



Curso de

Algoritmos de Clasificación de Texto



Francisco Camacho
@el_pachocamacho

[C1] Introducción a la desambiguación

Desambiguación y etiquetado
de palabras

¿Por qué es tan difícil?



El lenguaje humano es **difuso**,
ambiguo y requiere **mucho**
contexto

**Debo ir al banco
para retirar dinero**

**Te puedes sentar en
ese banco para
descansar**

**Mi hermano es una
persona muy noble**

**El noble del castillo no
quiere ayudar a su pueblo**

adjetivo

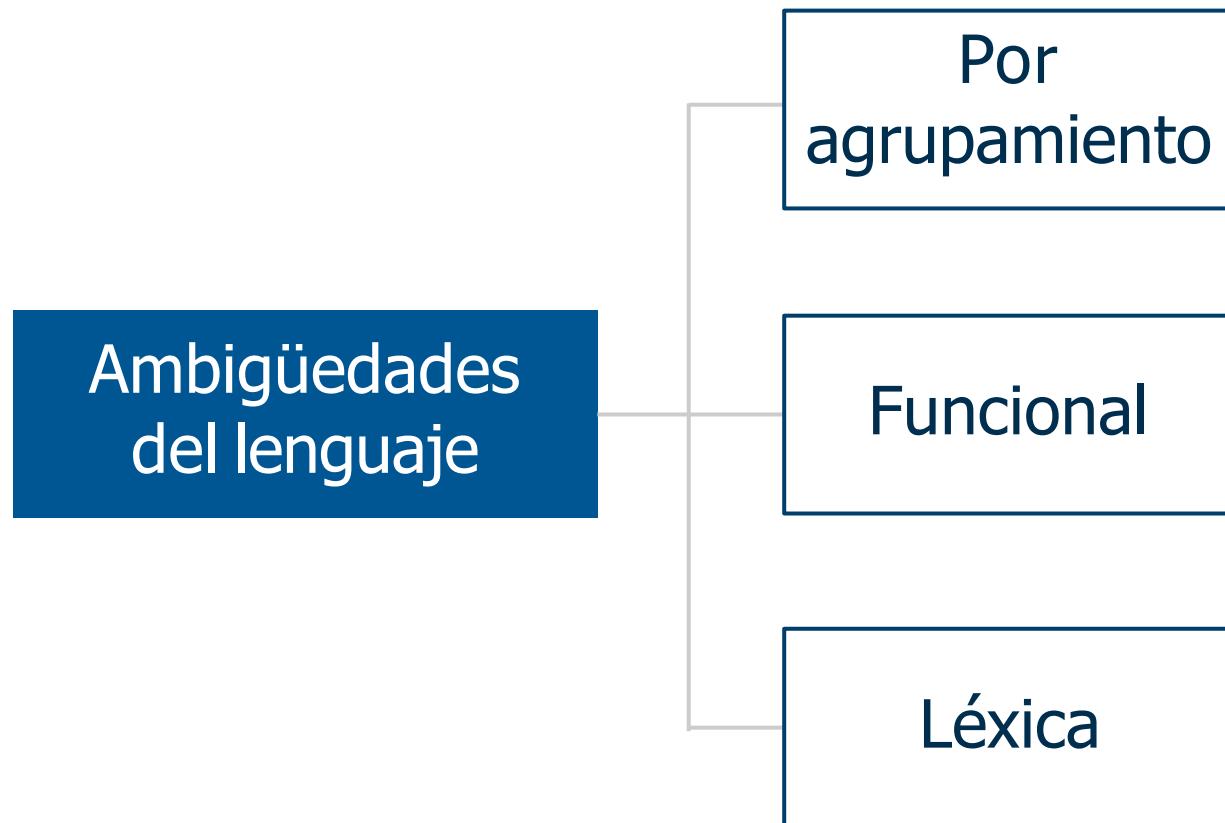
Mi hermano es una persona

sustantivo

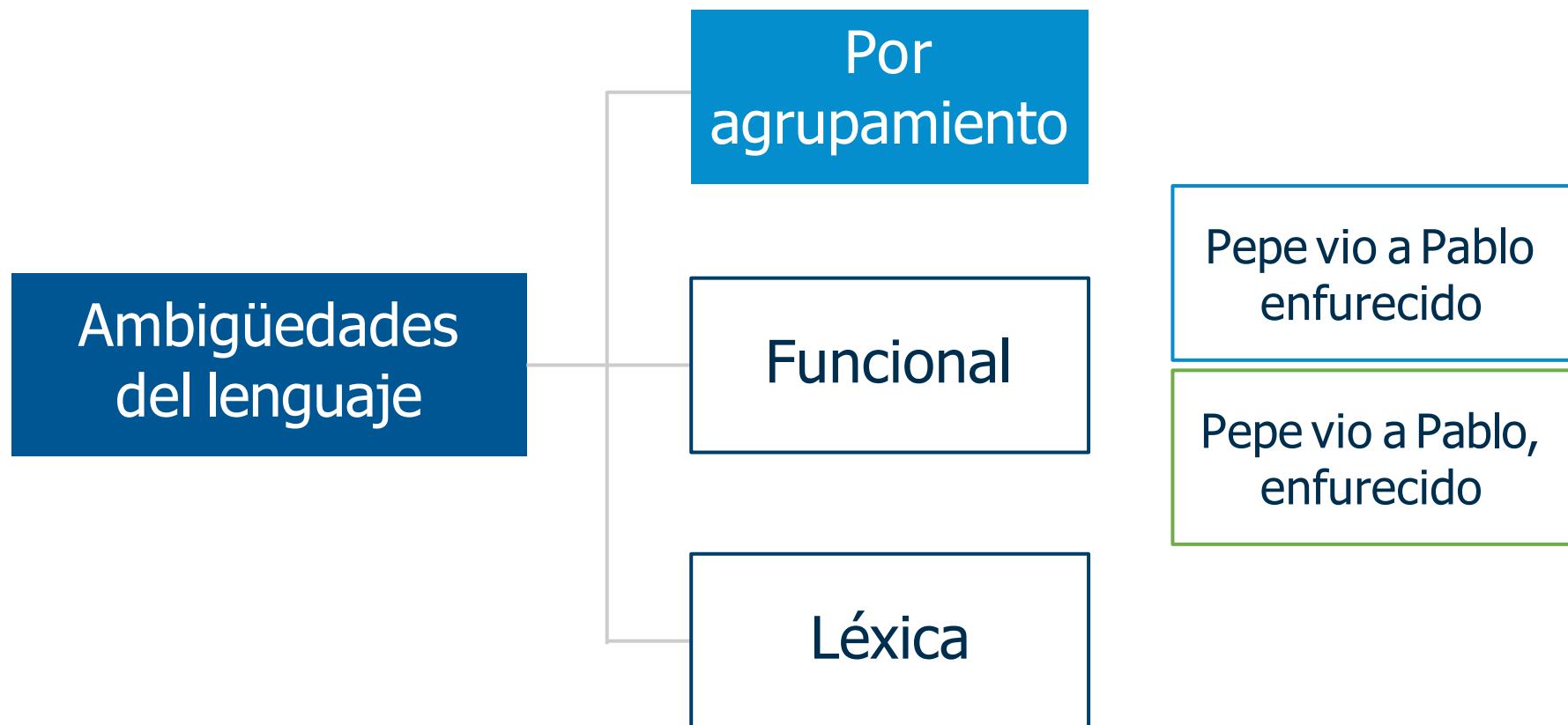
muy **noble**

El **noble** del castillo no
quiere ayudar a su pueblo

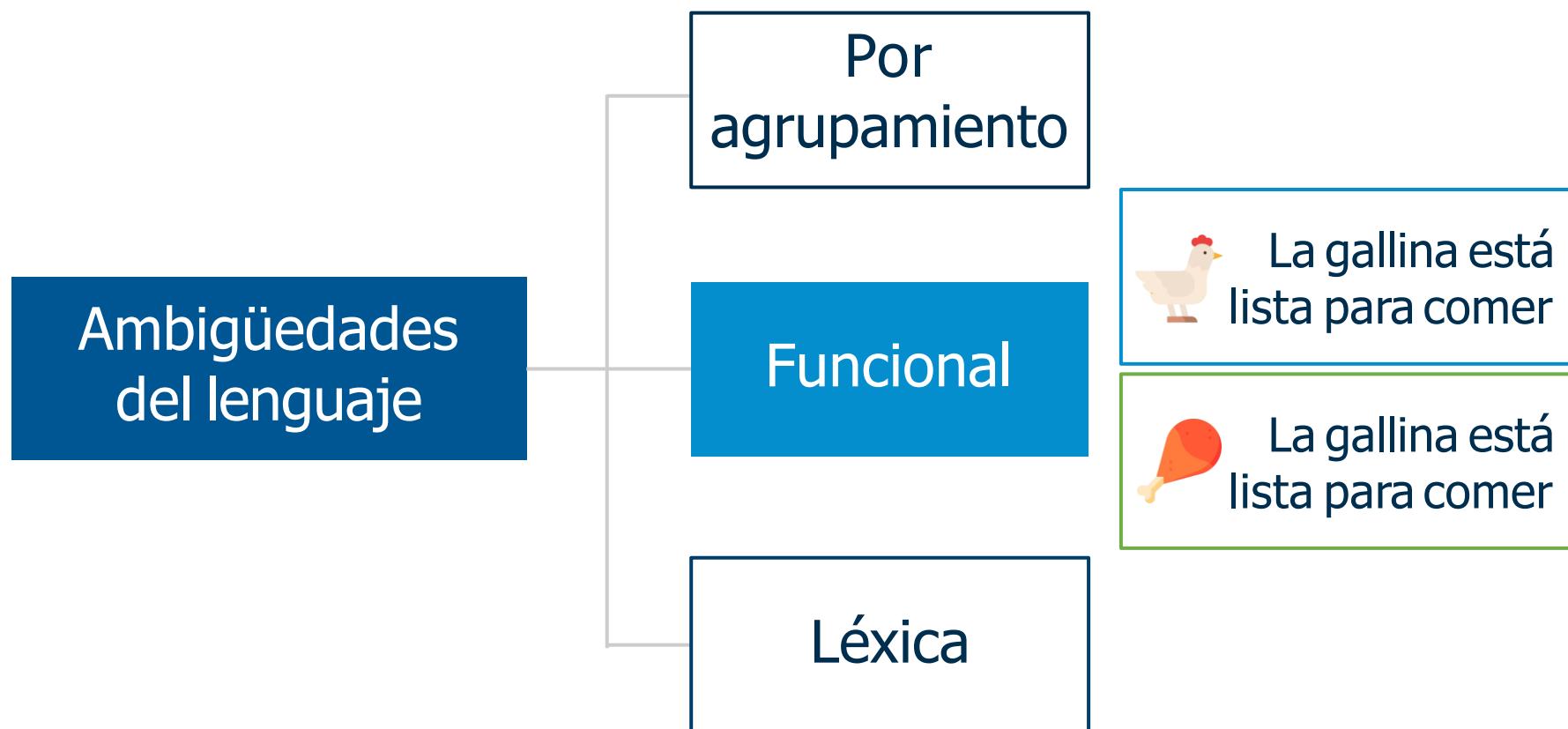
Ambigüedades del lenguaje



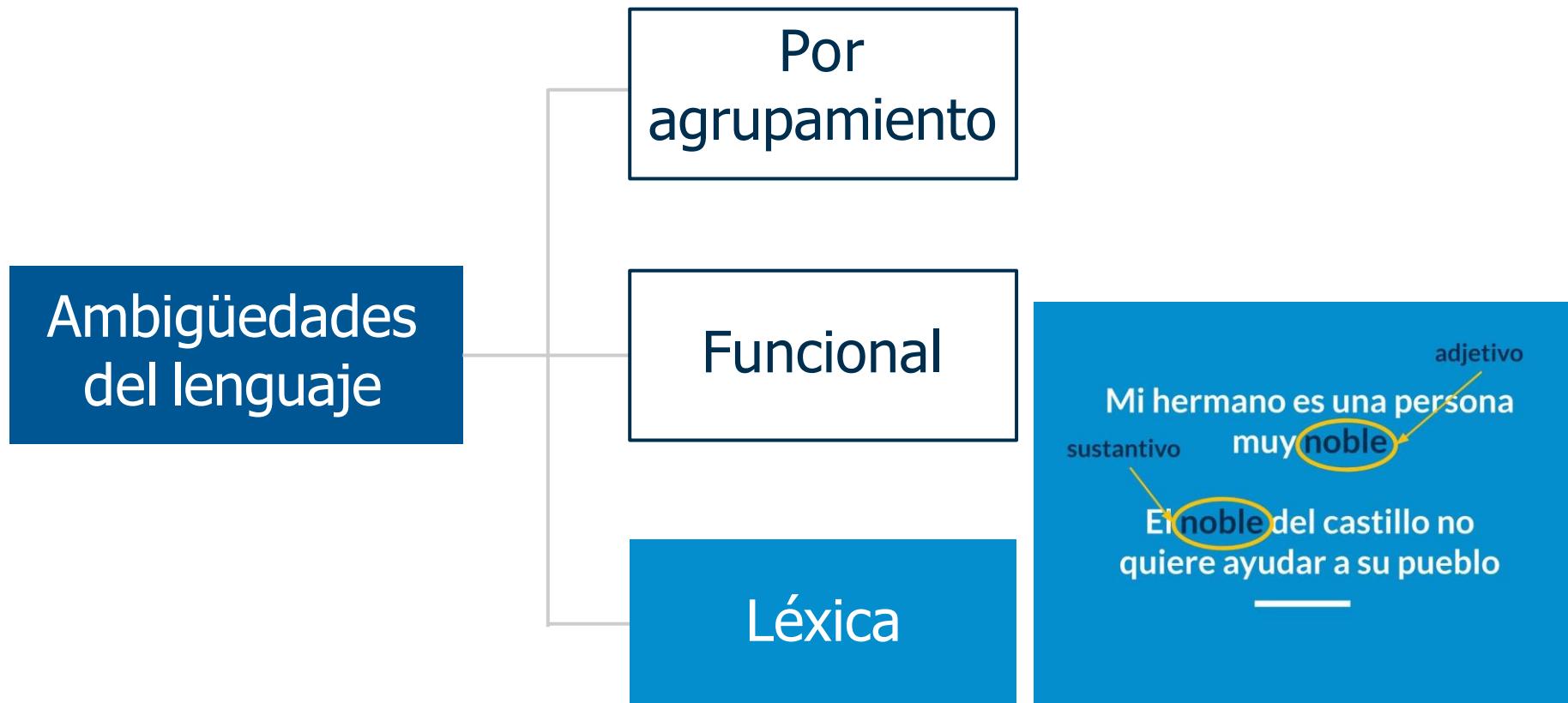
Ambigüedades del lenguaje



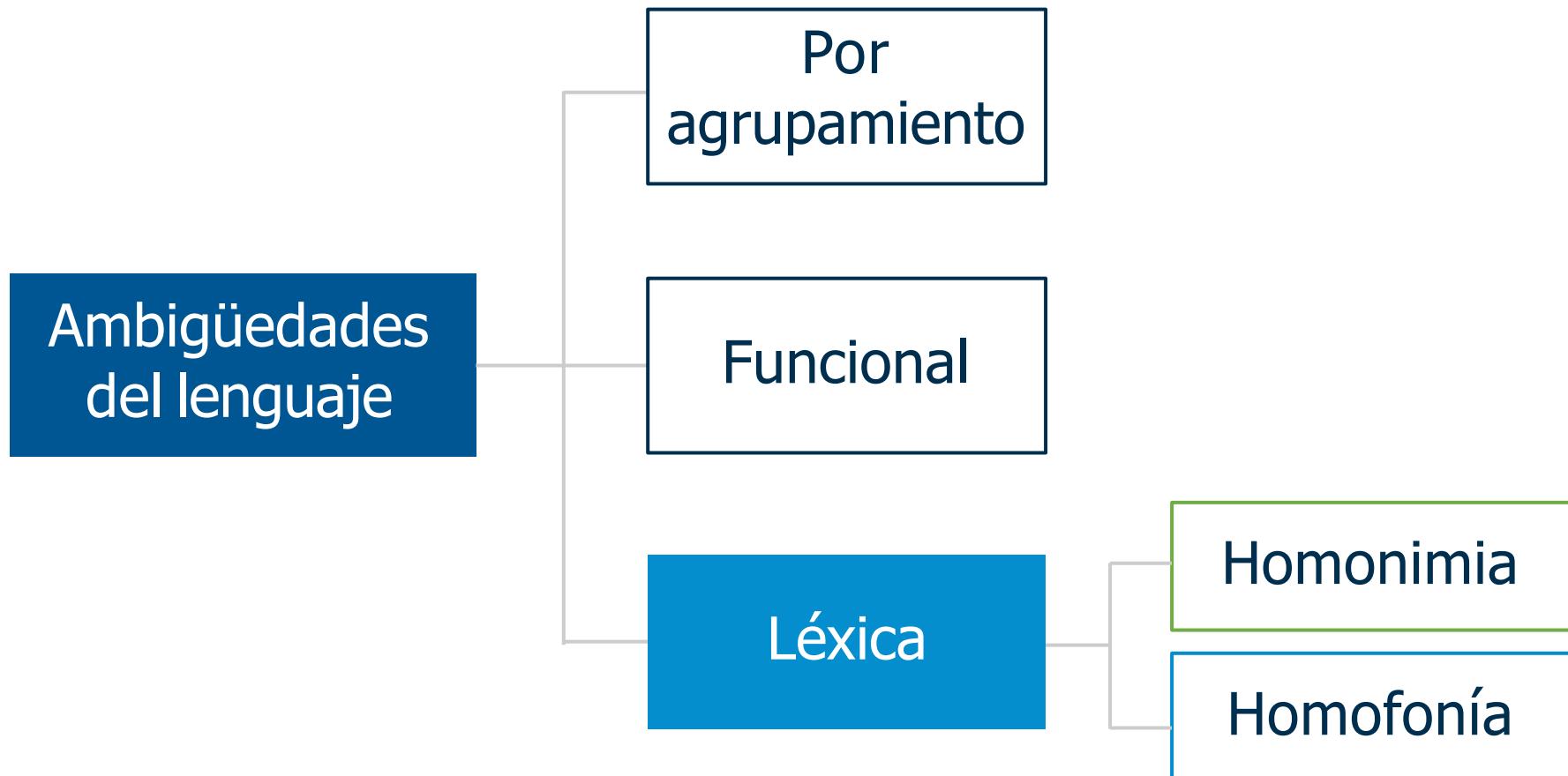
Ambigüedades del lenguaje



Ambigüedades del lenguaje



Ambigüedades del lenguaje



Etiquetado de palabras

Natural Language API demo

Try the API

El noble del palacio

RESET

See supported languages

Entities

Sentiment

Syntax

Categories

Dependency Parse label Part of speech Lemma Morphology

det

El

DET

gender=MASCULINE
number=SINGULAR
proper=NOT_PROPER

root

noble

