0 | 1 | 0 |
| scenes | 1 | 2 | 1 | 2 |
| the | 0 | 2 | 0 | 1 |
| twists | 1 | 1 | 1 | 1 |
| was | 0 | 2 | 0 | 1 |
| worst | 0 | 1 | 0 | 1 |

Hình 4.3: Ví dụ thuật toán Binary NB

Chú ý rằng, với Binary count, kết quả không nhất thiết phải là 1, vì từ có thể xuất hiện trong các câu khác nhau.

Một vấn đề chúng ta có thể thấy là, sự tiêu cực của một đánh giá rất mơ hồ. Ví dụ, với 2 câu sau *I really like this movie* (+) và *I didn't like this movie* (-), sự phân biệt rõ ràng ở đây là từ phủ định *didn't*, nhưng với trường hợp sau: *don't dismiss this film* và *doesn't let us get bored*, việc sử dụng từ phủ định để phân tích sẽ dẫn đến phân loại sai. Để giải quyết vấn đề này, ta thêm tiền tố *NOT* vào mỗi từ sau từ phủ định (*n't, not, no, never*) cho đến khi gặp dấu kết thúc câu tiếp theo. Ví dụ:

```
didn't like this movie, but I
```

thành:

```
didn't NOT_like NOT_this NOT_movie, but I
```

Khi đó, các từ *NOT_ like, NOT_ recommend* được phân loại là tiêu cực (-), *NOT_ bored, NOT_ dismiss* là tích cực (+)

## 4.5 Mô hình ngôn ngữ naive Bayes

Mô hình Naive Bayes có thể được xem như là một mô hình Unigram cho mỗi lớp dạng.

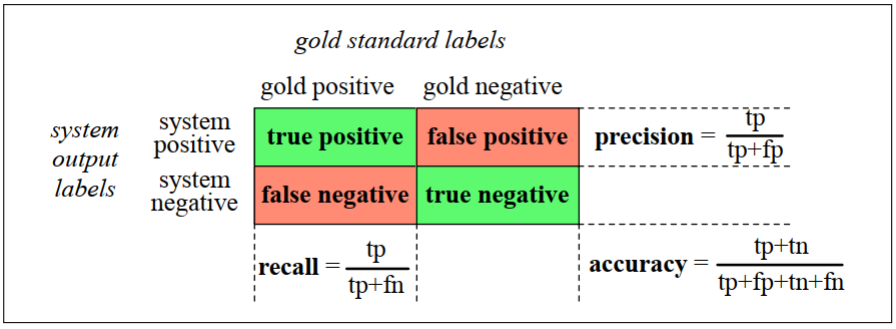
$$P(s|c) = \prod_{i \in position} P(w_i|c) \tag{4.15}$$

Xét mô hình naive Bayes với 2 lớp dạng tích cực (+) và tiêu cực (-):

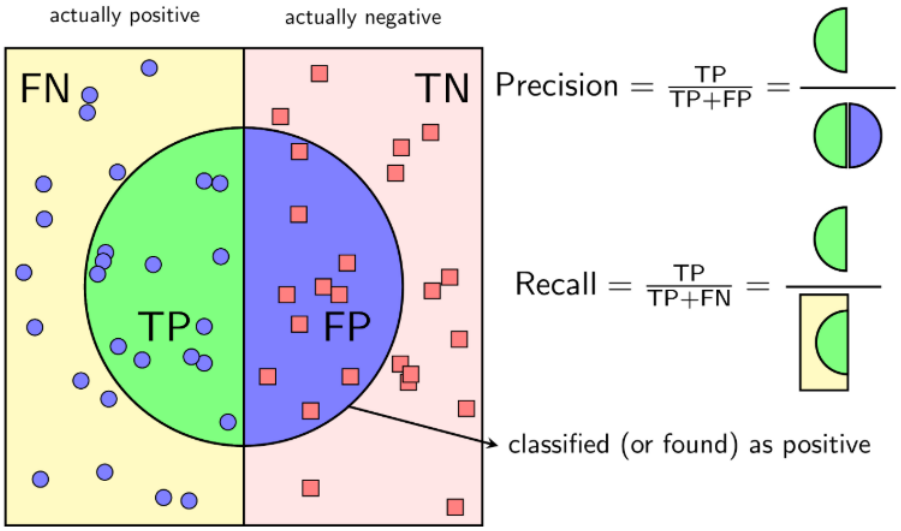
| w | P(w\|+) | P(w\|-) |
|---|---|---|
| I | 0.1 | 0.2 |
| love | 0.1 | 0.001 |
| this | 0.01 | 0.01 |
| fun | 0.05 | 0.005 |
| film | 0.1 | 0.1 |
| ... | ... | ... |

Khi đó:
$P(\text{"I love this fun film}| + \text{"}) = 0.1 \times 0.1 \times 0.01 \times 0.05 \times 0.1 = 0.0000005$
$P(\text{"I love this fun film}| - \text{"}) = 0.2 \times 0.001 \times 0.01 \times 0.005 \times 0.1 = 0.000000001$
Do, $P(s|+) > P(s|-)$, suy ra câu s thuộc tích cực (+).

## 4.6 Đánh giá: Precision, Recall, F-measure




Với bài toán phân loại mà tập dữ liệu của các lớp là chênh lệch nhau rất nhiều, có một phép đánh giá hiệu quả thường được sử dụng là Precison - Recall. Trước hết, xem xét bài toán phân loại nhị phân, ta coi một trong 2 lớp là positive (+) và negative (-). Xét hình 4.4 dưới đây:




Hình 4.4: Cách tính Precision và Recall

Với một cách xác định một lớp là positive (+), **Precision** được định nghĩa là tỷ lệ số điểm true positive trong số những điểm được phân loại là positive:

$$\textbf{Precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

**Recall** được định nghĩa là tỷ lệ điểm true positive trong số những điểm thực sự là positive:

$$\textbf{Recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

Có thể nhận thấy rằng, Recall và Precision đều là các số không âm, nhỏ hơn hoặc bằng 1. Precision cao đồng nghĩa với việc độ chính xác của điểm tìm được là cao, Recall cao đồng nghĩa với việc tỷ lệ bỏ sót các điểm thực sự là positive là thấp.

Trong thực tế, chúng ta thường kết hợp Precision và Recall để đánh giá mô hình, được gọi là độ đó **F-measure**, định nghĩa bằng công thức:

$$F_\beta = \frac{(\beta^2 + 1)P.R}{\beta^2 P + R}$$

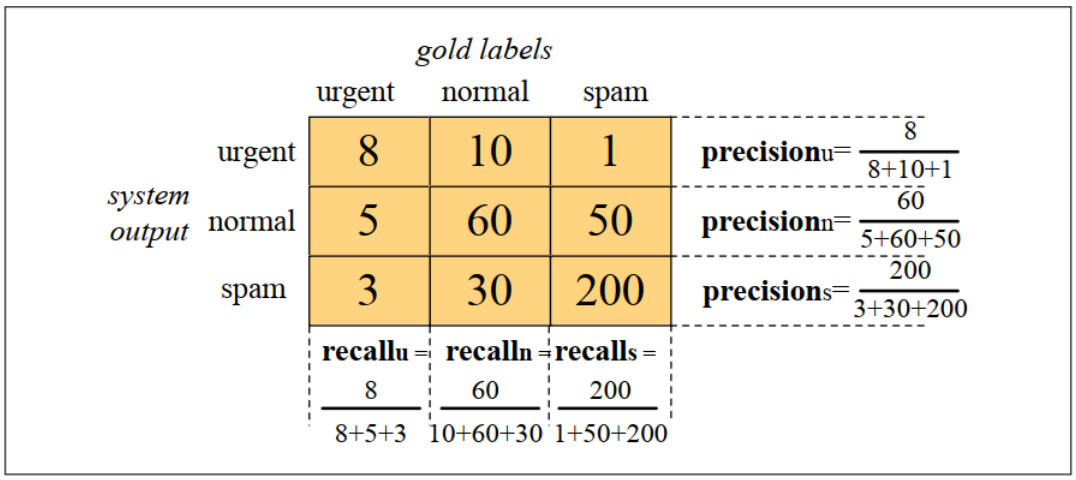
Tham số $\beta$ phụ thuộc vào từng bài toán. $\beta > 1$, ưu tiên Recall, $\beta < 1$, ưu tiên Precision, $\beta = 1$, Recall và Precision bằng nhau - trường hợp được sử dụng thường xuyên nhất, kí hiệu $F_1$:

$$F_1 = \frac{2P.R}{P + R} \tag{4.16}$$

Độ đo $F_1$ là harmonic mean của Precision và Recall. $F_1$ có giá trị trong nửa khoảng $(0, 1]$. $F_1$ càng cao, bộ phân loại càng tốt. Khi cả Recall và Precision đều bằng 1 (tốt nhất có thể), $F_1 = 1$.

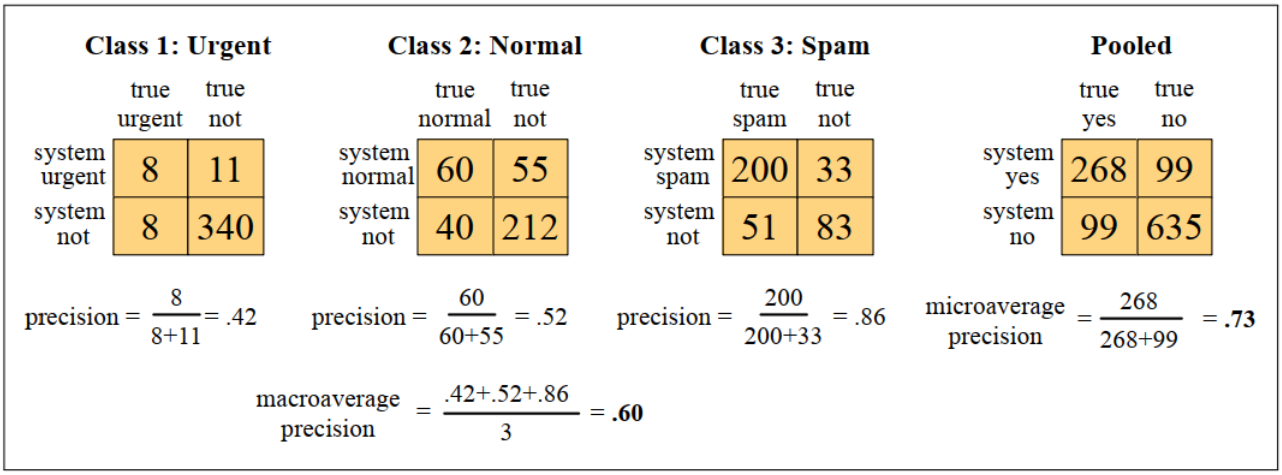
## 4.7 Bài toán phân loại có nhiều hơn 2 lớp

Có 2 loại bài toán phân loại nhiều lớp: mỗi phần tử có thể có nhiều nhãn và mỗi phần tử chỉ có 1 nhãn. Phổ biến hơn trong lớp các bài toán xử lý ngôn ngữ tự nhiên là phân loại với mỗi văn bản được gán với 1 nhãn duy nhất. Xem xét bài toán phân loại email gồm 3 nhãn: urgent (khẩn cấp), normal (bình thường) và spam. Hình **??** biểu diễn confusion matrix cho bài toán phân loại email.




