JVnTagger: Công cụ gán nhãn từ loại tiếng Việt dựa trên Conditional Random Fields và Maximum Entropy

Sản phẩm nhánh 8.3

Phụ trách: Phan Xuân Hiếu

Mục lục

1 Giới thiệu

JVnTagger là công cụ gán nhãn từ loại tiếng Việt dựa trên Conditional Random Fields (Lafferty et al., 2001) và Maximum Entropy (Nigam et al., 1999). JVnTagger được xây dựng trong khuôn khổ đề tài cấp nhà nước VLSP với dữ liệu huấn luyện khoảng 10.000 câu và 20,000 câu của Viet Treebank. Thử nghiệm với phương pháp 5-fold cross validation trên VTB-10,000 cho thấy kết quả gán nhãn với CRFs có thể đạt giá trị F1 lớn nhất là 93.45% và 10-fold cross validation với Maxent trên VTB-20,000 đạt giá trị F1 lớn nhất là 93.32%.

2 Cơ sở lý thuyết và thực nghiệm

2.1 Giới thiệu Maximum Entropy

Tư tưởng chính của Maximum Entropy là *"ngoài việc thỏa mãn một số rang buộc nào đó thì mô hình càng đồng đều càng tốt"*. Để rõ hơn về vấn đề này, ta hãy cùng xem xét bài toán phân lớp gồm có 4 lớp. Ràng buộc duy nhất mà chúng ta chỉ biết là trung bình 40% các tài liệu chứa từ "professor" thì nằm trong lớp ***faculty***. Trực quan cho thấy nếu có một tài liệu chứa từ "professor" chúng ta có thể nói có 40% khả năng tài liệu này thuộc lớp ***faculty***, và 20% khả năng cho các khả năng còn lại (thuộc một trong 3 lớp còn lại).

Mặc dù maximum entropy có thể được dùng để ượng lượng bất kì một phân phối xác suất nào, chúng ta xem xét khả năng maximum entropy cho việc gán nhãn dữ liệu chuỗi. Nói cách khác, ta tập trung vào việc học ra phân phối điều kiện của chuỗi nhãn tương ứng với chuỗi (xâu) đầu vào cho trước.

2.1.1 Các Ràng buộc và Đặc trưng

Trong maximum entropy, người ta dùng dữ liệu huấn luyện để xác định các ràng buộc trên phân phối điều kiện. Mỗi ràng buộc thể hiện một đặc trưng nào đó của dữ liệu huấn luyện. Mọi hàm thực trên quan sát đầu vào và nhãn đầu ra có thể được xem như là đặc trưng $f_i(o,s)$. Maximum Entropy cho phép chúng ta giới hạn các phân phối mô hình lý thuyết gần giống nhất các giá trị kì vọng cho các đặc trưng này trong dữ liệu huấn luyện D. Vì thế người ta đã mô hình hóa xác suất $P(o \mid s)$ như sau (ở đây, o là quan sát đầu vào và s là quan sát đầu ra)

$$P(o \mid s) = \frac{1}{Z(o)} \exp\left(\sum_i \lambda_i f_i(o,s) \right) \qquad (2.1)$$

Ở đây $f_i(o,s)$ là một đặc trưng, λ_i là một tham số cần phải ướng lượng và $Z(o)$ là thừa số chuẩn hóa đơn giản nhằm đảm bảo tính đúng đắn của định nghĩa xác suất (tổng xác suất trên toàn bộ không gian bằng 1) $Z(o) = \sum_c \exp \sum_c \lambda_i f_i(o,s)$.

Lưu ý, mỗi hàm đặc trưng $f_i(o,s)$ là một ánh xạ từ <ngữ cảnh, nhãn> →[0,1]. Một ví dụ về một hàm đặc trưng là f(từ hiện tại là "học_sinh", nhãn danh từ N) = 1.

Một số phương pháp huấn luyện mô hình từ dữ liệu học bao gồm: IIS (improved iterative scaling), GIS, L-BFGS, v.v.

2.2 Giới thiệu Conditional Random Fields

CRFs là mô hình trạng thái tuyến tính vô hướng (máy trạng thái hữu hạn được huấn luyện có điều kiên) và tuân theo tính chất Markov thứ nhất. CRFs đã được chứng minh rất thành công cho các bài toán gán nhãn cho chuỗi như tách từ, gán nhãn cụm từ, xác định thực thể, gán nhãn cụm danh từ, etc.