NOUN

gender=MASCULINE
number=SINGULAR
proper=NOT_PROPER

prep

del

ADP

gender=MASCULINE
number=SINGULAR
proper=NOT_PROPER

pobj

palacio

NOUN

gender=MASCULINE
number=SINGULAR
proper=NOT_PROPER

<https://cloud.google.com/natural-language>

Aplicaciones

- Mejoras en motores de búsqueda, e-commerce y web.
- Automatización en manejo de CRMs.



Aplicaciones

- Censura en redes sociales.
- Orden de datos no-estructurados.



Roadmap del contenido





[C4]

Cadenas

de Markov

Desambiguación y etiquetado
de palabras

```
import nltk  
nltk.download('punkt')   
nltk.download('averaged_perceptron...')  
from nltk import word_tokenize
```

sumit

2021-08-19 10:53:27

punkt: (Palabra alemana que significa puntuación) un tokenizador.

averaged_perceptron: un clasificador, es la base del universo de técnicas de clasificación.

tokenizer

```
import nltk  
nltk.download('punkt')  
nltk.download('averaged_perceptron...')  
from nltk import word_tokenize
```

tagger

Tokenizador: ¿Punkt?

Unsupervised Multilingual Sentence Boundary Detection

Tibor Kiss*
Ruhr-Universität Bochum

Jan Strunk**
Ruhr-Universität Bochum