Hình 4.5: Confusion matrix

Ma trận cho thấy hệ thống gán nhầm 1 email là spam thành urgent.



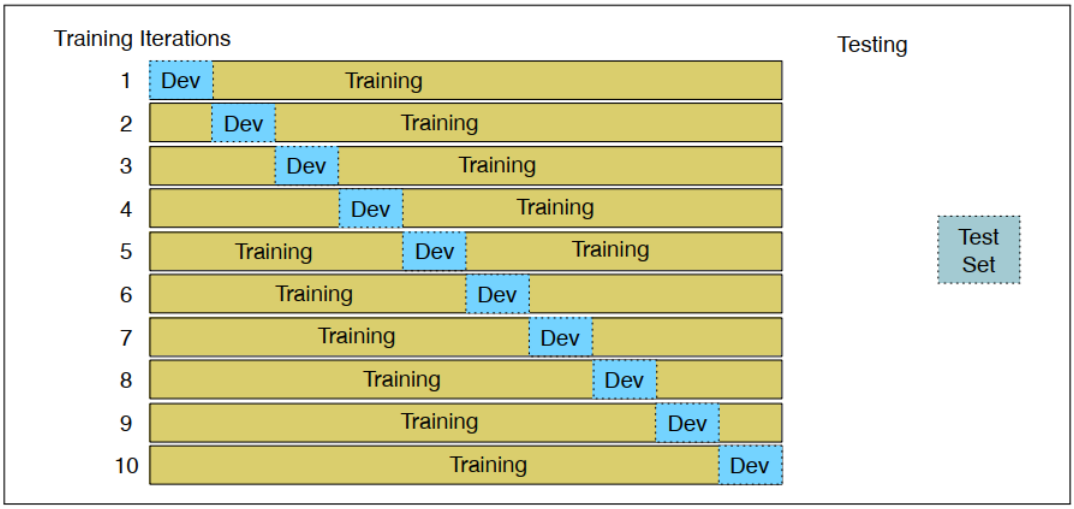

## 4.8 Tập huấn luyện và Cross-Validation

Overfitting là hiện tượng mô hình tìm được quá khớp với dữ liệu huấn luyện. Việc quá khớp này có thể dẫn đến việc dự đoán nhầm nhiễu, và chất lượng mô hình không còn tốt trên dữ liệu kiểm thử nữa. Vậy, có những kỹ thuật nào giúp tránh Overfitting?

Chúng ta vẫn quen với việc chia tập dữ liệu ra thành 2 tập: tập kiểm thử và tập huấn luyện. Trong đó, tập kiểm thử được giả sử là không biết trước và không được sử dụng để xây dựng mô hình. Vậy làm thế nào để biết được chất lượng của mô hình với tập dữ liệu chưa biết trước? Phương pháp đơn giản nhất là trích từ tập huấn luyện và một tập con và thực hiện việc đánh giá mô hình trên tập con này. Tập con được trích ra từ tập kiểm thử được gọi là validation set. Lúc này, tập huấn luyện là phần còn lại của tập huấn luyện ban đầu.

Trong nhiều trường hợp, chúng ta có hạn chế số lượng dữ liệu để xây dựng mô hình. Nếu lấy quá nhiều dữ liệu trong tập huấn luyện để làm dữ liệu validation, phần dữ liệu còn lại không đủ để xây dựng mô hình. Lúc này, tập validation phải nhỏ để tập huấn luyện đủ lớn. Tuy nhiên, một vấn đề khác nảy sinh, khi tập validation quá nhỏ, hiện tượng overfitting lại có thể xảy ra với tập huấn luyện còn lại. Giải pháp là sử dụng Cross-validation.

Cross-validation là một cải tiến của validatuon với lượng dữ liệu trong tập validation là nhỏ nhưng chất lượng mô hình được đánh giá trên nhiều tập validation khác nhau. Một cách thường được sử dụng là chia tập huấn luyện thành $k$ tập con không có phần tử chung, có kích thước gần bằng nhau. Tại mỗi lần kiểm thử, được gọi là run, một trong số $k$ tập con được lấy ra làm tập validation. Mô hình sẽ được xây dựng dựa vào hợp của $k-1$ tập con còn lại. Mô hình cuối được xác định dựa trên trung bình các tham số. Cách làm này còn có tên gọi là $k-foldvalidation$.




Hình 4.6: 10-fold crossvalidation

# Chương 5

# Hồi quy Logistic

Chúng ta chuyển sang một thuật toán thứ hai để phân loại được gọi là hồi quy logistic, đôi khi trong xử lý ngôn ngữ được gọi là mô hình entropy cực đại. Hồi quy logistic cùng loại với các mô hình phân loại theo cấp số mũ và cấp logarit tuyến tính. Ví dụ mô hình phân loại cấp logarit tuyến tính là Naive Bayes hoạt động trên nguyên tắc trích xuất một số đặc trưng có trọng số từ đầu vào, thực hiện lấy logarit và kết hợp tuyến tính. Về mặt kĩ thuật, hồi quy logistic là mô hình phân loại các mẫu quan sát vào một trong hai lớp, mà hồi quy logistic đa thức sử dụng khi muốn phân loại nhiều hơn hai lớp. Tuy nhiên, để cho ngắn gọn chúng ta sẽ gọi là hồi quy logistic kể cả trong trường hợp nhiều lớp.

Sự khác biệt quan trọng nhất giữa Naive Bayes và hồi quy logistic đó là hồi quy logistic là một mô hình phân loại phân biệt, trong khi đó Naive Bayes là mô hình phân loại sinh. Một mô hình phân loại đảm nhận công việc chọn nhãn đầu ra $y$ gán cho đầu vào $x$, và chọn $y$ sao cho cực đại $P(y|x)$. Trong mô hình Naive Bayes, chúng ta sử dụng công thức Bayes để ước lượng từ xác suất có điều kiện $P(x|y)$ và xác suất tiên nghiệm $P(y)$:

$$\hat{y} = \arg\max_y P(y|x) = \arg\max_y P(x|y)P(y) \tag{5.1}$$

Do đó, Naive Bayes là mô hình phân loại sinh: mô hình được huấn luyện để sinh dữ liệu $x$ từ lớp $y$. Xác suất có điều kiện $P(x|y)$ có nghĩa là từ lớp đầu vào $y$ ta cố gắng dự đoán đầu vào $x$. Sau đó chúng ta sử dụng quy tắc Bayes để tính toán xác suất hậu nghiệm chúng ta thực sự muốn $P(y|x)$.

Nhưng tại sao chúng ta không trực tiếp tính $P(y|x)$ ? Mô hình phân loại phân biệt sử dụng tư tưởng trực tiếp như vậy, tính giá trị $P(y|x)$ bằng cách phân biệt giữa các giá trị khác nhau có thể có của lớp y thay vì đầu tiên phải tính toán hợp lý:

$$\hat{y} = \arg\max_y P(y|x) \tag{5.2}$$

Trong khi hồi quy logistic khác về cách ước lượng xác suất, nó vẫn giống Naive Bayes trong việc đều là mô hình phân loại tuyến tính. Hồi quy logistic ước lượng $P(y|x)$ bằng

việc trích xuất một số đặc trưng có trọng số từ đầu vào, kết hợp tuyến tính và áp dụng một hàm số đã được xác định để thu được kết quả.

Chúng ta không thể tính trực tiếp giá trị $P(y|x)$ từ những đặc trưng và trọng số như sau:

$$P(y|x) \ ? = \sum_{i=1}^{N} w_i f_i \tag{5.3}$$

$$? = w \cdot f \tag{5.4}$$

Công thức $\sum_{i=1}^{N} w_i f_i$ nhận giá trị từ $-\infty$ đến $+\infty$ không thể đảm bảo giới hạn giá trị trong một khoảng cho phép, chẳng hạn như trong khoảng $(0, 1)$. Chúng ta có thể giải quyết vấn đề này bằng hai cách. Đầu tiên sử dụng hàm số *exp* để đưa về giá trị dương, sau đó sẽ chia cho một mẫu số thích hợp để đưa về một hàm xác suất hợp lý và có tổng bằng 1.

$$p(y = c|x) = p(c|x) = \frac{1}{Z} exp(\sum_{i} w_i f_i) \tag{5.5}$$

Một đặc trưng chỉ nhận giá trị trong khoảng $(0, 1)$ được gọi là hàm số chỉ thị. Hơn nữa, các hàm số chỉ thị không chỉ là một thuộc tính của quan sát $x$ mà còn là thuộc tính của đồng thời quan sát $x$ và lớp ứng cử viên đầu ra $c$. Do đó ta có:

$$p(c|x) = \frac{1}{Z} \exp\left(\sum_{i} w_i f_i(c, x)\right) \tag{5.6}$$

Cuối cùng, ta thực hiện phép chuẩn hóa để xác định yếu tố $Z$:

$$p(c|x) = \frac{\exp\left(\sum_{i} w_i f_i(c, x)\right)}{\sum_{c' \in C} \exp\left(\sum_{i} w_i f_i(c', x)\right)} \tag{5.7}$$

## 5.1 Hàm đặc trưng trong hồi quy logistic

Giả sử chúng ta xem xét bài toán phân loại văn bản, và chúng ta có ba lớp sắc thái văn bản $+, -$ và 0. Ta có thể định nghĩa 5 đặc trưng có tiềm năng:

$$f_1(c, x) = \begin{cases} 1 & \text{Nếu "great"} \ \in x \& c = + \\ 0 & otherwise \end{cases}$$

$$f_2(c, x) = \begin{cases} 1 & \text{Nếu "secon-rate"} \ \in x \& c = - \\ 0 & otherwise \end{cases}$$

$$f_3(c, x) = \begin{cases} 1 & \text{Nếu "no"} \ \in x \& c = - \\ 0 & otherwise \end{cases}$$

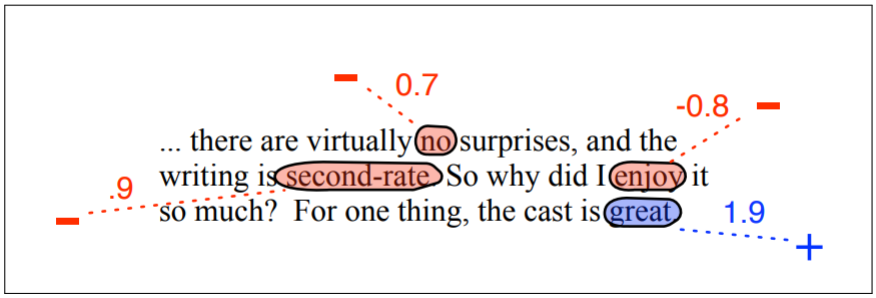
$$f_4(c,x) = \begin{cases} 1 & \text{Nếu "enjoy"} \ \in x \& c = - \\ 0 & otherwise \end{cases}$$

Các trọng số $w_i$ tương ứng đi với các đặc trưng $f_i$ sẽ đánh giá mức độ quan trọng của các từ ngữ xuất hiện trong văn bản ảnh hưởng tới sắc thái văn bản. Các trọng số này có thể nhận giá trị âm hoặc dương tương ứng với những từ ngữ mang sắc thái tiêu cực hoặc tích cực. Các hàm đặc trưng tương tự có thể được định nghĩa để giải quyết các bài toán phân loại khác trong xử lý ngôn ngữ tự nhiên.