Gọi $\mathbf{o} = (\mathbf{o_1}, \mathbf{o_2}, \ldots, \mathbf{o_T})$ là một chuỗi dữ liệu quan sát cần được gán nhãn. Gọi S là tập trạng thái, mỗi trạng thái liên kết với một nhãn $l \in L$. Đặt $\mathbf{s} = (\mathbf{s_1}, \mathbf{s_2}, \ldots, \mathbf{s_T})$ là một chuỗi trạng thái nào đó, CRFs xác định xác suất điều kiện của một chuỗi trạng thái khi biết chuỗi quan sát như sau:

$$p_\theta(\mathbf{s} \mid \mathbf{o}) = \frac{1}{Z(\mathbf{o})} \exp\left[\sum_{t=1}^{T}\sum_{k}\lambda_k f_k(s_{t-1}, s_t, \mathbf{o}, t)\right]. \quad (1)$$

Gọi $Z(\mathbf{o}) = \sum_{s'} \exp\left(\sum_{t=1}^{T}\sum_{k}\lambda_k f_k(s'_{t-1}, s'_t, \mathbf{o}, t)\right)$ là thừa số chuẩn hóa trên toàn bộ các chuỗi nhãn có thể. f_k xác định một hàm đặc trưng và λ_k là trọng số liên kết với mỗi đặc trưng f_k. Mục đích của việc học máy với CRFs là ước lượng các trọng số này. Ở đây, ta có hai loại đặc trưng f_k : đặc trưng trạng thái (per-state) và đặc trưng chuyển (transition).

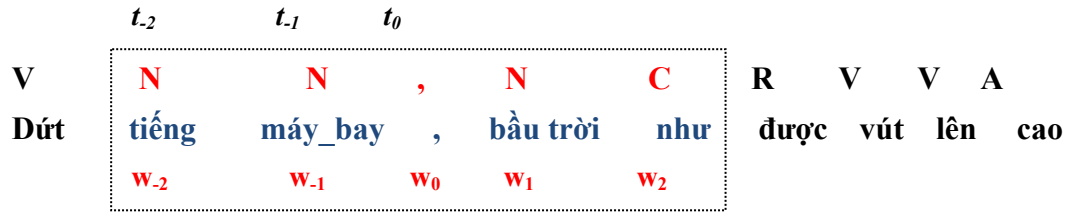
$$f_k^{(per-state)}(s_t, \mathbf{o}, t) = \delta(s_t, l)x_k(\mathbf{o}, t). \quad (2)$$

$$f_k^{(transition)}(s_{t-1}, s_t, t) = \delta(s_{t-1}, l)\delta(s_t, l). \quad (3)$$

Ở đây δ là Kronecker-δ . Mỗi đặc trưng trạng thái (2) kết hợp nhãn l của trạng thái hiện tại s_t và một vị từ ngữ cảnh - một hàm nhị phân $x_k(\mathbf{o},t)$ xác định các ngữ cảnh quan trọng của quan sát $\mathbf{o}$ tại vị trí t. Một đặc trưng chuyển (3) biểu diễn sự phụ thuộc chuỗi bằng cách kết hợp nhãn l' của trạng thái trước s_{t-1} và nhãn l của trạng thái hiện tại s_t.

Người ta thường huấn luyện CRFs bằng cách làm cực đại hóa hàm likelihood theo dữ liệu huấn luyện sử dụng các kĩ thuật tối ưu như L-BFGS. Việc lập luận (dựa trên mô hình đã học) là tìm ra chuỗi nhãn tương ứng của một chuỗi quan sát đầu vào. Đối với CRFs, người ta thường sử dụng thuật toán qui hoạch động điển hình là Viterbi để thực hiện lập luận với dữ liệu mới.

2.3 Lựa chọn đặc trưng



Hình 1. Cửa sổ trượt với kích cỡ size = 5 chuyển động dọc theo dữ liệu

Các mẫu ngữ cảnh cho việc lựa chọn đặc trưng với Maximum Entropy và Conditional Random Fields được cho trong bảng sau:

Loại	Ngữ cảnh	Giải thích
Mẫu ngữ cảnh cho cả Maxent và CRFs		
Mẫu ngữ cảnh cơ bản (loại 1)	w:-2; w:-1; w:0; w:1; w:2	w:i cho biết từ tại vị trí thứ i trong chuỗi đầu vào (nằm trong cửa sổ trượt với kích cỡ 5)
	w:0:1; w:1:2; w:-1:1	w:i:j kết hợp từ thứ i và từ thứ j trong chuỗi đầu vào
	is_all_capitalized(i) (i=0;1); is_initial_capitalized(i) (i=0;1); is_number(i) (i=-1;0;1); contain_numbers(i) (i, contain_hyphen, contain_comma, is_marks	Kiểm tra một số thuộc tính của từ thứ i trong cửa sổ hiện tại như: từ có phải là toàn chữ viết hoa hay có kí tự đầu viết hoa hay không, có chứa số, v.v...
Mẫu ngữ cảnh từ điển (loại 2)	tags_in_dictionary(i) (i=0,1)	Các từ loại có thể gán cho từ thứ i trong cửa sổ hiện tại (V, N, A, ...)
Mẫu ngữ cảnh đặc trưng tiếng Việt (loại 3)	is_full_repretative(0), is_partial_repretative(0)	Kiểm tra xem một từ có phải từ láy toàn bộ hay một phần không
Mẫu ngữ cảnh dựa vào suffix (loại 4)	prf(0), sff(0)	Âm tiết đầu tiên (ví dụ "sự" trong "sự hướng dẫn"), cuối cùng trong từ hiện tại ("hóa" trong "công nghiệp hóa")
Mẫu cho đặc trưng cạnh của CRFs		
t_{-1} t_0	Nhãn của từ trước đó và nhãn của từ hiện tại. Đặc trưng này được trích chọn trực tiếp từ dữ liệu bởi FlexCrfs	

2.4 Kết quả gán nhãn từ loại với CRFs và Maximum Entropy

2.4.1 Kết quả gán nhãn từ loại với CRFs và Maxent trên tập VTB-10.000

Dữ liệu VietTreebank gồm 10,000 câu được chia thành 5 folds. Đánh giá gán nhãn từ loại với CRFs và Maximum Entropy với phương pháp 5-fold-cross-validation, lấy lần lượt 4 fold để huấn luyện và thử

nghiệm trên fold còn lại sau đó lấy trung bình độ đo F1 trên 5 thử nghiệm, chúng tôi thu được kết quả như bảng sau:

Fold	F1-measure (CRFs)	F1-measure (Maxent)
Fold1	93.01%	93.18%
Fold2	93.33%	**93.26%**
Fold3	**93.46%**	93.25%
Fold4	93.15%	93.09%
Fold5	92.90%	93.11%
Trung bình	**93.17%**	**93.18%**

2.4.2 Kết quả gán nhãn từ loại với Maxent trên tập dữ liệu VTB-20,000

Dữ liệu VietTreebank gồm 20,000 câu được chia thành 10 folds. Đánh giá gán nhãn từ loại với Maximum Entropy với phương pháp 10-fold-cross-validation, lấy lần lượt 9 fold để huấn luyện và thử nghiệm trên fold còn lại sau đó lấy trung bình độ đo F1 trên 10 thử nghiệm, chúng tôi thu được kết quả như bảng sau:

Fold	F1-measure (Maxent)	Fold	F1-measure (Maxent)
Fold1	93.02%	Fold6	93.21%
Fold2	93.11%	Fold7	93.13%
Fold3	93.21%	Fold8	**93.32%**
Fold4	93.01%	Fold9	93.00%
Fold5	92.05%	Fold10	93.16%
Trung bình	**93.12%**		

3 Mô tả JVnTagger

Công cụ được cài đặt trên ngôn ngữ Java (phiên bản 1.6). Để có thể thực thi được công cụ, chúng ta chỉ cần cài đặt Java Runtime Environment.

3.1 Cấu trúc thư mục của JVnTagger:

bin (lưu các file .classes đã được biên dịch)

inputdir (lưu các file văn bản để thử nghiệm gán nhãn từ loại với JVnTagger)

lib (lưu các thư viện cần dùng cho JVnTagger)

lbfgs.jar (thư viện cần dung để tối ưu hóa hàm likelihood cho CRFs và Maxent)

model	(thư mục lưu các mô hình đã được huấn luyện của CRFs và Maxent. Mô hình của CRFs được sinh ra nhờ huấn luyện với công cụ FlexCRFs++. Mô hình của Maxent được sinh ra nhờ huấn luyện dùng jmaxent.Trainer trong JVnTagger).
src	(thư mục lưu mã nguồn của công cụ)