In this article, we present a language-independent, unsupervised approach to sentence boundary detection. It is based on the assumption that a large number of ambiguities in the determination of sentence boundaries can be eliminated once abbreviations have been identified. Instead of relying on orthographic clues, the proposed system is able to detect abbreviations with high accuracy using three criteria that only require information about the candidate type itself and are independent of context: Abbreviations can be defined as a very tight collocation consisting of a truncated word and a final period, abbreviations are usually short, and abbreviations sometimes contain internal periods. We also show the potential of collocational evidence for two other important subtasks of sentence boundary disambiguation, namely, the detection of initials and ordinal numbers. The proposed system has been tested extensively on eleven different languages and on different text genres. It achieves good results without any further amendments or language-specific resources. We evaluate its performance against three different baselines and compare it to other systems for sentence boundary detection proposed in the literature.

<https://www.aclweb.org/anthology/J06-4003.pdf>

Etiquetador: ¿Averaged Perceptron Tagger?

A Good Part-of-Speech Tagger in about 200 Lines of Python

SEP 17, 2013 · BY MATTHEW HONNIBAL · ~12 MIN. READ

Up-to-date knowledge about natural language processing is mostly locked away in academia. And academics are mostly pretty self-conscious when we write. We're careful. We don't want to stick our necks out too much. But under-confident recommendations suck, so here's how to write a good part-of-speech tagger.

A Maximum Entropy Model for Part-Of-Speech Tagging

Adwait Ratnaparkhi

University of Pennsylvania

Dept. of Computer and Information Science

adwait@gradient.cis.upenn.edu

Abstract

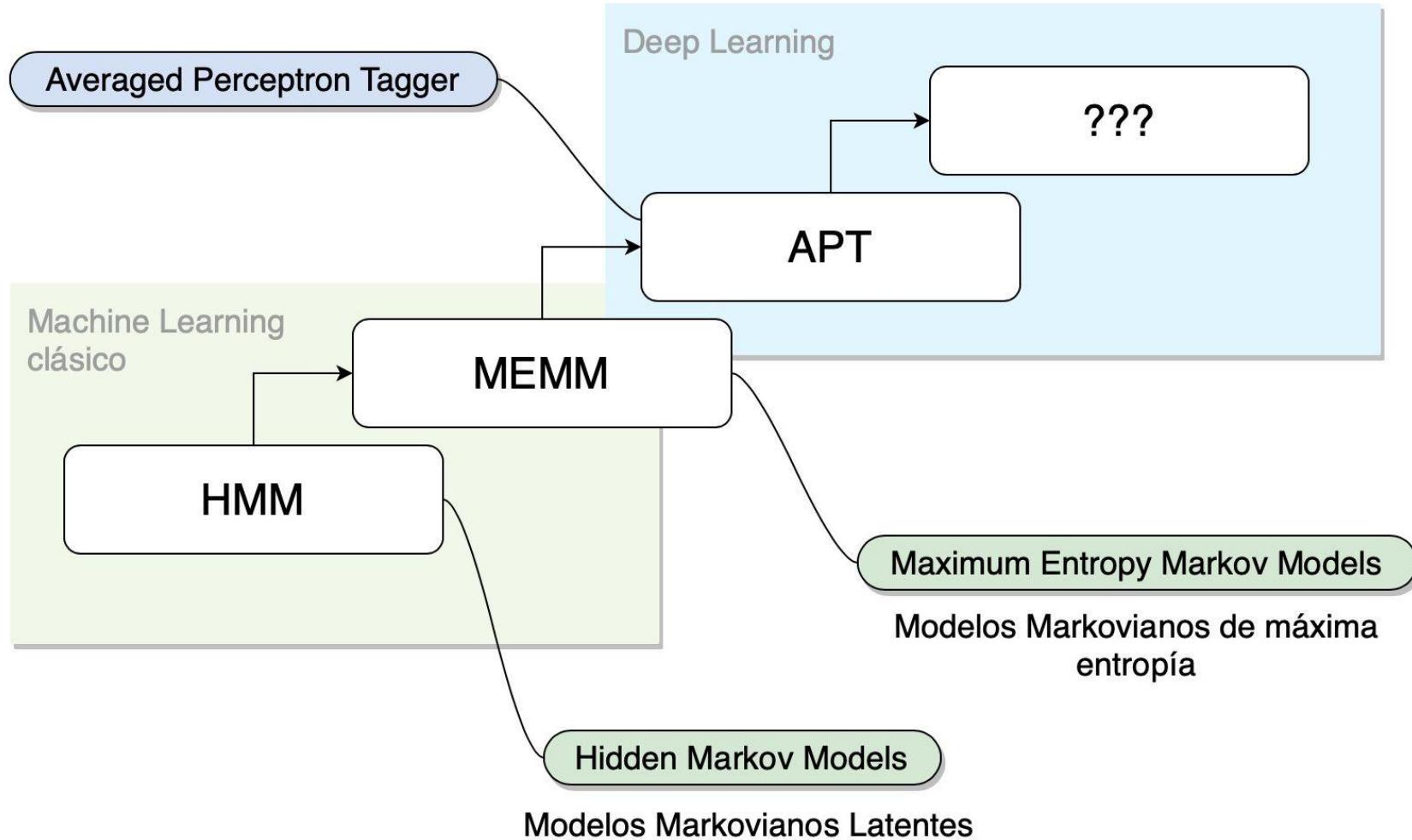
This paper presents a statistical model which trains from a corpus annotated with Part-Of-Speech tags and assigns them to previously unseen text with state-of-the-art accuracy(96.6%). The model can be classified as a *Maximum Entropy* model and simultaneously uses many contextual “features” to predict the POS tag. Furthermore, this paper demonstrates the use of specialized features to model difficult tagging decisions, discusses the corpus consistency problems discovered during the implementation of these features, and proposes a training strategy that mitigates these problems.