## 5.2 Phân loại trong hồi quy logistic

Trong hồi quy logistic chúng ta lựa chọn lớp đầu ra bằng việc tính toán xác suất đầu ra cho mỗi lớp và lựa chọn lớp có xác suất lớn nhất.

Hình 5.1 chỉ ra một trường hợp ngoại lệ từ bản đánh giá phim với 4 đặc trưng được định nghĩa trong công thức 5.8 cho 2 lớp sắc thái văn bản với trọng số lần lượt là $w_1 = 1.9, w_2 = 0.9, w_3 = 0.7, w_4 = -0.8$.




Hình 5.1: Một số đặc trưng có trọng số cho lớp tích cực và tiêu cực. Chú ý rằng trọng số âm cho từ ıenjoy có tác dụng chống lại lớp tiêu cực.

Sử dụng 4 hàm đặc trưng trên và đầu vào $x$, $P(+|x)$ và $P(-|x)$ có thể được tính toán:

$$P(+|x) = \frac{e^{1.9}}{e^{1.9} + e^{0.9+0.7-0.8}} = 0.82 \tag{5.8}$$

$$P(+|x) = \frac{e^{0.9+0.7-0.8}}{e^{1.9} + e^{0.9+0.7-0.8}} = 0.18 \tag{5.9}$$

Nếu mục tiêu chỉ là phân loại, chúng ta có thể bỏ qua mẫu số và hàm số $exp$ và chỉ

cần chọn lớp có kết quả phép cộng tuyến tính cao nhất:

$$\hat{c} = \arg\max_{c \in C} P(c|x) \tag{5.10}$$

$$= \arg\max_{c \in C} \frac{\exp(\sum_{i=1}^{N} w_i f_i(c, x))}{\sum_{c' \in C} \exp(\sum_{i=1}^{N} w_i f_i(c', x))} \tag{5.11}$$

$$= \arg\max_{c \in C} \exp(\sum_{i=1}^{N} w_i f_i(c, x)) \tag{5.12}$$

$$= \arg\max_{c \in C} \sum_{i=1}^{N} w_i f_i(c, x) \tag{5.13}$$

Tuy nhiên, tính toán xác suất thực tế thay vì chỉ chọn lớp tốt nhất lại mang lại tiện lợi hơn khi mô hình được nhúng trong một hệ thống lớn, ví dụ như trong một chuỗi phân loại theo miền như gắn thẻ một phần của lời nói (phần 10.5). Chú ý rằng trong khi chỉ số của phép tính tổng trong biểu thức 5.13 thực hiện trong phạm vi toàn bộ danh sách N đặc trưng, trên thực tế là không cần thiết. Như trong bài toán phân loại văn bản, chúng ta không cần phải xem xét các đặc trưng mà không xuất hiện trong dữ liệu kiểm thử.

## 5.3 Huấn luyện hồi quy logistic

Trong phần này, chúng ta sẽ tìm hiểu cách để ước lượng tham số $w$ bằng việc sử dụng phương pháp ước lượng hợp lý cực đại có điều kiện. Điều đó có nghĩa là ta sẽ ước lượng tham số $w$ sao cho cực đại xác suất logarit tự nhiên của nhãn $y$ ứng với quan sát $x$ trong tập dữ liệu huấn luyện.

Với một mẫu quan sát $x$ trong dữ liệu luyện, trọng số tối ưu $w$ là:

$$\arg\max_{w} \log P(y^{(j)}|x^{(j)}) \tag{5.14}$$

Như vậy, với một số dữ liệu luyện, ta lựa chọn $w$ thỏa mãn:

$$\arg\max_{w} \sum_{j} \log P(y^{(j)}|x^{(j)}) \tag{5.15}$$

Thực hiện cực đại hóa hàm mục tiêu $L$:

$$
\begin{aligned}
L(w) &= \sum_j \log P(y^{(j)}|x^{(j)}) \\
&= \sum_j \log \frac{\exp(\sum_{i=1}^{N} w_i f_i(y^{(j)}, x^{(j)}))}{\sum_{y' \in Y} \exp(\sum_{i=1}^{N} w_i f_i(y^{(j)}, x^{(j)}))} \\
&= \sum_j \log \exp(\sum_{i=1}^{N} w_i f_i(y^{(j)}, x^{(j)})) - \sum_j \log \sum_{y' \in Y} \exp(\sum_{i=1}^{N} w_i f_i(y^{(j)}, x^{(j)}))
\end{aligned}
$$

Bài toán tối ưu trên là một bài toán tối ưu lồi, do vậy ta có thể sử dụng giải thuật leo đồi như Stochastic Gradient hoặc phương pháp Gradient liên hợp. Phương pháp Stochastic Gradient bắt đầu với một bộ trọng số $w$ khởi tạo ngẫu nhiên hoặc có thể bằng 0 và dịch chuyển theo hướng của Gradient $L'(w)$ (đạo hàm riêng của hàm mục tiêu $L$ ứng với các trọng số). Trong không gian $k$ chiều, đạo hàm này có thể được biểu thị bằng sự sai khác giữa 2 số:

$$L'(w) = \sum_j f_k(y^{(j)}, x^{(j)}) - \sum_j \sum_{y' \in Y} P(y'|x^{(j)}) f_k(y'^{(j)}, x^{(j)}) \tag{5.16}$$

Hai giá trị trên cho ta thấy một cách giải thích rất gọn gàng. Giá trị đầu tiên là số lượng đặc trưng $f_k$ trong dữ liệu, giá trị tiếp theo là số lượng $f_k$ dự kiến theo xác suất được chỉ định bởi mô hình hiện tại:

$$L'(w) = \sum_j Observed_count(f_k) - Expected_count(f_k) \tag{5.17}$$

Do đó, các trọng số tối ưu cho mô hình phù hợp với số lượng thực tế của các đặc trưng trong dữ liệu.

## 5.4 Regularization

Có một vấn đề trong huấn luyện mô hình đó là các trọng số được huấn luyện khiến mô hình hoàn toàn phù hợp với dữ liệu đào tạo, nhưng mang lại độ chính xác thấp trên dữ liệu kiểm thử, được gọi là overfitting. Để tránh overfitting, regularization được sử dụng trong 5.16. Thay vì tối ưu 5.15, ta thực hiện tối ưu:

$$\hat{w} = \arg\max_w \sum_j \log P(y^{(j)}|x^{(j)}) - \alpha R \tag{5.18}$$

với $R(w)$ là đại lượng regularization, được sử dụng để giảm thiểu các trọng số lớn khiến cho mô hình quá phụ thuộc vào các đặc trưng tương ứng đó. Điều này đương nhiên sẽ dẫn đến việc giảm thiểu sự phù hợp giữa mô hình và dữ liệu huấn luyện.

Có hai tiêu chuẩn regularization thông dụng. L2 Regularization là hàm số bậc hai của các giá trị trọng số. Chuẩn L2, $||W||_2$ giống với khoảng cách Euclidean:

$$[R(W) = ||W||_2^2 = \sum_{j=1}^{N} w_j^2 \tag{5.19}$$

Hàm mục tiêu L2 Regularization trở thành:

$$\hat{w} = \arg\max_w \sum_j \log P(y^{(j)}|x^{(j)}) - \alpha \sum_{j=1}^{N} w_j^2 \tag{5.20}$$

L1 Regularization là hàm tuyến tính của các trọng số, sử dụng chuẩn L1 $||W||_1$ giống với khoảng cách Manhattan:

$$[R(W) = ||W||_1 = \sum_{j=1}^{N} |w_j| \tag{5.21}$$

Hàm mục tiêu L1 Regularization trở thành:

$$\hat{w} = \arg\max_w \sum_j \log P(y^{(j)}|x^{(j)}) - \alpha \sum_{j=1}^{N} |w_j| \tag{5.22}$$

Các loại regularization này đến từ phân tích thống kê, trong đó L1 Regularization được gọi là hồi quy lasso (Tibshirani, 1996) và L2 Regularization được gọi là hồi quy ridge, và cả hai đều được sử dụng thông dụng trong xử lý ngôn ngữ. L2 Regularization dễ hơn trong việc tối ưu vì tính đơn giản của đạo hàm, trong khi đó L1 có chút phức tạp hơn (đạo hàm không liên tục tại 0). Trong khi L2 tạo ra các trọng số với giá trị nhỏ, L1 thích hợp hơn trong việc tạo ra bộ trọng số trong đó một số trọng số lớn kèm với rất nhiều các trọng số bằng 0, có nghĩa là ít hơn nhiều một số lượng các đặc trưng.

Cả L1 và L2 Regularization đều có diễn giải Bayesian. L1 Regularization có thể giả định các trọng số tuân theo hàm mật độ xác suất Laplace, trong khi L2 Regularization tương ứng với giả định rằng các trọng số tuân theo phân phối chuẩn có trung mình $\mu = 0$. Trong phân phối chuẩn, giá trị càng xa giá trị trung bình, xác suất càng thấp (được chia theo tỉ lệ phương sai $\sigma^2$). Gaussian cho một trọng số $w_j$ là:

$$\frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(w_j - \mu_j)^2}{2\sigma_j^2}\right) \tag{5.23}$$

Nếu ta nhân mỗi trọng số với hàm mật độ xác suất chuẩn, ta sẽ thực hiện cực đại hàm số sau:

$$\hat{w} = \arg\max_w \prod_j^M P(y^{(j)}|x^{(j)}) \times \prod_j^M \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(w_j - \mu_j)^2}{2\sigma_j^2}\right) \tag{5.24}$$

với trong không gian logarith tự nhiên, với $\mu = 0$ và giả thiết $2\sigma^2 = 1$, ta có:

$$\hat{w} = \arg\max_w \sum_j \log P(y^{(j)}|x^{(j)}) - \alpha \sum_{j=1}^{N} w_j^2 \tag{5.25}$$

giống với công thức 5.20.

## 5.5 Trích chọn đặc trưng

Kĩ thuật regularization được giới thiệu trong phần trước rất hữu ích để tránh overfitting bằng việc loại bỏ hoặc giảm thiểu các đặc trưng không có khả năng khái quát tốt. Tuy nhiên rất nhiều mô hình phân loại, bao gồm Naive Bayes, không có regularization, và do đó thay vì trích chọn đặc trưng được sử dụng để giữ các đặc trưng quan trọng và loại bỏ các đặc trưng còn lại. Cơ sở của trích chọn đặc trưng đó là giả định một số metric về hiệu quả của từng đặc trưng, xếp hạng cho các đặc trưng, lựa chọn các đặc trưng có xếp hạng cao và số lượng các đặc trưng giữ được gọi là tham số meta có thể được tối ưu hóa dựa trên tập kiểm thử dev.

