

Natural Language Processing Sequence Labeling and Hidden Markov Models

Felipe Bravo-Marquez

June 23, 2023

Overview

- The Sequence Labeling (or Tagging) Problem
- Generative models, and the noisy-channel model, for supervised learning
- Hidden Markov Model (HMM) taggers
 - Basic definitions
 - Parameter estimation
 - The Viterbi algorithm

This slides are based on the course material by Michael Collins: <http://www.cs.columbia.edu/~mcollins/cs4705-spring2019/slides/tagging.pdf>

Sequence Labeling or Tagging Tasks

- Sequence Labeling or Tagging is a task in NLP different from document classification.
- Here the goal is to map a sentence represented as a sequence of tokens x_1, x_2, \dots, x_n into a sequence of tags or labels y_1, y_2, \dots, y_n .
- Well known examples of this task are Part-of-Speech (POS) tagging and Named Entity Recognition (NER) to be presented next.

Part-of-Speech Tagging

INPUT: Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

OUTPUT: Profits/**N** soared/**V** at/**P** Boeing/**N** Co./**N** ,/, easily/**ADV** topping/**V** forecasts/**N** on/**P** Wall/**N** Street/**N** ,/, as/**P** their/**POSS** CEO/**N** Alan/**N** Mulally/**N** announced/**V** first/**ADJ** quarter/**N** results/**N** ./.

- **N** = Noun
- **V** = Verb
- **P** = Preposition
- **Adv** = Adverb
- **Adj** = Adjective
- ...

Named Entity Recognition

INPUT: Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

OUTPUT: Profits soared at [Company Boeing Co.], easily topping forecasts on [Location Wall Street], as their CEO [Person Alan Mulally] announced first quarter results.

Named Entity Extraction as Sequence Labeling

INPUT: Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

OUTPUT: Profits/NA soared/NA at/NA Boeing/SC Co./CC ,/NA easily/NA topping/NA forecasts/NA on/NA Wall/SL Street/CL ,/NA as/NA their/NA CEO/NA Alan/SP Mulally/CP announced/NA first/NA quarter/NA results/NA ./NA

- NA = No entity
- SC = Start Company
- CC = Continue Company
- SL = Start Location
- CL = Continue Location
- SP = Start Person
- CP = Continue Person

Our Goal

Training set:

1. Pierre/**NNP** Vinken/**NNP** ,/, 61/**CD** years/**NNS** old/**JJ** ,/, will/**MD** join/**VB** the/**DT** board/**NN** as/**IN** a/**DT** nonexecutive/**JJ** director/**NN** Nov./**NNP** 29/**CD** ./.
2. Mr./**NNP** Vinken/**NNP** is/**VBZ** chairman/**NN** of/**IN** Elsevier/**NNP** N.V./**NNP** ,/, the/**DT** Dutch/**NNP** publishing/**VBG** group/**NN** ./.
3. Rudolph/**NNP** Agnew/**NNP** ,/, 55/**CD** years/**NNS** old/**JJ** and/**CC** chairman/**NN** of/**IN** Consolidated/**NNP** Gold/**NNP** Fields/**NNP** PLC/**NNP** ,/, was/**VBD** named/**VBN** a/**DT** nonexecutive/**JJ** director/**NN** of/**IN** this/**DT** British/**JJ** industrial/**JJ** conglomerate/**NN** ./.
4. ...

Our Goal: From the training set, induce a function/algorithm that maps new sentences to their tag sequences.

Two Types of Constraints

Influential/JJ members/NNS of/IN the/DT House/NNP
Ways/NNP and/CC Means/NNP Committee/NNP
introduced/VBD legislation/NN that/WDT would/MD restrict/VB
how/WRB the/DT new/JJ savings-and-loan/NN bailout/NN
agency/NN can/MD raise/VB capital/NN ./.

“Local”:

- e.g., “can” is more likely to be a modal verb MD rather than a noun NN

“Contextual”:

- e.g., a noun is much more likely than a verb to follow a determiner

Sometimes these preferences are in conflict:

- The trash can is in the garage

Sequence Labeling as Supervised Learning

- We have a sequence of inputs $x = (x_1, x_2, \dots, x_n)$ and corresponding labels $y = (y_1, y_2, \dots, y_n)$.
- Task is to learn a function f that maps input sequences to label sequences: $f(x_1, x_2, \dots, x_n) = y_1, y_2, \dots, y_n$.
- We have a training set of labeled sequences:
 $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$.