bines diverse forms of contextual information in a principled manner, and does not impose any distributional assumptions on the training data. Previous uses of this model include language modeling(Lau et al., 1993), machine translation(Berger et al., 1996), prepositional phrase attachment(Ratnaparkhi et al., 1994), and word morphology(Della Pietra et al., 1995). This paper briefly describes the maximum entropy and maximum likelihood properties of the model, features used for POS tagging, and the experiments on the Penn Treebank Wall St. Journal corpus. It then discusses the consistency problems discovered during an attempt to use specialized

<https://www.aclweb.org/anthology/W96-0213.pdf>

Escalera de modelos

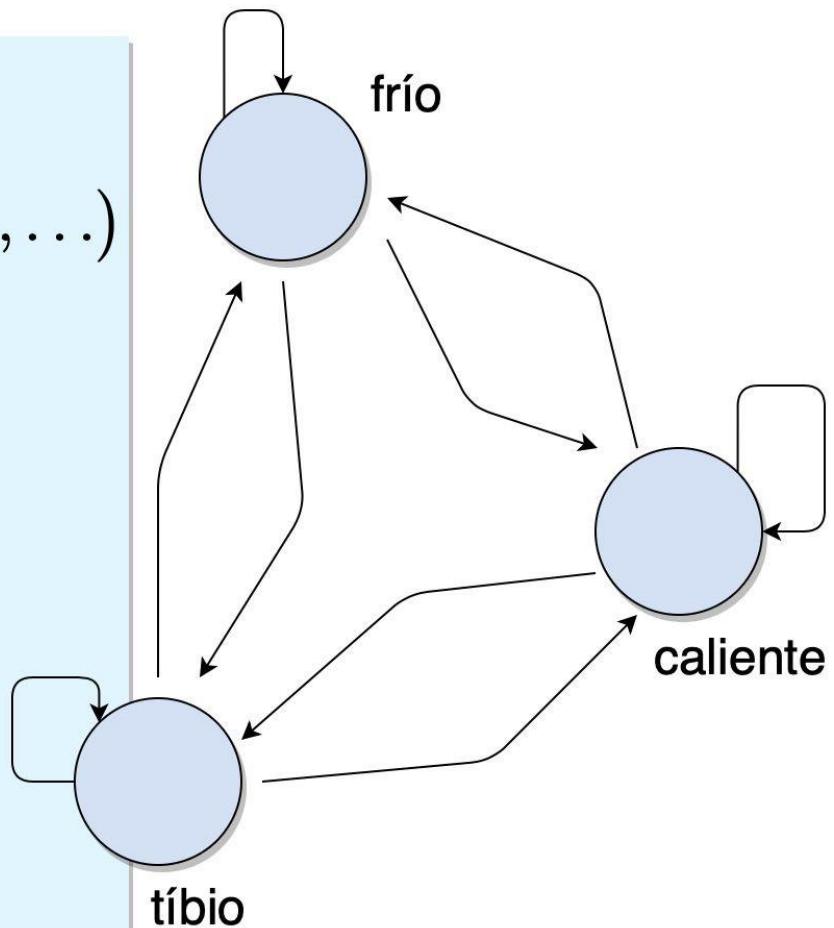
En base a ello podemos definir una escalera de modelos, donde los modelos markovianos latentes serán nuestra base para entender los siguientes.