Các đặc trưng thường được xếp hạng dựa theo mức độ thông tin mà chúng đóng góp ảnh hưởng tới quyết định phân loại. Một metric rất thông dụng đó là thông tin đạt được. Thông tin đạt được cho chúng ta biết có bao nhiêu bit thông tin mà sự hiện diện của từ mang lại cho chúng ta quyết định phân vào lớp và có thể được tính như sau ($C$ là tổng số các lớp):

$$G(w) = -\sum_{i=1}^{C} P(c_i) \log P(c_i) \tag{5.26}$$

$$+ P(w) \sum_{i=1}^{C} P(c_i|w) \log P(c_i|w) \tag{5.27}$$

$$+ P(\bar{w}) \sum_{i=1}^{C} P(c_i|\bar{w}) \log P(c_i|\bar{w}) \tag{5.28}$$

Mặc dù trích chọn đặc trưng là rất quan trọng với các mô hình phân loại unregularization, đôi khi nó cũng được sử dụng trong các mô hình phân loại regularization ưu tiên về mặt tốc độ vì nó giảm thiểu số lượng các đặc trưng, dẫn đến giảm thiểu chi phí tính toán.

## 5.6 Lựa chọn mô hình phân loại và các đặc trưng

Hồi quy logistic có một số lợi thế so với Naive Bayes. Các giả định độc lập của Naive Bayes có nghĩa là nếu hai đặc trưng trên thực tế có tương quan lẫn nhau thì Naive Bayes sẽ nhân cả hai như thể chúng độc lập. Hồi quy logistic tốt hơn đối với các đặc

trưng tương quan lẫn nhau. Nếu hai đặc trưng $f_1$ và $f_2$ là hoàn toàn tương quan, hồi quy logistic sẽ đơn giản giả định 1/2 trọng số thuộc về $w_1$ và 1/2 trọng số thuộc về $w_2$.

Do đó, khi có nhiều đặc trưng tương quan lẫn nhau, hồi quy logistic sẽ chỉ định xác suất chính xác hơn so với Naive Bayes. Mặc dù độ chính xác thấp hơn, Naive Bayes vẫn thường đưa ra quyết định phân loại chính xác. Hơn nữa, Naive Bayes hoạt động rất tốt (thậm chí tốt hơn hồi quy logistic hoặc SVMs) trong các bộ dữ liệu nhỏ (Ng và Jordan, 2002) hoặc văn bản ngắn (Wang và Manning, 2012). Và Naive Bayes rất dễ thực hiện và đào tạo rất nhanh. Tuy nhiên các thuật toán như hồi quy logistic và SVM thường hoạt động tốt hơn trên các văn bản hoặc bộ dữ liệu lớn hơn.

Lựa chọn mô hình phân loại cũng bị ảnh hưởng bởi sự đánh đổi sai lệch. Sự thiên vị của một mô hình phân loại chỉ ra độ chính xác của việc mô hình hóa các bộ dữ liệu huấn luyện khác nhau. Mục đích biến đổi của mô hình phân loại cho biết mức độ quyết định của nó bị ảnh hưởng bởi sự thay đổi nhỏ trên dữ liệu luyện. Mô hình với độ thiên vị thấp (SVMs với hàm nhân đa thức hoặc hàm nhân RBF) là rất chính xác trong việc mô hình hóa dữ liệu huấn luyện. Mô hình với phương sai thấp (như Naive Bayes) có khả năng đi đến cùng một quyết định phân loại ngay cả từ sự hơi khác nhau từ dữ liệu huấn luyện. Nhưng các mô hình thiên vị thấp có xu hướng rất chính xác trong việc khớp với dữ liệu luyện, và khó mô hình hóa các bộ dữ liệu khác. Và các mô hình phương sai thấp có xu hướng khái quát hóa tốt cho nên có thể dẫn đến đạt độ chính xác không cao bằng, nhưng mang lại độ chính xác tương đối với các bộ dữ liệu kể cả không có cùng phân phối với dữ liệu luyện. Do đó, bất kì mô hình nào cũng có sự đánh đổi giữa thiên vị và phương sai. Thêm nhiều đặc trưng sẽ giảm thiểu sự thiên vị nhưng tăng phương sai bởi overfitting.

Ngoài việc lựa chọn một mô hình phân loại, chìa khóa để tạo ra mô hình phân loại tốt còn nằm ở việc thiết kế các đặc trưng một cách phù hợp. Các đặc trưng thường được thiết kế bằng cách kiểm tra dữ liệu luyện bằng trực giác với kiến thức phù hợp với từng miền bài toán. Sự phân tích cẩn thận trên tập huấn luyện, hoặc tập kiểm thử dev từ một số phiên bản đầu tiên của hệ thống thường cung cấp cái nhìn sâu sắc về các đặc trưng.

Đối với một số tác vụ, đặc biệt hữu ích khi xây dựng các đặc trưng phức tạp đó là sự kết hợp từ các đặc trưng đơn giản hơn. Đối với hồi quy logistic và Naive Bayes, sự kết hợp các đặc trưng này phải được thực hiện bằng tay.

Một số mô hình học máy khác có thể tự động mô hình hóa các tương tác giữa các đặc trưng. Đối với các tác vụ trong đó sự kết hợp các đặc trưng này là quan trọng (đặc biệt khi kết hợp các đặc trưng phân loại và các đặc trưng mang giá trị thực). Các mô hình phân loại hữu ích nhất bao gồm SVMs với hàm nhân đa thức hoặc RBF, và rừng ngẫu nhiên.

## 5.7 Tổng kết

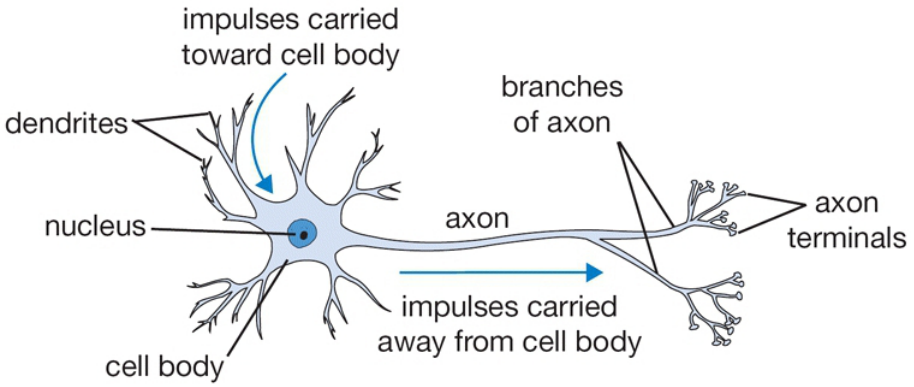
Trong chương này đã trình bày về hồi quy logistic dùng trong phân loại.

- Hồi quy logistic (có thể gọi là MaxEnt hay cực đại Entropy trong xử lý ngôn ngữ) là mô hình phân loại phân biệt chỉ định nhãn cho một quan sát bằng việc tính xác suất từ một hàm số mũ của các đặc trưng với các trọng số tương ứng của nó.
- Regularization là quan trọng trong mô hình MaxEnt để tránh overfitting.
- Trích chọn đặc trưng có thể hữu ích trong việc loại bỏ các đặc trưng không hữu ích để tăng tốc đào tạo, và cũng rất quan trọng trong các mô hình unregularization để tránh overfitting.

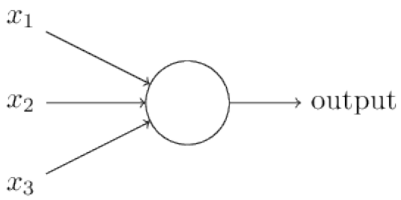
# Chương 6

# Neural Networks

**Neural** là mô hình toán học và biểu hiện cho một số chức năng của neurone thần kinh trong não bộ con người.

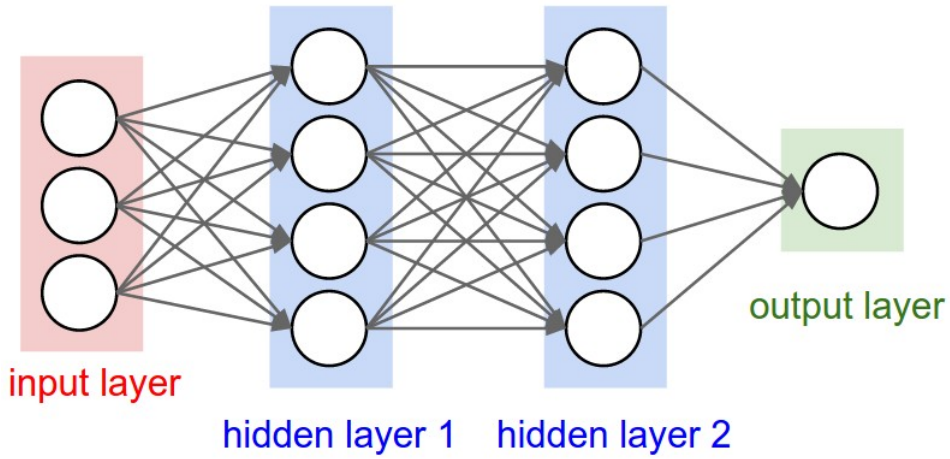



Hình 6.1: Neurone sinh học




Hình 6.2: Mô hình perceptron

**Neural network** dù đã ra đời từ lâu nhưng nó vẫn là một công cụ thiết yếu trong các bài toán xử lý ngôn ngữ. Từ network bắt nguồn từ *McCulloch-Pitts (1943)*, là một mô hình dựa trên hệ thần kinh của con người. Trong chương này, chúng ta giới thiệu về neural net classifier, được tạo nên bằng cách kết hợp các **unit** thành một mạng lưới. Đây được gọi là **feed-forward network** bởi vì phép tính được tiến hành lặp đi lặp lại từ lớp unit này đến lớp unit tiếp theo. Neural network là tập hợp của nhiều neural, các tập hợp này tạo thành một cấu trúc hệ thống gồm các lớp, còn gọi là các *layer*. Trong




Hình 6.3: Mô hình Neural Network

đó, ta sẽ thấy một neural network đơn hay một đơn vị neural network thường có các lớp: Input layer (lớp đầu vào), Hidden layer (lớp ẩn), Output layer (lớp đầu ra). Các lớp này có nhiệm vụ xử lý tín hiệu nhận được theo thứ tự lớp sau nhận giá trị output của lớp trước để tiến hành. Việc các lớp này xử lý theo cách nào thường phụ thuộc vào từng yêu cầu khác nhau. Số lượng các lớp ẩn là không giới hạn. Số lớp ẩn và cách xử lý ở từng lớp kể trên sẽ quyết định kết quả và hiệu quả của bài toán cần xử lý.

Hình ảnh ví dụ một neural network

**Deep learning** là một neural network với nhiều lớp ẩn, mỗi lớp ẩn có một nhiệm vụ riêng, output của tầng này sẽ là output của tầng sau.

Logistic regression là mô hình neural network đơn nhất nhất chỉ với 2 lớp là input layer và output layer.

