# 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 $x_1, x_2, \ldots, x_n$ into a sequence of tags or labels $y_1, y_2, \ldots, y_n$.
- 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
- ...

# Part-of-Speech Tag Descriptions

| | Tag | Description | Example |
|---|---|---|---|
| Open Class | **ADJ** | Adjective: noun modifiers describing properties | *red*, *young*, *awesome* |
| | **ADV** | Adverb: verb modifiers of time, place, manner | *very*, *slowly*, *home*, *yesterday* |
| | **NOUN** | words for persons, places, things, etc. | *algorithm*, *cat*, *mango*, *beauty* |
| | **VERB** | words for actions and processes | *draw*, *provide*, *go* |
| | **PROPN** | Proper noun: name of a person, organization, place, etc.. | *Regina*, *IBM*, *Colorado* |
| | **INTJ** | Interjection: exclamation, greeting, yes/no response, etc. | *oh*, *um*, *yes*, *hello* |
| Closed Class Words | **ADP** | Adposition (Preposition/Postposition): marks a noun's spacial, temporal, or other relation | *in, on, by, under* |
| | **AUX** | Auxiliary: helping verb marking tense, aspect, mood, etc., | *can, may, should, are* |
| | **CCONJ** | Coordinating Conjunction: joins two phrases/clauses | *and*, *or*, *but* |
| | **DET** | Determiner: marks noun phrase properties | *a, an, the, this* |
| | **NUM** | Numeral | *one, two, first, second* |
| | **PART** | Particle: a function word that must be associated with another word | *'s, not, (infinitive) to* |
| | **PRON** | Pronoun: a shorthand for referring to an entity or event | *she, who, I, others* |
| | **SCONJ** | Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement | *that*, *which* |
| Other | **PUNCT** | Punctuation | ; , () |
| | **SYM** | Symbols like $ or emoji | $, % |
| | **X** | Other | asdf, qwfg |

Source: [Jurafsky and Martin, 2008]

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

# 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 $p(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)$ 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: $\arg\max_{y_1, y_2, \ldots, y_n} p(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)$.

# 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 $x = x_1, x_2, \ldots, x_n$ ($x_i$ is the $i$-th word in the sentence).
- We have a tag sequence $y = y_1, y_2, \ldots, y_n$ ($y_i$ is the $i$-th tag in the sentence).
- We'll use an HMM to define $p(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)$ for any sentence $x_1, \ldots, x_n$ and tag sequence $y_1, \ldots, y_n$ of the same length. [Kupiec, 1992]
- Then, the most likely tag sequence for $x$ is:

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

# Trigram Hidden Markov Models (Trigram HMMs)

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

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

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

# Parameters of the Model

- $q(s|u, v)$ for any $s \in S \cup \{\text{STOP}\}$, $u, v \in S \cup \{*\}$
  - The value for $q(s|u, v)$ can be interpreted as the probability of seeing the tag $s$ immediately after the bigram of tags $(u, v)$.
- $e(x|s)$ for any $s \in S$, $x \in V$
  - The value for $e(x|s)$ can be interpreted as the probability of seeing observation $x$ paired with state $s$.

# 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:

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

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

# Independence Assumptions in Trigram HMMs

- Trigram Hidden Markov Models (HMMs) are derived by making specific independence assumptions in the model.
- Consider two sequences of random variables: $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$, where $n$ is the length of the sequences.
- Each $X_i$ can take any value in a finite set $V$ of words, and each $Y_i$ can take any value in a finite set $K$ of possible tags (e.g., $K = \{D, N, V \ldots\}$).
- Our goal is to model the joint probability:

$$P(X_1 = x_1, \ldots, X_n = x_n, Y_1 = y_1, \ldots, Y_n = y_n)$$

- We define an additional random variable $Y_{n+1}$ that always takes the value "STOP."

# Independence Assumptions in Trigram HMMs

- The key idea in HMMs is the factorization of the joint probability:

$$P(X_1 = x_1, \ldots, X_n = x_n, Y_1 = y_1, \ldots, Y_{n+1} = y_{n+1})$$

$$= \prod_{i=1}^{n+1} P(Y_i = y_i | Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1}) \times \prod_{i=1}^{n} P(X_i = x_i | Y_i = y_i)$$

- We first assume that:

$$P(Y_i = y_i | Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1}) = q(y_i | y_{i-2}, y_{i-1})$$

- This assumes that the sequence $Y_1, \ldots, Y_{n+1}$ is a second-order Markov sequence, where each state depends only on the previous two states.
- And we also assume that:

$$P(X_i = x_i | Y_i = y_i) = e(x_i | y_i)$$

- This assumes that the value of the random variable $X_i$ depends only on the value of $Y_i$.
- These independence assumptions allow for the derivation of the joint probability equation.