Modelos Markovianos

MC: Markov Chain

$(\text{dia}_1, \text{dia}_2, \dots) = (\text{frío}, \text{caliente}, \dots)$



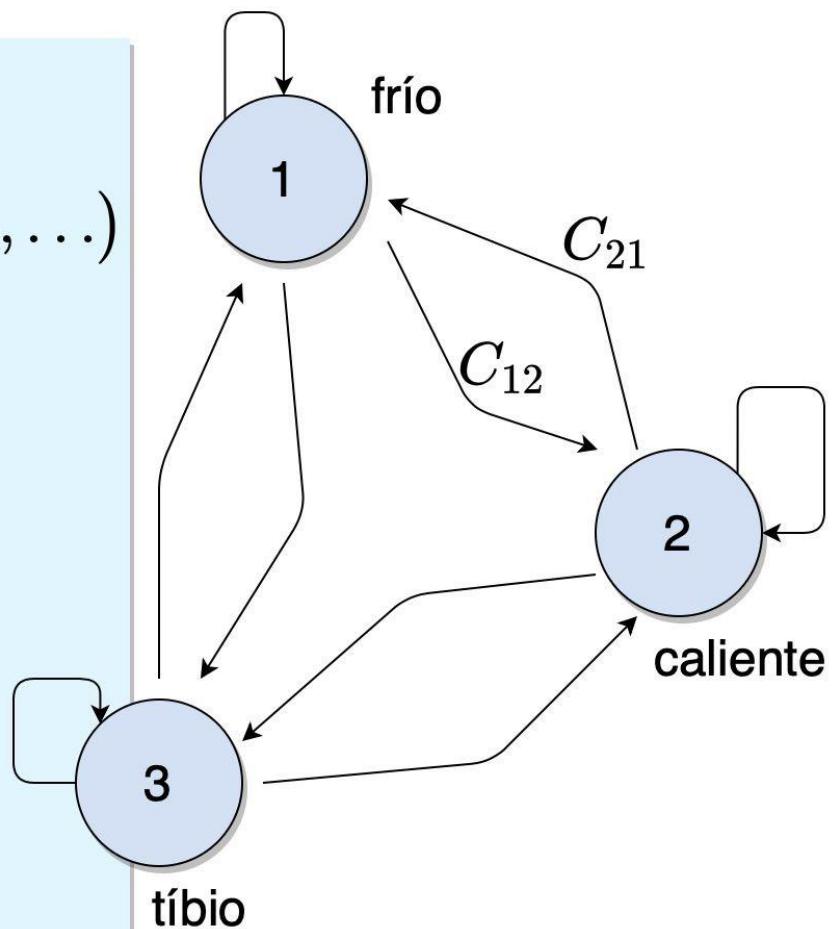
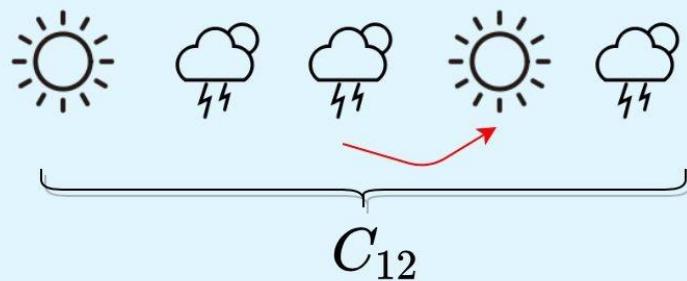
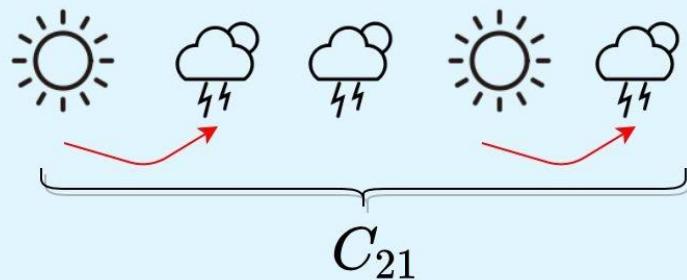
Modelos Markovianos

El diagrama mide la probabilidad de que a un dia frio le siga otro dia frio, o que de un dia frio le siga un dia caliente, etc. Todas las flechas definen una transición, en conjunto definen todas las transiciones.

Una cadena de Markov define la probabilidad de transición entre los posibles estados que un sistema puede tener. Haciendo la lógica para las posibles transiciones obtenemos la matriz de transición.

Probabilidades de transición

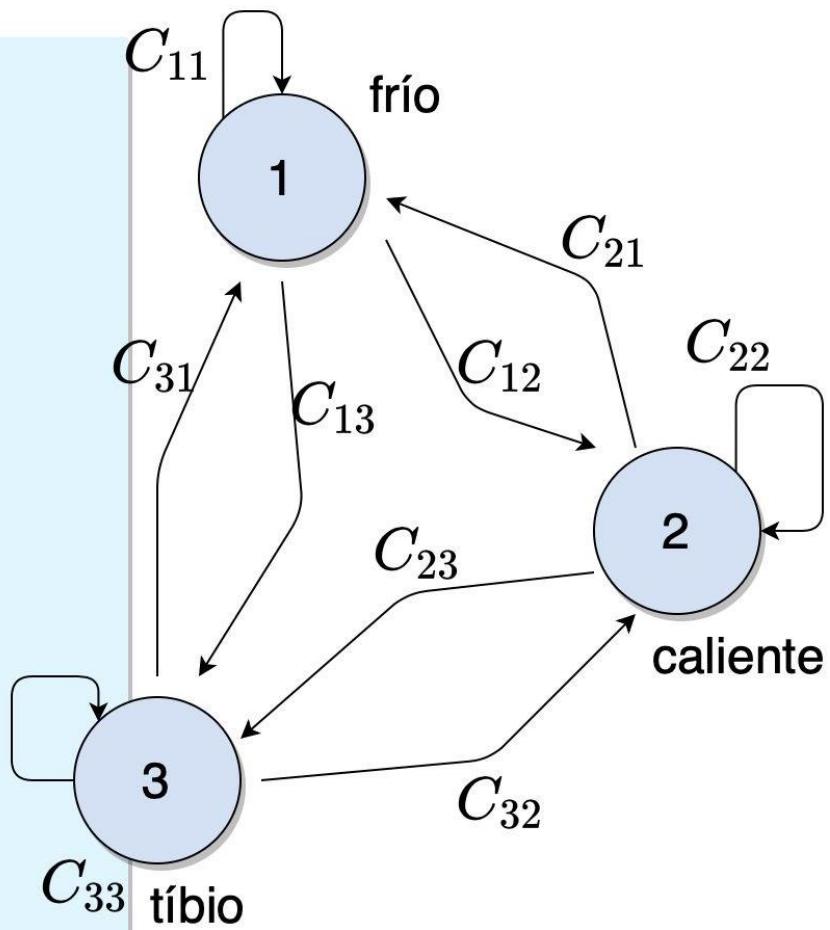
$$(\text{dia}_1, \text{dia}_2, \dots) = (\text{frío}, \text{caliente}, \dots)$$