## 6.1 Units

Nền tảng của một neural network là các **unit** tính toán độc lập. Các unit thực hiện một số tính toán trên bộ giá trị thực input và một output. Một unit sẽ lấy tổng trọng số của các input và một hệ số tự do (được gọi là **bias**). Khi input là một tập hợp $x_1 \dots x_n$, một unit sẽ có một tập trọng số tương ứng $w_1 \dots w_n$ và bias $b$, công thức tổng trọng số $z$:

$$z = b + \sum_i w_i x_i \tag{6.1}$$

$$z = w.x + b \tag{6.2}$$

Cuối cùng, thay vì sử dụng hàm tổng trọng số $z$ là một hàm tuyến tính của $x$, các unit áp dụng hàm phi tuyến $f$, output của hàm này là giá trị kích hoạt (activation value) cho unit, đặt là $a$. Giá trị kích hoạt cho một mạng đặt là $y$, xác định $y$ bằng công thức:

$$y = a = f(z) \tag{6.3}$$

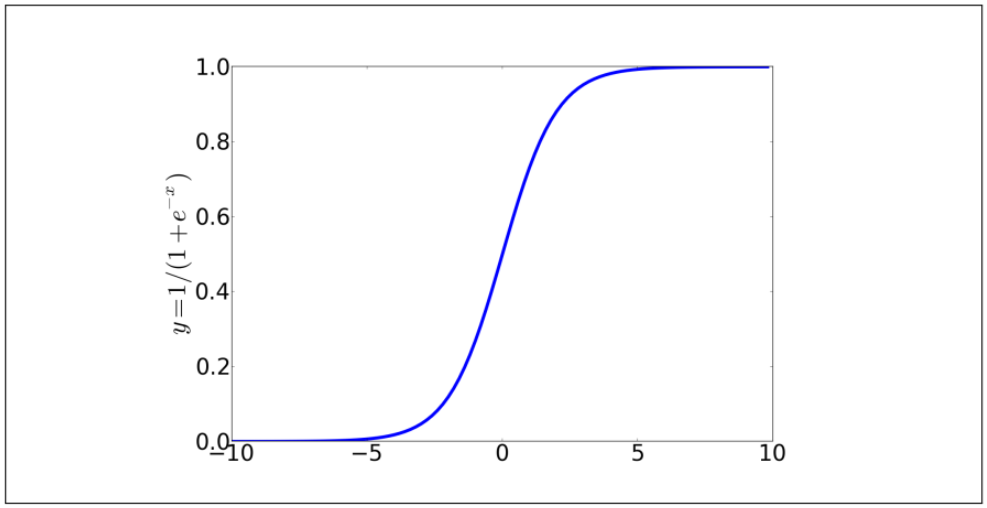
Trong đó, $f$ được gọi là các hàm kích hoạt (activation function). Hàm kích hoạt là những hàm phi tuyến được áp dụng vào output của các layer trong tầng ẩn của một mô hình mạng, và được sử dụng làm input cho layer tiếp theo. Trong phần này, chúng ta sẽ giới thiệu một số hàm kích hoạt phổ biến (hàm *sigmoid*, hàm tanh, hàm *ReLU* - rectified linear).

Ở chương trước, chúng ta đã giới thiệu về Logistic Regression, có thể thấy linh hồn của logistic regression chính là hàm *sigmoid*. Công thức hàm *sigmoid*:

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{6.4}$$

Hàm *sigmoid* là một hàm phi tuyến với input là các số thực và cho kết quả nằm trong khoảng $[0, 1]$ và được xem là xác suất trong một số bài toán. Trong hàm *sigmoid*, một sự thay đổi nhỏ trong input dẫn đến một output không mấy thay đổi. Vì vậy, nó đem lại một output mịn hơn và liên tục hơn so với input (hình 6.4). Thay vào công thức 6.2, ta được:

$$y = \sigma(w.x + b) = \frac{1}{1 + e^{-(w.x+b)}} \tag{6.5}$$




Hình 6.4: Dồ thị hàm *sigmoid*

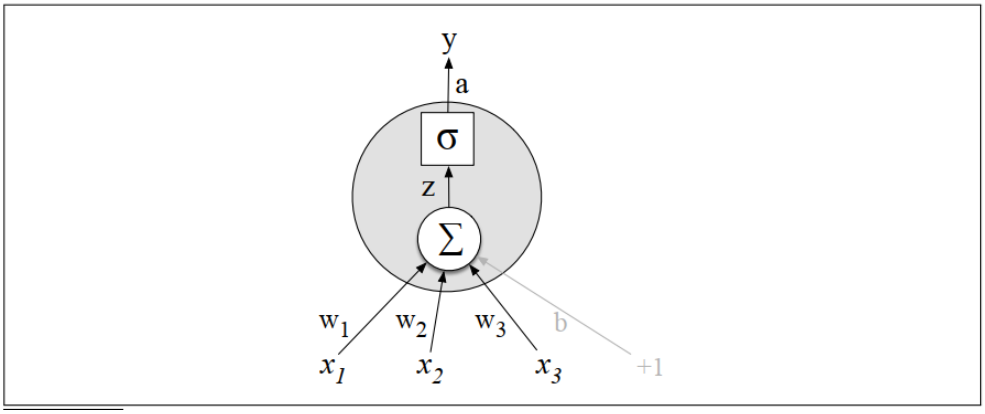
**Ví dụ 6.1.1.** *Cho một unit với các vector trọng số và hệ số bias như sau:*

$$w = [0.2, 0.3, 0.9]$$

$$b = 0.5$$

*Input:*

$$x = [0.5, 0.6, 0.1]$$




Hình 6.5: Mô hình một unit

Khi đó:

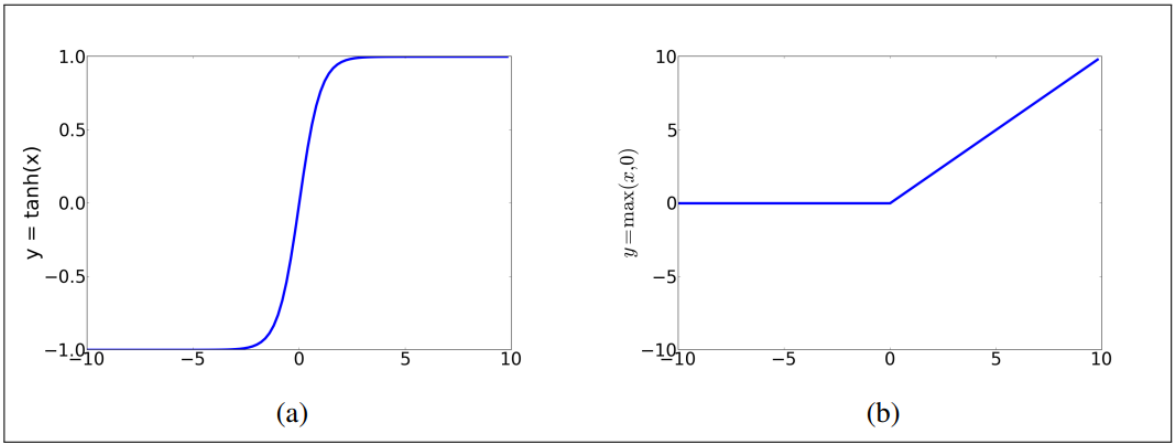
$$y = \sigma(w.x + b) = \frac{1}{1 + e^{-(w.x+b)}} = \frac{1}{1 + e^{-(0.5*0.6+0.3*0.6+0.8*0.1+0.5)}} = e^{-0.86} = 0.42$$

Ngoài hàm *sigmoid*, một số hàm phi tuyến khác cùng thường được sử dụng, như hàm tanh, một biến thể của hàm *sigmoid* trả output trong khoảng $[-1, 1]$ (hình 6.6a):

$$y = \tanh z = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{6.6}$$

Một hàm kích hoạt khác rất phổ biến đó là hàm rectified linear unit, còn được gọi là hàm *ReLU* (hình 6.6b):

$$y = \max(x, 0) \tag{6.7}$$




Hình 6.6: Đồ thị hàm tanh và hàm *ReLU*

Các hàm kích hoạt đều có những ưu và nhược điểm riêng. Ví dụ hàm *ReLU* có ưu điểm là rất gần với một hàm tuyến tính. Trong khi, các hàm *sigmoid* và tanh thường gặp tình trạng bão hòa (saturated), khi sử dụng input có giá trị cực lớn ($+\infty$), output

của nó sẽ đạt giá trị rất gần 1, ngược lại, nếu giá trị input cực nhỏ ($-\infty$), output sẽ tiệm cận 0 (hàm *sigmoid*), và tiệm cận 1 với hàm tanh, gây khó khăn trong việc train tập dữ liệu. Ngược lại, các ưu điểm của hàm *sigmoid* và tanh là có thể phân biệt dễ dàng.

Trong hình 6.3, mô hình neural network có 1 input layer, 2 hidden layer, 1 output layer. Mỗi node trong hidden layer và output layer đều liên kết với tất cả các node ở layer trong đó với các trọng số $w$ riêng, mỗi node có một hệ số bias riêng, tại mỗi node, mô hình sẽ diễn ra 2 bước: tính tổng trọng số và áp dụng hàm kích hoạt (activation fuction). Đặt số node trong layer thứ $i$ là $\ell^i$. Ma trận $W^k$ kích thước $\ell^{k-1} \times \ell^k$ là ma trận trọng số giữa layer $k-1$ và layer $k$, trong đó $w_{ij}^k$ là trọng số kết nối từ node thứ $i$ của layer $k-1$ đến node thứ $j$ của layer $k$. Vector $b^k$ kích thước $\ell^k \times 1$ là hệ số bias của các node trong layer $k$, trong đó $b_i^k$ là bias của node thứ $i$ trong layer $k$. Với node thứ $i$ trong layer $\ell$ có bias $b_i^\ell$ thực hiện 2 bước:

- Tính tổng trọng số $z_i^\ell = \sum_{j=1}^{\ell-1} a_j^{\ell-1} \times w_{ij}^\ell + b_i^\ell$, là tổng giá trị tất cả các node trong layer trước nhân với trọng số $w$ tương ứng, rồi cộng với bias $b$.
- Áp dụng hàm kích hoạt: $y_i^\ell = \sigma(z_i^\ell)$

Vector $z^k$ kích thước $\ell^k \times 1$ là giá trị các node trong layer $k$ sau bước tính tổng trọng số. Vector $y^k$ kích thước $\ell^k \times 1$ là giá trị của các node trong layer $k$ sau khi áp dụng hàm kích hoạt.