Generative Approach for Sequence Labeling

- Generative models such as Naive Bayes was used for classification can also be used for sequence labeling tasks in NLP.
- Approach:
 - Training: Learn the joint distribution $p(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n)$ of input sequences.
 - Decoding: Use the learned distribution to predict label sequences for new input sequences.
- Decoding in sequence labeling involves finding the label sequence with the highest joint probability: $\arg \max_{y_1, y_2, \dots, y_n} p(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n)$.

Hidden Markov Models

- Hidden Markov Models (HMMs) provide a principled way to handle sequence labeling problems using generative modeling and efficient decoding algorithms.
- We have an input sentence $x = x_1, x_2, \dots, x_n$ (x_i is the i -th word in the sentence).
- We have a tag sequence $y = y_1, y_2, \dots, y_n$ (y_i is the i -th tag in the sentence).
- We'll use an HMM to define $p(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n)$ for any sentence x_1, \dots, x_n and tag sequence y_1, \dots, y_n of the same length. [Kupiec, 1992]
- Then, the most likely tag sequence for x is:

$$\arg \max_{y_1, \dots, y_n} p(x_1, \dots, x_n, y_1, \dots, y_n)$$

Trigram Hidden Markov Models (Trigram HMMs)

For any sentence x_1, \dots, x_n where $x_i \in V$ for $i = 1, \dots, n$, and any tag sequence y_1, \dots, y_{n+1} where $y_i \in S$ for $i = 1, \dots, n$, and $y_{n+1} = \text{STOP}$, the joint probability of the sentence and tag sequence is:

$$p(x_1, \dots, x_n, y_1, \dots, y_{n+1}) = \prod_{i=1}^{n+1} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^n e(x_i | y_i)$$

where we have assumed that $x_0 = x_{-1} = *$.

Parameters of the Model

- $q(s|u, v)$ for any $s \in S \cup \{\text{STOP}\}$, $u, v \in S \cup \{*\}$
- $e(x|s)$ for any $s \in S$, $x \in V$

An Example

If we have $n = 3$, x_1, x_2, x_3 equal to the sentence "the dog laughs", and y_1, y_2, y_3, y_4 equal to the tag sequence "D N V STOP", then:

$$\begin{aligned} p(x_1, \dots, x_n, y_1, \dots, y_{n+1}) = & q(D|*, *) \times q(N|*, D) \\ & \times q(V|D, N) \times q(\text{STOP}|N, V) \\ & \times e(\text{the}|D) \times e(\text{dog}|N) \times e(\text{laughs}|V) \end{aligned}$$

- STOP is a special tag that terminates the sequence.
- We take $y_0 = y_{-1} = *$, where $*$ is a special "padding" symbol.

Why the Name?

$$\begin{aligned} p(x_1, \dots, x_n, y_1, \dots, y_n) &= q(\text{STOP} | y_{n-1}, y_n) \\ &\quad \times \prod_{j=1}^n q(y_j | y_{j-2}, y_{j-1}) \\ &\quad \times \prod_{j=1}^n e(x_j | y_j) \end{aligned}$$

- Markov Chain:

$$q(\text{STOP} | y_{n-1}, y_n) \times \prod_{j=1}^n q(y_j | y_{j-2}, y_{j-1})$$

- Observed:

$$e(x_j | y_j)$$

Smoothed Estimation

$$\begin{aligned} q(Vt|DT, JJ) = & \lambda_1 \times \frac{\text{Count}(Dt, JJ, Vt)}{\text{Count}(Dt, JJ)} \\ & + \lambda_2 \times \frac{\text{Count}(JJ, Vt)}{\text{Count}(JJ)} \\ & + \lambda_3 \times \frac{\text{Count}(Vt)}{\text{Count}()} \end{aligned}$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and for all i , $\lambda_i \geq 0$.

$$e(\text{base}|Vt) = \frac{\text{Count}(Vt, \text{base})}{\text{Count}(Vt)}$$

Dealing with Low-Frequency Words

A common method is as follows:

- Step 1: Split vocabulary into two sets
 - Frequent words = words occurring ≥ 5 times in training
 - Low frequency words = all other words
- Step 2: Map low frequency words into a small, finite set, depending on prefixes, suffixes, etc.