Modelos Markovianos

Matriz de transición

	C_{11}	C_{12}	C_{13}
	C_{21}	C_{22}	C_{23}
	C_{31}	C_{32}	C_{33}



Modelos Markovianos

Distribución inicial de estados

$$A = \begin{array}{|c|c|c|} \hline C_{11} & C_{12} & C_{13} \\ \hline C_{21} & C_{22} & C_{23} \\ \hline C_{31} & C_{32} & C_{33} \\ \hline \end{array}$$

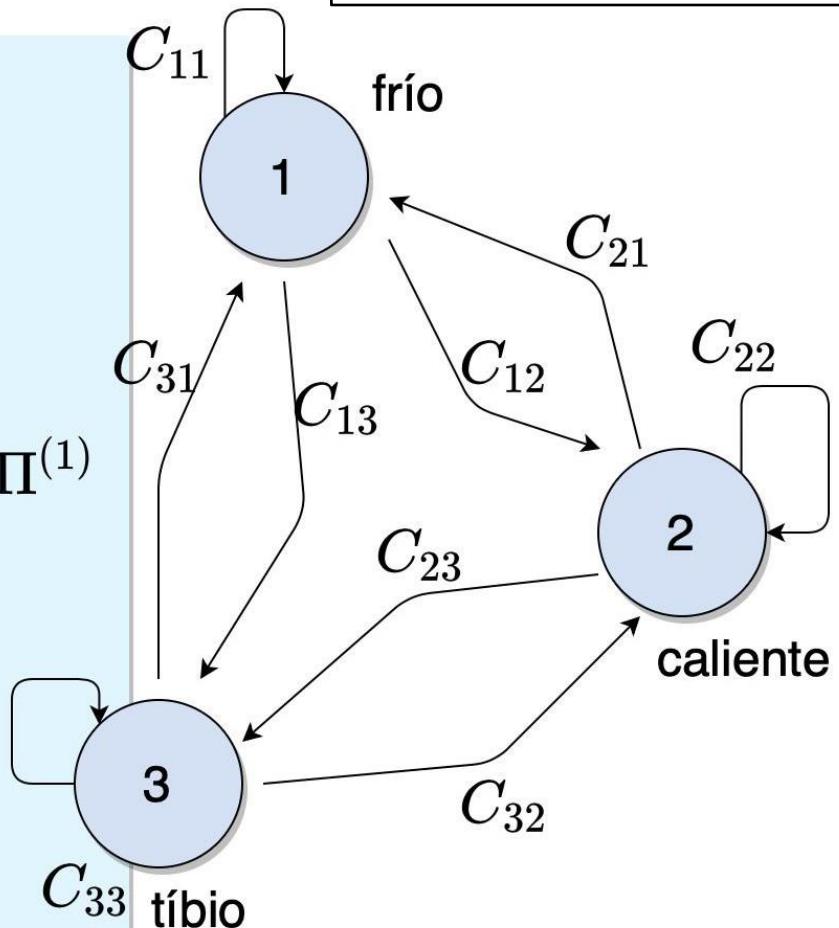
$$\Pi^{(0)} = \begin{array}{|c|c|c|} \hline \rho_1^{(0)} & \rho_2^{(0)} & \rho_3^{(0)} \\ \hline \end{array}$$

$$\Pi^{(1)} = \begin{array}{|c|c|c|} \hline \rho_1^{(1)} & \rho_2^{(1)} & \rho_3^{(0)} \\ \hline \end{array}$$

$$\Pi^{(0)} A = \Pi^{(1)}$$

Otro componente importante es la distribución inicial de estados (letra PI). PI tiene 3 componentes, que definen la probabilidad inicial de que un dia sea frio o caliente o tibio.

Lo que sucede es que nosotros multiplicamos nuestra matriz de transición por nuestro estado inicial PI(0), ese resultado nos dará el siguiente vector de estados PI(1). Eso nos indica la manera en que las probabilidades van cambiando a medida que el sistema evoluciona.