## 6.2 Toán tử XOR

Câu hỏi đặt ra ở đây, việc có nhiều layer có giúp mô hình có thể giải quyết được các bài toán phức tạp hơn hay không? Để trả lời câu hỏi này, ta xem xét một bài toán tính toán các hàm logic đơn giản AND, OR, XOR của các input. Dưới đây là bảng chân lý:

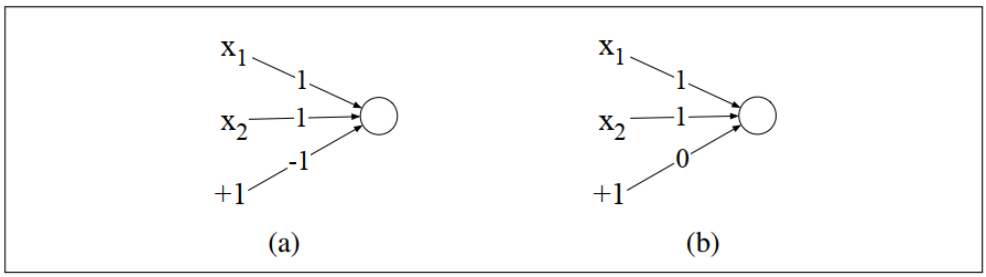
| AND | | | OR | | | XOR | | |
|---|---|---|---|---|---|---|---|---|
| x1 | x2 | y | x1 | x2 | y | x1 | x2 | y |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

Hình 6.7: Bảng chân lý

Perceptron là một unit đơn giản với output nhị phân và không có hàm kích hoạt phi tuyến. Output $y$ được biểu diễn:

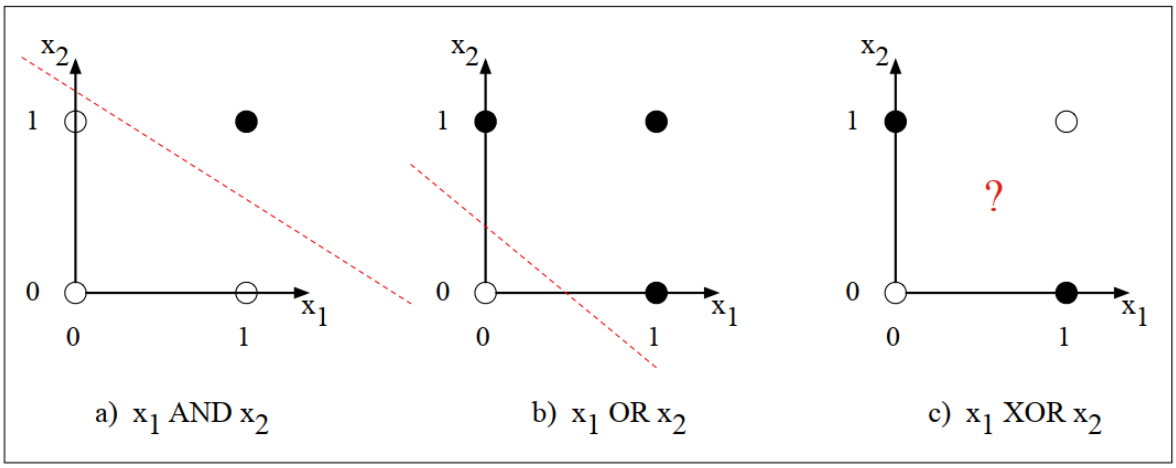
$$y = \begin{cases} 0 & w.x + b \leq 0 \\ 1 & w.x + b > 0 \end{cases} \tag{6.8}$$

Chúng ta cùng xét khả năng biểu diễn của perceptron (PLA) cho các bài toán biểu diễn các hàm logic nhị phân: AND, OR. (hình 6.8)




Hình 6.8: Các trọng số w và bias b của một perceptron để tính toán các hàm logic. Các input là $x_1$ và $x_2$ và bias là một nút đặc biệt có giá trị +1 được nhân với trọng số bias $b$. (a) logic AND, hiển thị trọng số $w_1 = 1$ và $w_2 = 1$ và trọng số bias $b = -1$ (b) logic OR, hiển thị trọng số $w_1 = 1$ và $w_2 = 1$ và trọng số bias $b = 0$.

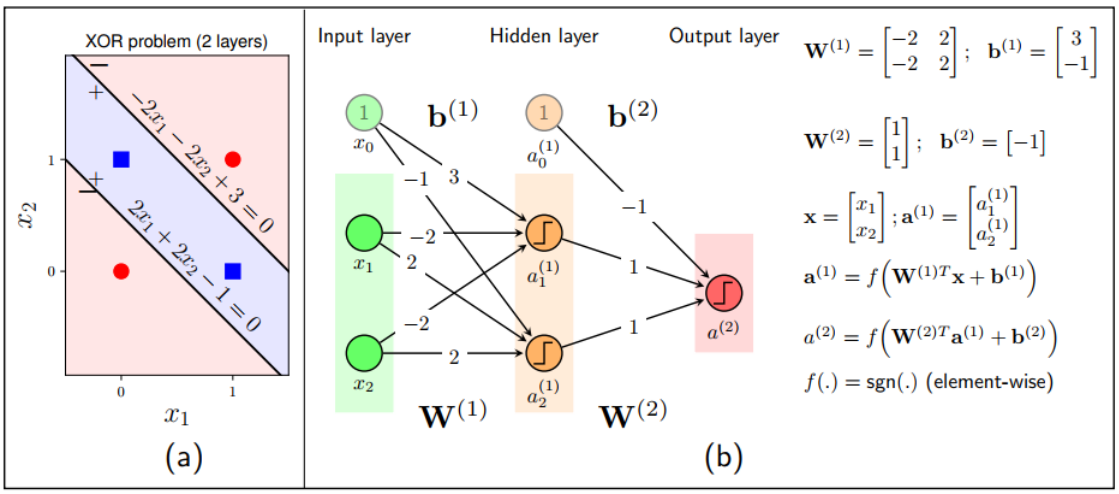
Nhận thấy rằng, với các bài toán OR, AND, dữ liệu giữa 2 lớp là linearly separable, vì vậy ta có thể tìm được các hệ số cho perceptron giúp biểu diễn chính xác mỗi hàm số. Với hàm XOR, vì dữ liệu không linearly separable, không thể biểu diễn bằng một perceptron. Nếu thay perceptron bằng logistic regression, tức là thay hàm kích hoạt từ hàm *sign* sang hàm *sigmoid*, ta cũng không tìm được hệ số thỏa mãn, vì về bản chất, logistic regression cũng chỉ tạo ra các ranh giới có dạng tuyến tính. Hình 6.9 cho thấy với các input logic $(00, 01, 10, 11)$, với phân loại AND và OR, luôn vẽ được 1 đường thẳng phân tách, nhưng không có cách nào để vẽ được một đường thẳng có thể tách với phân loại XOR.




Hình 6.9: Hàm logic AND, OR, XOR

Nhận thấy rằng, nếu cho phép sử dụng 2 đường thẳng, bài toán biểu diễn hàm XOR có thể được giải quyết như hình 6.10. Các trọng số tương ứng với 2 đường thẳng trong hình 6.10a được minh họa trên hình 6.10b bằng các mũi tên xuất phát từ các điểm màu lục và cam. Đầu ra $a_1^1$ bằng 1 với các điểm nằm về phóa (+) của đường thẳng

$3 - 2x_1 - 2x_2 = 0$, bằng $-1$ với các điểm nằm về phía (-). Tương tự, đầu ra $a_2^1$ bằng 1 với các điểm nằm về phía (+) của đường thẳng $-1 + 2x_1 + 2x_2 = 0$. Như vậy, 2 đường thẳng ứng với 2 perceptron này tạo ra 2 output tại các node $a_1^1$ và $a_2^1$. Vì hàm XOR chỉ có một output nên ta cần làm thêm một bước nữa: coi $a_1^1$ và $a_2^1$ như là một input của perceptron tiếp theo. Trong perceptron mới này, input là các node màu cam (giá trị bias của các node bằng 1), output là các node màu đỏ. Các hệ số được cho trên hình 6.10b. Kiểm tra lại một chút, với các điểm hình vuông xanh (hình 6.10a), $a_1^1 = a_2^1 = 1$, khi đó $a^2 = sgn(-1 + 1 + 1) = 1$. Với các điểm hình tròn đỏ, vì $a_1^1 = -a_2^1$ nên $a^2 = sgn(-1 + a_1^1 + a_2^1) = -1$. Trong cả 2 trường hợp, output dự đoán đều giống với output thực sự. Vậy nếu sử dụng 3 perceptron tương ứng với các ouput $a_1^1$,$a_2^1$ và $a^2$, ta sẽ biểu diễn được hàm XOR. 3 perceptron kể trên được xếp vào 2 layer: layer 1: input-lục, output-cam, layer 2: input-cam, output-đỏ. Ở đây đầu ra có layer thứ 1 chính là input của layer thứ 2. Vì vậy, hàm XOR có thể được tính bằng một tầng network với nhiều unit.

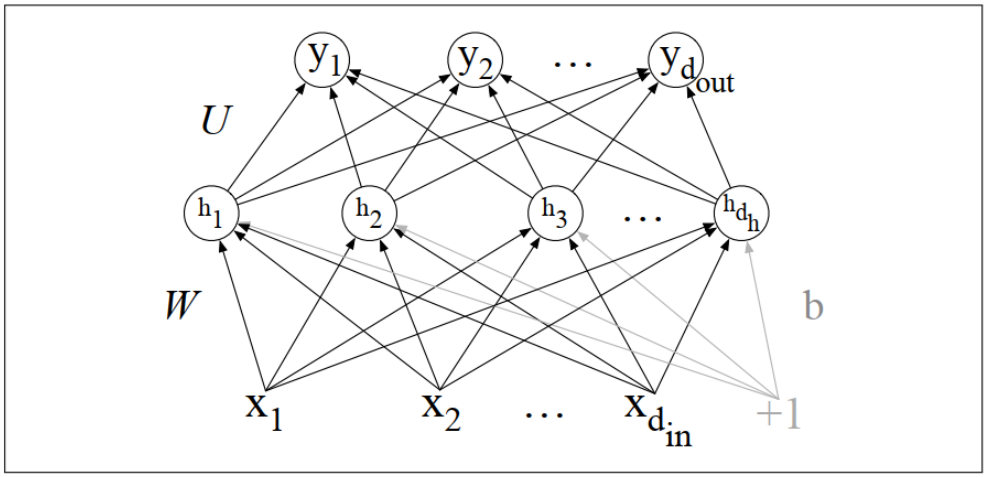



Hình 6.10: Ba perceptron biểu diễn hàm XOR.

## 6.3 Feed-forward Neural Network

Một neural network với nhiều hơn 2 layer còn được gọi là *multilayer neural network (MLPs), deep feed-forward network hoặc feed-forward network.* Từ feed-forward được hiểu là dữ liệu đi thẳng từ input tới output theo chiều các mũi tên mà không quay lại ở điểm nào, tức network có dạng acylic graph (đồ thị không chứa chu trình kín). Tên gọi perceptron ở đây có thể gây nhầm lẫn chút, vì cụm từ này để chỉ neural network với nhiều layer và mỗi layer không nhất thiết là một hoặc nhiều perceptron. Hàm kích hoạt có thể là các hàm phi tuyến khác thay vì hàm *sgn*.