Dealing with Low-Frequency Words: An Example

Below is an example of word classes for named entity recognition [Bikel et al., 1999]:

Word class	Example	Intuition
twoDigitNum	90	Two-digit year
fourDigitNum	1990	Four-digit year
containsDigitAndAlpha	A8956 – 67	Product code
containsDigitAndDash	09 – 96	Date
containsDigitAndSlash	11/9/89	Date
containsDigitAndComma	23,000.00	Monetary amount
containsDigitAndPeriod	1.00	Monetary amount, percentage
othernum	456789	Other number
allCaps	BBN	Organization
capPeriod	M.	Person name initial
firstWord	First word of sentence	No useful capitalization information
initCap	Sally	Capitalized word
lowercase	can	Uncapitalized word
other	,	Punctuation marks, all other words

Dealing with Low-Frequency Words: An Example

Original Sentence:

Profits/NA soared/NA at/NA Boeing/SC Co./CC ,/NA easily/NA topping/NA forecasts/NA on/NA Wall/SL Street/CL ,/NA as/NA their/NA CEO/NA Alan/SP Mulally/CP announced/NA first/NA quarter/NA results/NA ./NA

Transformed Sentence:

firstword/NA soared/NA at/NA initCap/SC Co./CC ,/NA easily/NA lowercase/NA forecasts/NA on/NA initCap/SL Street/CL ,/NA as/NA their/NA CEO/NA Alan/SP initCap/CP announced/NA first/NA quarter/NA results/NA ./NA

- NA = No entity
- SC = Start Company
- CC = Continue Company
- SL = Start Location
- CL = Continue Location
- SP = Start Person
- CP = Continue Person

Dynamic Programming

- Dynamic programming is a technique used to solve optimization problems by breaking them down into overlapping subproblems.
- It stores the solutions to these subproblems in a table, so they do not need to be recalculated.
- Dynamic programming can greatly improve the efficiency of algorithms.

Factorial

- Recursive implementation:

```
def recur_factorial(n):  
    # Base case  
    if n == 1:  
        return n  
    else:  
        return n * recur_factorial(n-1)
```

- Dynamic programming implementation:

```
def dynamic_factorial(n):  
    table = [0 for i in range(0, n+1)]  
  
    # Base case  
    table[0] = 1  
  
    for i in range(1, len(table)):  
        table[i] = i * table[i-1]
```

Fibonacci

- Recursive implementation:

```
def recur_fibonacci(n):  
    if n == 1 or n == 0:  
        return 1  
    else:  
        return recur_fibonacci(n-1) + recur_fibo
```

- Dynamic programming implementation:

```
def dynamic_fibonacci(n):  
    table = [0 for i in range(0, n+1)]  
  
    # Base case  
    table[0] = 1  
    table[1] = 1  
  
    for i in range(2, len(table)):  
        table[i] = table[i-1] + table[i-2]
```

Complexity

- Recursive factorial: Exponential complexity
- Dynamic factorial: Linear complexity
- Recursive Fibonacci: Exponential complexity
- Dynamic Fibonacci: Linear complexity

The Viterbi Algorithm

Problem: For an input $x_1 \dots x_n$, find

$$\arg \max_{y_1 \dots y_{n+1}} p(x_1 \dots x_n, y_1 \dots y_{n+1})$$

where the $\arg \max$ is taken over all sequences $y_1 \dots y_{n+1}$ such that $y_i \in S$ for $i = 1 \dots n$, and $y_{n+1} = \text{STOP}$.

We assume that p takes the form:

$$p(x_1 \dots x_n, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^n e(x_i | y_i)$$

Brute Force Search is Hopelessly Inefficient

Problem: For an input $x_1 \dots x_n$, find

$$\arg \max_{y_1 \dots y_{n+1}} p(x_1 \dots x_n, y_1 \dots y_{n+1})$$

where the $\arg \max$ is taken over all sequences $y_1 \dots y_{n+1}$ such that $y_i \in S$ for $i = 1 \dots n$, and $y_{n+1} = \text{STOP}$.

The Viterbi Algorithm

The Viterbi algorithm efficiently computes the maximum probability of a tag sequence by using dynamic programming.

Steps:

- Define n as the length of the sentence.
- Define S_k for $k = -1 \dots n$ as the set of possible tags at position k : $S_{-1} = S_0 = \{*\}$, $S_k = S$ for $k \in \{1 \dots n\}$.
- Define
$$r(y_{-1}, y_0, y_1, \dots, y_k) = \prod_{i=1}^k q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^k e(x_i | y_i).$$
- Define a dynamic programming table: $\pi(k, u, v) =$ maximum probability of a tag sequence ending in tags u, v at position k .