3.2 Các packages trong JVnTagger

Packages	Mô tả
jflexcrfs	Lưu mã nguồn CRFs cho gán nhãn dữ liệu với mô hình đã được huấn luyện lưu trong model/crfs. Lưu ý định dạng mô hình phù hợp với định dạng mô tả trong FlexCRFs++[1]. Xem thêm phần 3 để biết thêm cơ sở lý thuyết của CRFs.
*flexcrfs.**Labeling***	*Gán nhãn câu với CRFs*
jmaxent	Lưu mã nguồn Maximum Entropy. Xem thêm phần 3 để biết thêm cơ sở lý thuyết của Maxent.
*jmaxent.**Trainer***	*Huấn luyện mô hình Maximum Entropy*
*jmaxent.**Classification***	*Phân lớp với Maximum Entropy*
jvntagger.data	(mã nguồn cho phép thao tác, xử lý dữ liệu)
*jvntagger.data .**TWord***	*Lưu từ vựng và nhãn từ loại tương ứng. Nhãn có thể nhận giá trị **null** (trong trường hợp từ chưa được gán nhãn)*
*jvntagger.data .**Sentence***	*Một tập các từ vựng cùng nhãn tương ứng (tập các TWord)*
*jvntagger.data .**DataReader***	*Lớp trừu tượng, thừa kết lớp này để đọc dữ liệu vào với các dữ liệu với định dạng khác nhau.*
*jvntagger.data .**DataWriter***	*Lớp trừu tượng, thừa kế lớp này để lưu dữ liệu đầu ra với các định dạng khác nhau.*
*jvntagger.data.**ContextGenerator***	*Lớp trừu tượng, thừa kế lớp này để thực hiện các chiến lược trích chọn các thông tin ngữ cảnh từ dữ liệu khác nhau.*
*jvntagger.data .**TaggingData***	*Lựa chọn đặc trưng từ dữ liệu theo một ContextGenerator xác định.*
jvntagger	Package chính cho gán nhãn từ vựng tiếng Việt

[1] http://flexcrfs.sourceforge.net/

*jvntagger.**BasicContextGenerator***	*Thực thi **ContextGenerator** với một số đặc trưng cơ bản.*
*Jvntagger.**POSContextGenerator***	*Thực thi **ContextGenerator** với các đặc trưng được thiết lập theo một file cấu hình theo định dạng XML.*
*jvntagger.**POSTagger***	*Interface định nghĩa các hàm cơ bản cho một bộ gán nhãn từ loại*
*jvntagger.**POSDataReader***	*Đọc dữ liệu đã được tách từ theo định dạng trong đó mỗi câu được lưu trên một dòng.*
*jvntagger.**POSDataWriter***	*Ghi dữ liệu sau khi gán nhãn dưới định dạng trong đó mỗi câu trên một dòng và các từ trong câu được gán thêm nhãn từ loại. Ví dụ "Mãi_mãi/R tuổi/N 20/M ./." là một câu sau khi gán nhãn từ loại.*
*jvntagger.**CRFTagger***	*Thực thi POSTagger với CRFs*
*jvntagger.**MaxentTagger***	*Thực thi POSTagger với phương pháp Maximum Entropy.*
*jvntagger.**POSTagging***	*Giao diện dòng lệnh cho gán nhãn từ loại tiếng Việt.*
jvntagger.service	Package cung cấp dịch vụ gán nhãn từ loại qua socket 2929
*jvntagger.service.**TaggingService***	*Phía server: Mở socket lắng nghe yêu cầu gán nhãn từ loại tại cổng 2929. Khi có một yêu cầu đến, TaggingService mở một luồng mới thực hiện giao tiếp với client theo một cổng khác và quay về tiếp tục lắng nghe trên cổng 2929.*
*jvntagger.service.**Session***	*(Phía server) Nhận dữ liệu từ client theo định dạng UTF-8 (luồng dữ liệu kết thúc bởi "0"), thực hiện việc gán nhãn từ loại; trả lại dữ liệu cho client theo định dạng UTF-8*
*jvntagger.service.**TaggingClient***	*Phía client: mở kết nối đến server, gửi dữ liệu định dạng UTF-8 (luồng dữ liệu kết thúc bởi "0"), nhận dữ liệu từ phía server trả về.*

3.3 Gọi JVnTagger từ dòng lệnh

Câu lệnh:

java –mx512M –cp [classpath] jvntagger.POSTagging -tagger [tagger] –modeldir [model dir] –inputfile/-inputdir [input file/input dir]