Mô hình feed-forward network đơn giản gồm có 3 node: input unit, hidden unit, và output unit (hình 6.11)




Hình 6.11: Mô hình feed-forward neural network

Với mỗi hidden unit, mô hình có các tham số $w$ (vector trọng số) và hệ số bias $b$. Ma trận trọng số cho toàn bộ lớp ẩn, ký hiệu $W$, trong đó $W_{ij}$ là trọng số kết nối từ input unit thứ $i$ $x_i$ đến hidden unit thứ $j$ $h_j$. Việc tính toán của mạng gồm 2 bước: tính ma trận trọng số của vector input $x$ và áp dụng hàm kích hoạt $f$ (hàm *sigmoid*, tanh, *ReLU*). Dựa vào hàm *sigmoid* $\sigma$, vector $h$, output của hidden layer được biểu diễn với công thức:

$$h = \sigma(Wx + b) \tag{6.9}$$

Đặt $d_{in}$ là số lượng input. Hidden layer có kích thước $d_h \times 1$, $h \in \mathbb{R}^{d_h}$, $b \in \mathbb{R}^{d_h}$, $W \in \mathbb{R}^{d_h \times d_{in}}$. Khi đó, công thức 6.9 được biểu diễn:

$$h_{ij} = \sum_{i=1}^{d_{in}} \tag{6.10}$$

Cũng giống như hidden layer, output layer cũng có một ma trận trọng số, ký hiệu: $U$, thường thì $U$ không có vector bias $b$. Output trung gian $z$ được biễn diễn như sau:

$$z = U.h$$

Đặt $d_{out}$ là số node output, khi đó $z \in \mathbb{R}^{d_{out} \times d_h}$, trong đó $U_{ij}$ là trọng số từ unit $j$ trong hidden layer đến unit $i$ trong output layer. Với bài toán phân loại, $z$ không thể là output vì $z$ là một vector có các giá trị thực, trong khi với bài toán phân loại, chúng ta cần là một vector xác suất. Chúng ta cần một mô hình xác suất sao cho với mỗi input $x$, hàm $a_i$ thể hiện xác suất để input đó rơi vào lớp thứ $i$. Vậy điều kiện cần là các $a_i$ phải dương, và tổng của chúng bằng 1 - Hàm $softmax$ được định nghĩa:

$$a_i = softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \leq i \leq D \tag{6.11}$$

Xét công thức xác suất có điều kiện sao cho biến thuộc lớp $c$ khi biết $x$:

$$p(c|x) = \frac{exp(\sum_{i=1}^{N} w_i f_i(c, x))}{\sum_{c' \in C} exp(\sum_{i=1}^{N} w_i f_i(c', x))} \tag{6.12}$$

Đầu ra của feed-forward neural network ứng với vector input $x$ được tính:

$$h = \sigma(W.x + b) \tag{6.13}$$

$$z = U.h \tag{6.14}$$

$$y = softmax(z) \tag{6.15}$$

## 6.4 Huấn luyện Neural Network

### 6.4.1 Hàm mất mát (Loss function)

Các mô hình neural network là phân loại có giám sát (supervised classifier), tức là dự đoán đầu ra (outcome) của một dữ liệu mới (new input) dựa trên các cặp (input, outcome) đã biết từ trước. Xác định hàm mất mát (loss function) là một phương pháp đánh giá độ hiệu quả của một thuật toán nào đó trên bộ dữ liệu cho trước. Nếu kết quả dự đoán chênh lệch quá nhiều so với kết quả thực tế, hàm mất mát sẽ là một số rất lớn. Mục tiêu của chúng ta là xác định hàm mất mát và tìm cách giảm sai số output.

Hàm mất mát cho neural network là mean-squared error (MSE) giữa giá trị đúng $y^i$ và output của mạng $\hat{y}^i$.

$$L_{MSE(\hat{y},y)} = \frac{1}{n} \sum_{i=1}^{m} (\hat{y}^i - y^i)^2 \tag{6.16}$$

Giống như cái tên, MSE được tính bằng trung bình của hiệu các bình phương của hiệu giữa giá trị dự đoán và giá trị thực tế. Nó chỉ quan tâm đến độ lớn trung bình của lỗi mặc kệ hướng của chúng. Tuy nhiên, do việc lấy bình phương, các dự đoán khi bị lệch xa khỏi giá trị thực sẽ gây tổn thất khá lớn so với các dự đoán gần đúng. Bù lại MSE rất dễ tính đạo hàm thuận tiện cho việc tính toán.

MSE thường được áp dụng trong các bài toán hồi quy, đối với các bài toán phân loại xác suất, hàm mất mát thường được sử dụng là cross entropy loss. Như ta đã biết, với các bài toán phân loại, output sẽ là kết quả của hàm $softmax$, nó có dạng một vector ở dạng one-hot với chỉ một phần từ bằng 1 tại vị trí tương ứng với lớp đó (tính từ 1), các phần tử còn lại bằng 0. Với mỗi input $x$, đầy ra tương ứng qua mạng là đầu

ra dự đoán, trong khi đầu ra thực sự là một vector ở dạng one-hot. Cross entropy giữa 2 vector phân phối $p$ và $q$ là:

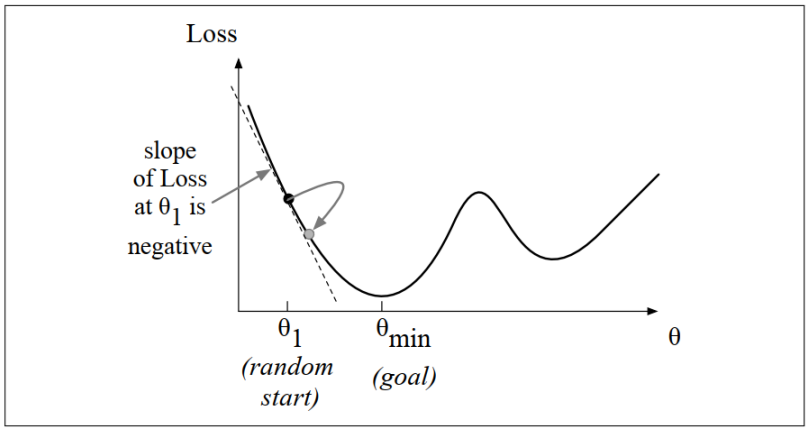
$$H(p,q) = -\sum_{i=1}^{C} p_i \log(q_i) \tag{6.17}$$

Hàm mất mát giữa $\hat{y}$ và $y$ là:

$$L(\hat{y}, y) = -\log p(\hat{y}^i) \tag{6.18}$$

### 6.4.2 Gradient Descent

Để tối ưu các bài toán sử dụng neural network, chúng ta cần tim cách tối thiểu hàm mất mát, một trong những phương pháp thường được áp dụng là gradient descen (GD).




Hình 6.12: Bước đầu tiên trong việc tối ưu hàm mất mát này, bằng cách di chuyển q theo hướng ngược đạo hàm, chúng ta cần di chuyển q theo hướng (+), sang phải.

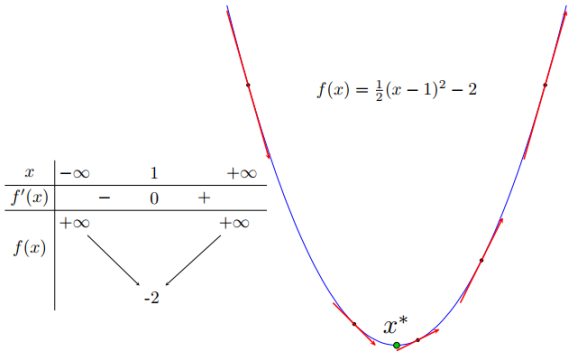
Khởi tạo ngẫu nhiên $\theta = \theta_1$, giả sử hàm mất mát $L$ có đồ thị như trong hình 6.12. Bước đầu tiên trong việc xác định tối thiểu hàm mất mát là di chuyển theo hướng ngược với *độ dốc* của hàm.

Xét trường hợp cơ bản GD với hàm số một biến $f : \mathbb{R} \to \mathbb{R}$. Hình 6.13 mô tả sự biến thiên của hàm số $f(x) = \frac{1}{2}(x-1)^2 - 2$. Ta cần xác định thuật toán để đưa $x_t$ về càng gần $x^*$ càng tốt. Có 2 nhận xét sau đây:

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

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

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




Hình 6.13: Khảo sát sự biến thiên của đa thức bậc 2

- $x_t$ càng xa $x^*$ về phía bên phải thì $f'(x_t)$ càng lớn hơn 0 và ngược lại. Vậy lượng di chuyển $\Delta$ tỷ lệ thuận với $-f'(x_t)$.

2 nhận xét phía trên ta có:

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

Trong đó $\eta > 0$ được gọi là tốc độ học (learning rate). Dấu trừ thể hiện việc chúng ta phải đi ngược với đạo hàm.

Tiếp theo, chúng ta xét GD đối với hàm nhiều biến. Giả sử chúng ta cần tìm giá trị cực tiểu cho hàm $f(\theta)$ trong đó $\theta$ là tập hợp các tham số cần tối ưu. Đạo hàm của hàm số đó tại một điểm $\theta$ bất kỳ được ký hiệu $\nabla_\theta f(\theta)$. Tương tự như hàm một biến, thuật toán GD cho hàm nhiều biến cũng bắt đầu bằng một điểm dự đoán $\theta_0$, sau đó, ở vòng lặp thứ $t$, công thức là:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta f(\theta_t)$$

Hoặc

$$\theta \leftarrow \theta - \eta \nabla_\theta f(\theta)$$

Quay lại với bài toán neural network, hàm mất mát được biểu diễn bởi công thức:

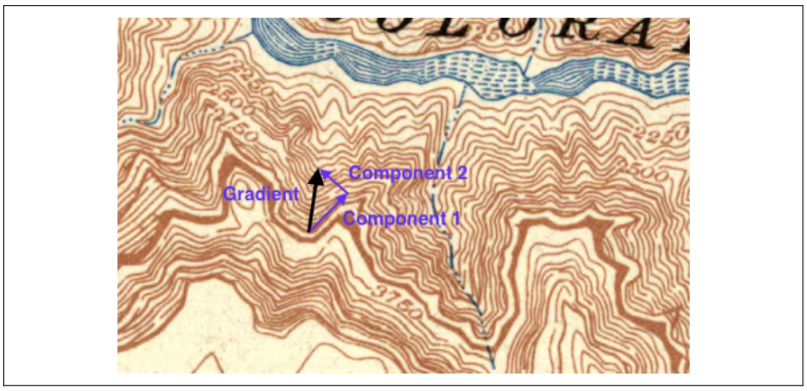
$$L(\hat{y}, y) = -\log p(\hat{y}^i)$$

Trong đó, $\hat{y} = f(x, \theta)$, $x$ là vector input. Khi đó, thuật toán GD áp dụng để tối ưu hàm mất mát được biểu diễn bởi công thức:

$$\nabla_\theta L(f(x,\theta), y) = \begin{bmatrix} \nabla_{\theta_1} L(f(x,\theta), y) \\ \nabla_{\theta_2} L(f(x,\theta), y) \\ \cdot \\ \cdot \\ \cdot \\ \nabla_{\theta_m} L(f(x,\theta), y) \end{bmatrix} \tag{6.19}$$

Hoặc:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(f(x,\theta), y)$$




Hình 6.14: Vector GD trong không gian 2 chiều

### 6.4.3 Backpropagation

Phương pháp phổ biến nhất để tối ưu feed-forward neural network chính là GD. Để áp dụng GD, chúng ta cần tính được đạo hàm của hàm mất mát theo từng ma trận trọng số $W^\ell$ và vector bias $b^\ell$.

Giả sử $L(W, b, X, Y)$ là một hàm mất mát của bài toán, trong đó $W, b$ là tập hợp tất cả các ma trận trọng số giữa các layer và vector bias của mỗi layer. $X, Y$ là cặp dữ liệu huấn luyện với mỗi cột tương ứng với một node dữ liệu. Để áp dụng các phương pháp GD, chúng ta cần tính được các $\nabla_{W^\ell} L$, $\nabla_{b^\ell} L$ $\forall \ell = 1, 2, \ldots, M$.