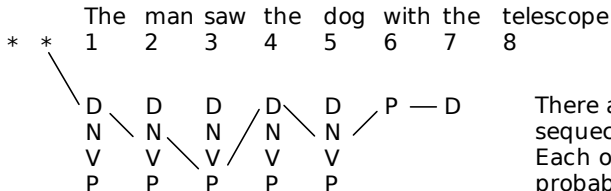
An Example

$\pi(k, u, v)$ = maximum probability of a tag sequence ending in tags u, v at position k

The man saw the dog with the telescope

An Example

$$S = \{D, N, P, V\}$$



There are many possible sequences of tags. Each of these will have a probability calculated from parameters q and e .

$\Pi(7, P, D) =$ This is the maximum probability of any of those tag sequences ending in $P D$ at position 7, the path represents the sequence with the maximum probability.

A Recursive Definition

Base case:

$$\pi(0, *, *) = 1$$

Recursive definition: For any $k \in \{1 \dots n\}$, for any $u \in S_{k-1}$ and $v \in S_k$:

$$\pi(k, u, v) = \max_{w \in S_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

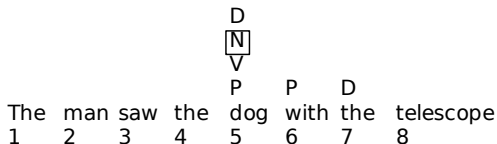
Justification for the Recursive Definition

For any $k \in \{1 \dots n\}$, for any $u \in S_{k-1}$ and $v \in S_k$:

$$\pi(k, u, v) = \max_{w \in S_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

The man saw the dog with the telescope

Justification for the Recursive Definition



$$\mathcal{S}_5 = \mathcal{S} = \{D, N, V, P\}$$

$$\Pi(7, P, D) = \max_{w \in \mathcal{S}_5} (\Pi(6, w, P) \times q(D|w, P) \times e(\text{the}|D))$$

If we think of any tag sequence that ends with tags P and D at position 7, it must contain some tag at position 5.

We are basically searching for the tag that maximizes the probability at position 5.

The Viterbi Algorithm

Algorithm 1: Viterbi Algorithm

Input: a sentence $x_1 \dots x_n$, parameters $q(s|u, v)$ and $e(x|s)$

Initialization: Set $\pi(0, *, *) = 1$; $S_{-1} = S_0 = \{*\}$, $S_k = S$
for $k \in \{1 \dots n\}$.

for $k = 1$ **to** n **do**

for $u \in S_{k-1}, v \in S_k$ **do**

$$\pi(k, u, v) = \max_{w \in S_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

end

end

return $(\max_{u \in S_{n-1}, v \in S_n} (\pi(n, u, v) \times q(STOP|u, v)))$

The Viterbi Algorithm with Backpointers

Algorithm 2: Viterbi Algorithm with Backpointers

Input: a sentence $x_1 \dots x_n$, parameters $q(s|u, v)$ and $e(x|s)$

Initialization: Set $\pi(0, *, *) = 1$; $S_{-1} = S_0 = \{*\}$, $S_k = S$ for $k \in \{1 \dots n\}$.

```
for  $k = 1$  to  $n$  do
    for  $u \in S_{k-1}, v \in S_k$  do
         $\pi(k, u, v) = \max_{w \in S_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$ 
         $\text{bp}(k, u, v) = \arg \max_{w \in S_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$ 
    end
end
 $(y_{n-1}, y_n) = \arg \max_{(u, v)} (\pi(n, u, v) \times q(\text{STOP}|u, v))$ ;           // Find maximum
probability and corresponding tags
for  $k = (n-2)$  to  $1$  do
     $y_k = \text{bp}(k+2, y_{k+1}, y_{k+2})$ ;           // Retrieve tag sequence using
backpointers
end
return(the tag sequence  $y_1 \dots y_n$ ) ; // Return the final tag sequence
```

The Viterbi Algorithm: Running Time

- $O(n|S|^3)$ time to calculate $q(s|u, v) \times e(x_k|s)$ for all k, s, u, v .
- $n|S|^2$ entries in π to be filled in.
- $O(|S|)$ time to fill in one entry.

$\Rightarrow O(n|S|^3)$ time in total.

Pros and Cons

- Hidden Markov Model (HMM) taggers are simple to train (compile counts from training corpus).
- They perform relatively well (over 90
- Main difficulty is modeling $e(\text{word}|\text{tag})$, which can be very complex if "words" are complex.

Questions?

Thanks for your Attention!

References I



Bikel, D. M., Schwartz, R. M., and Weischedel, R. M. (1999).
An algorithm that learns what's in a name.
Mach. Learn., 34(1-3):211–231.



Kupiec, J. (1992).
Robust part-of-speech tagging using a hidden markov model.
Computer speech & language, 6(3):225–242.