Nếu đang trong thư mục ngoài cùng của JVnTagger, chúng ta có thể thiết lập các tùy chọn để phân loại từ vựng như sau:

[classpath] = bin:lib\lbfgs.jar (hoặc bin;lib\lbfgs.jar nếu ở trong Windows)

[tagger] = crfs hoặc maxent

[model dir] = thư mục chứa mô hình của crfs hoặc maxent trong thư mục model

[inputfile/inputdir] : đường dẫn đến file (thư mục) cần xử lý nếu chọn tùy chọn phân loại từ cho file –inputfile (cho thư mục –inputdir)

Ví dụ: Chúng ta có thể thực hiện gán nhãn từ vựng cho các file trong thư mục **inputdir** theo câu lệnh như sau. (giả sử chúng ta đang trong thư mục ngoài cùng của JVnTagger.

Java –mx512M –cp bin:lib/lbfgs.jar jvntagger.POSTagging –tagger maxent –modeldir model/maxent –inputdir samples/inputdir

3.4 Lập trình với JVnTagger

JVnTagger có thể được dùng để tích hợp vào một hệ thống lớn hơn. JVnTagger nhận vào một chuỗi dữ liệu và trả về một chuỗi đã được gán nhãn. Để gán nhãn từ loại cho một chuỗi đầu vào, trước hết chúng ta cần khai báo một đối tượng của lớp POSTagger và khởi tạo nó với 1 trong hai bộ gán nhãn CRFTagger hoặc MaxentTagger. Lưu ý, việc khởi tạo một đối tượng như vậy mất một khoảng thời gian để tải mô hình vào bộ nhớ, vì thế chúng ta nên khởi tạo một đối tượng duy nhất một lần và dùng nó cho nhiều xử lý sau này.

```
modelDir = "model\maxent"

POSTagger tagger = null;

tagger = new MaxentTagger(modelDir);
```

Gán nhãn với JVnTagger: Đầu vào của JVnTagger là một các câu đã được tách từ (các âm tiết trong mỗi từ được gán với nhau bởi dấu gạch dưới "_"). Đầu ra của JVnTagger là câu được gán nhãn từ loại.

```
String inputStr = "Mãi_mãi tuổi 20."

String resultStr = tagger.tagging(inputStr);
```

Kết quả phân loại từ vựng tiếng Việt với JVnTagger cho ra xâu kết quả (`resultStr`) là

```
"“Mãi_mãi/R tuổi/N 20/M ./."
```

Lớp POSTagging trong package jvntagger là một ví dụ về lập trình với JVnTagger.

3.5 Kết nối đến dịch vụ gán nhãn từ loại

Mở kết nối đến server cung cấp dịch vụ gán nhãn từ loại tại cổng mặc định là 2929. Gửi dữ liệu theo định dạng UTF-8 kết thúc bởi kí tự "0", dữ liệu gửi lại cũng theo định dạng UTF-8 và kết thúc bởi "0".

Khởi động dịch vụ gán nhãn từ loại:

java –mx512M –cp [classpath] jvntagger.service.TaggingService [model dir]

[model dir] = Thư mục chứa mô hình của crfs hoặc maxent trong thư mục model. Nếu thư mục có tên là "crfs" (maxent), mô hình dùng để gán nhãn là CRFs (maxent).

Ví dụ về kết nối tới dịch vụ gán nhãn từ loại:

Mở kết nối:

```
public boolean connect(){
    try {
        sock = new Socket(host, port);
        in = new BufferedReader(new InputStreamReader(
                sock.getInputStream(), "UTF-8"));
        out = new BufferedWriter(new OutputStreamWriter(
                sock.getOutputStream(), "UTF-8"));
        return true;
    }
    catch (Exception e){
        System.out.println(e.getMessage());
        return false;
    }
}
```

Gửi và nhận dữ liệu tới/từ server

```
public String process(String data){
    try {
        out.write(data);
        out.write((char)0);
        out.flush();

        //Get data from server
        String tagged = "";
        while (true){
            int ch = in.read();

            if (ch == 0) break;
            tagged += (char) ch;
        }
        return tagged;
    }
    catch (Exception e){
        System.out.println(e.getMessage());
        return "";
    }

}
```

Đóng kết nối:

```
public void close(){
    try {
        this.sock.close();
    }
    catch (Exception e){
        System.out.println(e.getMessage());
    }
}
```