Nhắc lại quá trình feed-forward:

$$h = \sigma(W.x + b)$$

$$z = U.h$$

$$y = softmax(z) = a$$

Hàm mất mát là hàm MSE:

$$L(W, b, X < Y) = \frac{1}{N} \sum_{i=1}^{N} ||y_i - \hat{y}_i||_2^2 = \frac{1}{N} \sum_{i=1}^{N} ||y_i - a^M||_2^2$$

Với $N$ là cặp dữ liệu $(x, y)$ trong tập huấn luyện. Theo công thức này, việc tính toán trực tiếp các giá trị đạo hàm là cực kỳ phức tạp vì hàm mất mát không phụ thuộc trực tiếp và các ma trận trọng số và vector bias. Phương pháp phổ biến nhất được dùng là backpropagation giúp tính đáo hàm ngược từ layer cuối cùng đến layer đầu tiên. Layer cuối cùng được tính toán trước vì nó gần gũi hơn với output dự đoán và hàm mất mát. Việc tính toán đạo hàm của các ma trận trọng số trong các lớp layer trước được thực hiện dựa trên quy tắc đạo hàm của hàm hợp.

### 6.4.4 Stochstic Gradient Descent (SGD)

Trong SGD, tại một thời điểm (vòng lặp), ta chỉ tính đạo hàm của hàm mất mát dựa trên chỉ một điểm dữ liệu $x_i$ rồi cập nhật $\theta$ dựa trên đạo hàm này. Chú ý rằng

hàm mất mát thường được lấy trung bình trên mỗi điểm dữ liệu nên đạo hàm tại một điểm cũng được kỳ vòng là khá gần với đạo hàm của hàm mất mát trên mọi điểm dữ liệu. Sau khi duyệt qua tất cả các điểm dữ liệu, thuật toán lặp lại quá trình trên. Biển thể đơn giản này trên thực tế làm việc rất hiệu quả.

**function** STOCHASTIC GRADIENT DESCENT($L()$, $f()$, $x$, $y$) **returns** $\theta$
  # where: L is the loss function
  # f is a function parameterized by $\theta$
  # x is the set of training inputs $x^{(1)}$, $x^{(2)}$,..., $x^{(n)}$
  # y is the set of training outputs (labels) $y^{(1)}$, $y^{(2)}$,..., $y^{(n)}$

$\theta \leftarrow$ small random values
**while** not done
  Sample a training tuple $(x^{(i)}, y^{(i)})$
  Compute the loss $L(f(x^{(i)};\theta), y^{(i)})$  # How far off is $f(x^{(i)})$ from $y^{(i)}$?
  $g \leftarrow \nabla_\theta L(f(x^{(i)};\theta), y^{(i)})$  # How should we move $\theta$ to maximize loss ?
  $\theta \leftarrow \theta - \eta_k\, g$  # go the other way instead

Hình 6.15: Thuật toán SGD

Mỗi lần duyệt qua một lượt tất cả các điểm trên toàn bộ dữ liệu được gọi là một epoch. Với GD thông thường, mỗi epoch ứng với một lần cập nhật $\theta$. Với SGD, 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 một epoch. Mặt khác, với SGD, nghiệm có thể hội tụ sau vài epoch. Vì vậy, SGD phù hợp với các bài toán có lượng cơ sở dữ liệu lớn và các bài toán yêu cầu mô hình thay đổi liên tục như online learning. Với một mô hình đã được huấn luyện từ trước, khi có thêm điểm dữ liệu, ta chỉ cần chạy thêm một vài epoch nữa có thể có nghiệm hội tụ.

Một điểm cần lưu ý là sau mỗi epoch, chúng ta cần xáo trộn thứ tự của các dữ liệu để đảm bảo tính ngẫu nhiên. Việc này cũng ảnh hưởng tới hiệu năng của SGD. Đây cũng là lý do thuật toán này có chứa từ stochtis. Một cách toán học, công thức của mỗi lần cập nhật của SGD là:

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta, x_i, y_i)$$

Trong đó $L(\theta, x_i, y_i) = L_i(\theta)$ là hàm mất mát với chỉ một điểm dữ liệu thứ $i$.

## 6.5 Neural Language Models

Trong phần này, chúng ta giới thiệu một bài toán ứng dụng của neural network: dự đoán các từ tiếp theo trong câu dựa vào những từ đã cho trước. Mặc dù chúng ta đã giới thiệu một mô hình ngôn ngữ rất hữu ích là N-grams (Chương 2), các mô hình

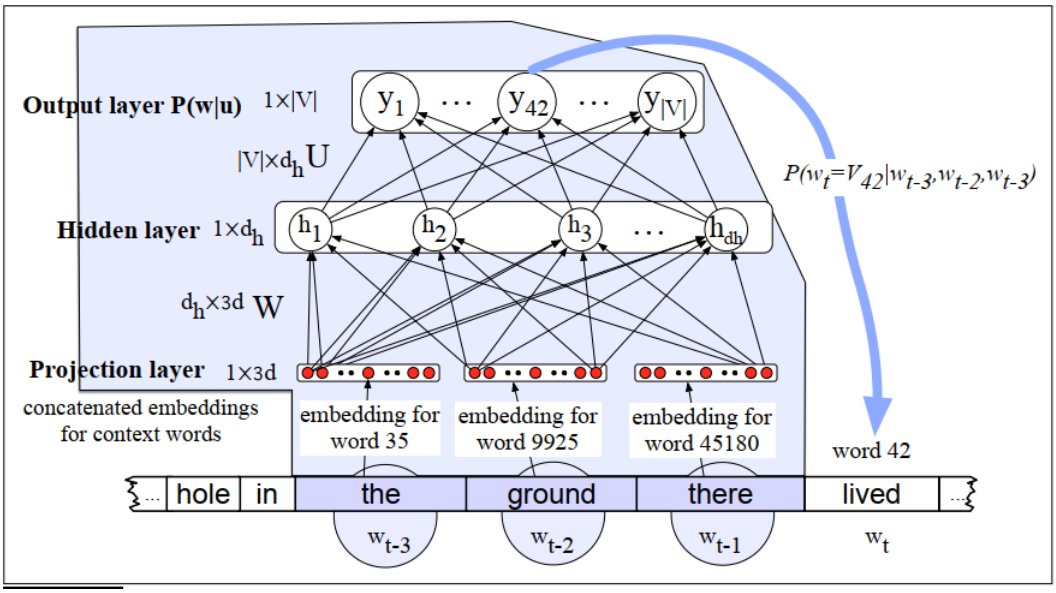
ngôn ngữ dựa trên neural network cũng có nhiều ưu điểm hơn. Feed-forward neural LM là một mạng feed-forward tiêu chuẩn, input tại thời điểm $t$ là các đại diện cho các từ trước đó $(w_{t-1}, w_{t-2}, \dots)$, output là một phân phối xác suất trên các từ tiếp theo của chuỗi. Như vậy, LM xấp xỉ xác suất có điều kiện của một từ dựa trên các từ trước đó $P(w_t|w_1^{t-1})$ bằng công thức:

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1}) \tag{6.20}$$

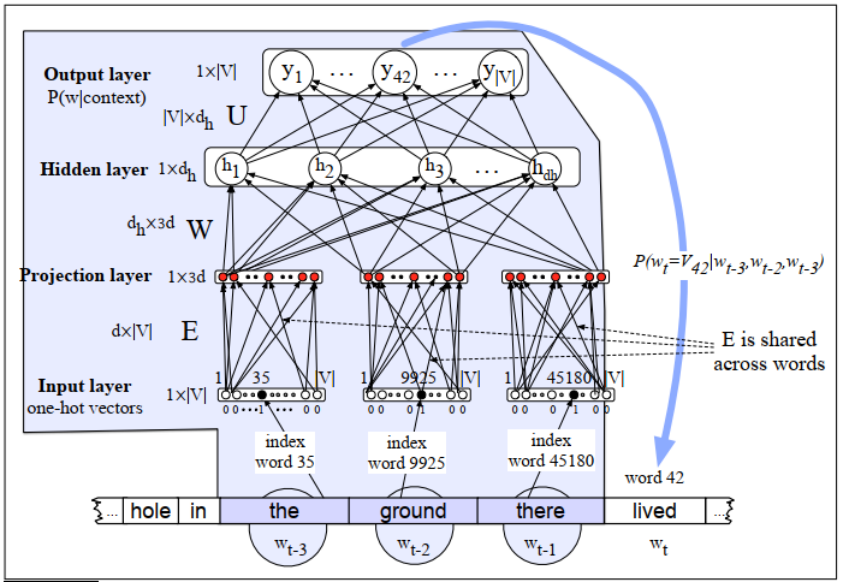
Trong các ví dụ dưới đây, chúng ta sử dụng 4-grams, vì vậy chúng ta sẽ hiển thị mạng biểu diễn ước lượng xác suất $P(w_t = i|w_{t-1}, w_{t-2}), w_{t-3}$.

### 6.5.1 Embeddings

Mỗi từ được biểu diễn dưới dạng vector có kích thước $d$, việc các từ hoặc cụm từ được ánh xạ sang các vector số được gọi là embedding, thương được gọi là vector space model. Hình 6.16 cho thấy một FFNNLM đơn giản với $N = 3$, tại thời điểm $t$, mỗi từ trong 3 từ trước đó được biểu diễn dưới dạng one-hot vecto. 3 vector này nối với nhau để tạo ra $x$, output layer có đầu ra là một softmax với xác suất phân phối trên mỗi từ. Do đó, từ $y_{42}$, giá trị của output node 42 là xác suất của từ tiếp theo $w_t$ là $V_{42}$, đồng thời đánh số từ vựng là 42.




Hình 6.16: Cách hoạt động của feed-forward neurral language model đối với một đoạn văn bản. Tại mỗi thời điểm $t$, mạng có 3 từ ngữ cảnh, chuyển đổi từng từ thành vector $d$ chiều, kết nối 3 vector tạo thành input unit có kích thước $1 \times Nd$. Mỗi unit thực hiện việc tính ma trận trọng số và đi qua hàm kích hoạt để tạo lớp ẩn $h$. Cuối cùng, output layer là một softmax dự đoán tại mỗi nút $i$, xác suất để từ kế tiếp $w_t$ là một từ vựng $V_i$.




Hình 6.17: Neural Language Models

Các bước được biểu diễn trong hình 6.17:

- Chọn 3 từ embedding từ $E$
- Nhân với ma trận trọng số $W$
- Nhận với $U$
- Sử dụng hàm softmax

Tóm lại, phương trình cho neural language model:

$$e = (Ex_1, Ex_2, \ldots, Ex) \tag{6.21}$$

$$h = \sigma(W.e + b) \tag{6.22}$$

$$z = U.h \tag{6.23}$$

$$y = softmax(z) \tag{6.24}$$

### 6.5.2 Huấn luyện neural language model

Đặt $\theta = E, W, U, b$, sử dụng thuật toán SGD (hình 6.15) để tối ưu hàm mất mát. Với mỗi từ $w_t$, cross-entropy:

$$L = -\log p(w_t|w_{t-1}, w_{t-2}, \ldots, w_{t-n+1})$$

Công thức SGD:

$$\theta_{t+1} = \theta_t - \eta \frac{\partial \log p(w_t|w_{t-1}, w_{t-2}, \ldots, w_{t-n+1})}{\partial \theta}$$