# Why the Name?

$$p(x_1,\ldots,x_n,y_1,\ldots,y_n) = q(\mathrm{STOP}|y_{n-1},y_n) \times \prod_{j=1}^{n} q(y_j|y_{j-2},y_{j-1}) \times \prod_{j=1}^{n} e(x_j|y_j)$$

- Markov Chain:

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

- 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 $\geq 5$ 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

# Decoding Problem

Decoding Problem: For an input $x_1 \ldots x_n$, find

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

where the $\arg\max$ is taken over all sequences $y_1 \ldots y_{n+1}$ such that $y_i \in S$ for $i = 1 \ldots n$, and $y_{n+1} = \text{STOP}$.
We assume that $p$ takes the form:

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

Recall that we have assumed in this definition that $y_0 = y_{-1} = *$, and $y_{n+1} = \text{STOP}$.

# Naive Brute Force Method

The naive, brute force method for finding the highest scoring tag sequence is to enumerate all possible tag sequences $y_1, \ldots, y_{n+1}$, score them under the function $p$, and select the sequence with the highest score.

- Example:
  - Input sentence: *the dog barks*
  - Set of possible tags: $K = \{D, N, V\}$
- Enumerate all possible tag sequences:
  - *D D D STOP*
  - *D D N STOP*
  - *D D V STOP*
  - *D N D STOP*
  - *D N N STOP*
  - *D N V STOP*
  - ...

# Naive Brute Force Method

- In this case, there are $3^3 = 27$ possible sequences.
- However, for longer sentences, this method becomes inefficient.
- For an input sentence of length $n$, there are $|K|^n$ possible tag sequences.
- The exponential growth makes brute-force search infeasible for reasonable length sentences.

# Viterbi Decoding Dynamic Programming

- The algorithm used by HMMs to perform efficient decoding is called Viterbi decoding.
- Viterbi decoding uses dynamic programming.
- Dynamic programming is a technique for solving optimization problems by breaking them down into overlapping subproblems.
- It stores the solutions to these subproblems in a table so that they do not have to be recalculated.
- Dynamic programming can greatly improve the efficiency of algorithms.
- Next, we show how dynamic programming works with two examples: Factorial and Fibonacci

# Factorial

- Recursive implementation:

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

- Dynamic programming implementation:

```python
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]

    return table[n]
```

# Fibonacci

- Recursive implementation:

```
def recur_fibonacci(n):
    if n == 1 or n == 0:
        return 1
    else:
        return recur_fibonacci(n-1) + recur_fibonacci(n-2)
```

- 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]

    return table[n]
```

# Complexity

- In recursive implementations, the complexity can be quite high due to repeated calculations of the same subproblems.
- However, dynamic programming can significantly reduce the complexity by storing the solutions to subproblems in a table or array and reusing them when needed.
- This approach eliminates the redundant calculations and allows for a more efficient computation.
- For the case of Fibonacci the complexity is reduced from exponential to linear.

# The Viterbi Algorithm

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

**Definitions:**

- Define $n$ as the length of the sentence.
- Define $S_k$ for $k = -1 \ldots n$ as the set of possible tags at position $k$: $S_{-1} = S_0 = \{*\}$, $S_k = S$ for $k \in \{1 \ldots n\}$.
- Define a truncated version of the probability encoded by the HMM until position $k$, $r(y_{-1}, y_0, y_1, \ldots, y_k)$ as:

$$r(y_{-1}, y_0, y_1, \ldots, y_k) = \prod_{i=1}^{k} q(y_i | y_{i-2}, y_{i-1})$$

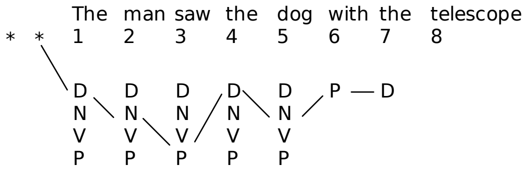
- Define a dynamic programming table $\pi(k, u, v)$ as the maximum probability of a tag sequence ending in tags $u, v$ at position $k$:

$$\pi(k, u, v) = \max_{y_{-1}, y_0, y_1, \ldots, y_k : y_{k-1} = u, y_k = v} r(y_{-1}, y_0, y_1, \ldots, y_k)$$

# An Example

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

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




- There are many possible sequences of tags.
- Each of them has a probability calculated from the parameters $q$ and $e$.
- $\pi(7, P, D)$ is the maximum probability that one of these tag sequences ends 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 $k \in \{1 \ldots n\}$, for any $u \in S_{k-1}$ and $v \in S_k$:

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

# Justification for the Recursive Definition

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

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

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | D | | | |
| | | | | $\boxed{\text{N}}$ | | | |
| | | | | V | | | |
| | | | | P | P | D | |
| The | man | saw | the | dog | with | the | telescope |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

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

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

- Let's consider an arbitrary 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 $x_1 \ldots x_n$, parameters $q(s|u, v)$ and $e(x|s)$

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

**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 $x_1 \ldots x_n$, parameters $q(s|u, v)$ and $e(x|s)$
**Initialization:** Set $\pi(0, *, *) = 1$; $S_{-1} = S_0 = \{*\}$, $S_k = S$ for $k \in \{1 \ldots n\}$.

**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* $y_1 \ldots y_n$) ; `// 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 $\pi$ 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% performance on named entity recognition).
- 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.

Jurafsky, D. and Martin, J. H. (2008).
*Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*.
Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition.

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