[C5] Modelos Markovianos Latentes

Desambiguación y etiquetado
de palabras

Modelos Markovianos

Ingredientes de MC

$(\text{dia}_1, \text{dia}_2, \dots) = (\text{frío}, \text{caliente}, \dots)$

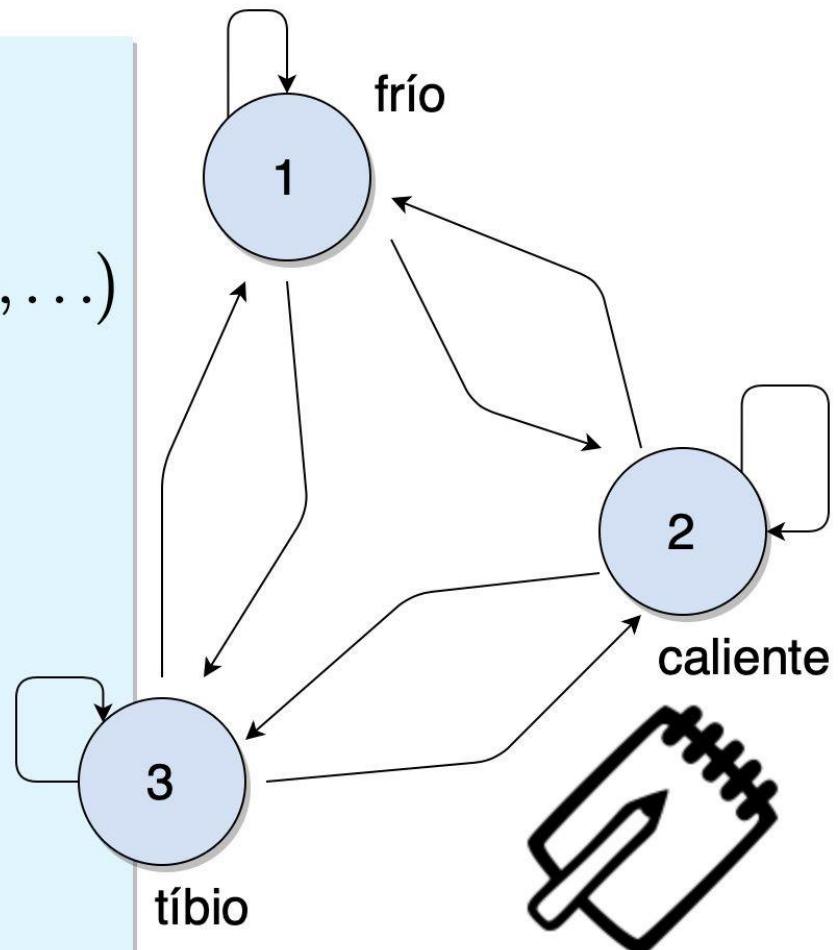
A : matriz transición

Π : distribución de estados

$$\Pi^{(0)} A = \Pi^{(1)}$$

Las cadenas de markov son la base para entender como funcionan los etiquetadores de palabras. En una cadena de markov tenemos dos ingredientes principales:

- Matriz de transición (cada elemento representa la probabilidad de transición al siguiente estado)
- Distribución de estados



Ejemplo

Consideramos una cadena de Markov donde tenemos 3 estados(1: frio, 2: caliente, 3:tibio).

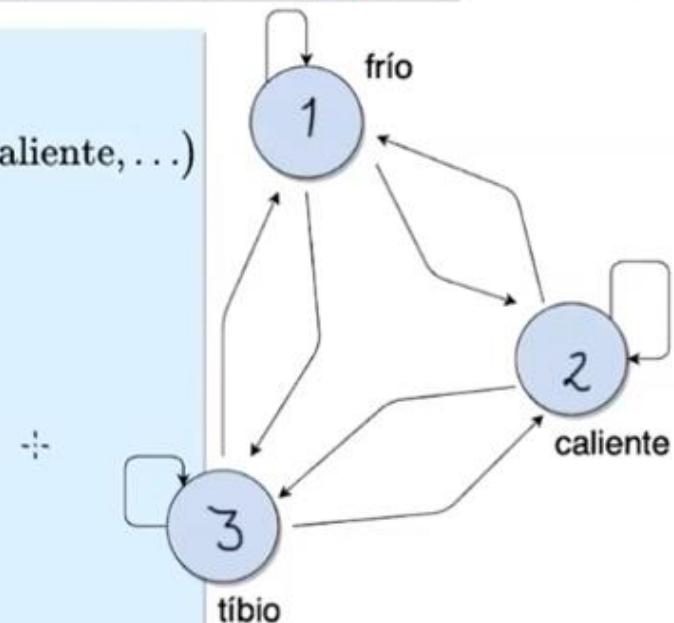


Probabilidad Condicional

$$P(j|i) = P(i,j)/P(i)$$

MC: Markov Chain

$$(dia_1, dia_2, \dots) = (\text{frío}, \text{caliente}, \dots)$$



$$\{d_1, d_2, d_3, d_4, d_5\} \rightarrow \{f, f, t, c, c, c\}$$

Vamos a calcular un caso particular, la transición de que yo estando en el estado 3, transiciones al estado 2

Probabilidad Condicional

$$P(j|i) = P(i,j)/P(i)$$

$$P(2|3) = P(3,2)/P(3)$$

$$\overbrace{P(3,2)}^{P(3,2)} = \frac{1}{5}$$

$$P(3) = \frac{1}{5}$$

$$P(2|3) = 1$$

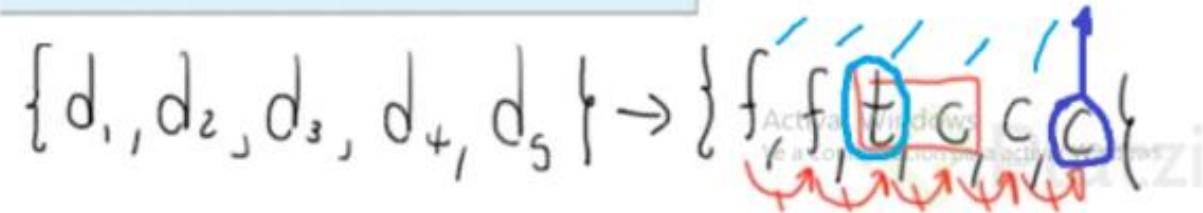
MC: Markov Chain

$$(dia_1, dia_2, \dots) = (\text{frío}, \text{caliente}, \dots)$$

$P(2|3)$ = La transición de que estando en el estado 3(tíbido), al día siguiente el clima sea 2(caliente).

$P(3/2)$ = La probabilidad de que siendo un día 3(tíbido), el día siguiente sea 2(caliente).

$P(3)$ = probabilidad de cuantos días tíbidos hay en la secuencia.



Calculamos la probabilidad del evento 2 dado 3, primero calculamos la probabilidad $P(3:\text{tíbido}, 2:\text{caliente})$, esto es la probabilidad de que transicionemos de un estado 3 a un estado 2, (contamos los eventos y tenemos solo 1 evento posible, y lo dividimos entre la cantidad de eventos totales posibles (5 eventos)).

$$P(3,2) = 1/5$$

Ahora contamos la cantidad de días tíbidos en nuestros datos, y lo dividimos entre la cantidad de días.

$$P(3) = 1/5$$

Por tanto la probabilidad $P(2|3) = 1$, y estará representada en la matriz con la celda C32

Ahora replicamos los pasos para construir la matriz.

$$P(j|i) = P(i,j)/P(i)$$

$$P(2|3)=1 = C_{32}$$

— (o) —

$$\begin{matrix} & 1 & 2 & 3 \\ \text{(f)} 1 & \frac{1}{2} & 0 & \frac{1}{2} \\ \text{(c)} 2 & 0 & 1 & 0 \\ \text{(t)} 3 & 0 & 1 & 0 \end{matrix}$$

ya con la matriz de estados consideramos una distribución de estados inicial.

$$\Pi^{(0)} = [0.4 \quad 0.2 \quad 0.4]$$

Los estados de markov se definen solo por el estado inmediatamente anterior.

Como vimos en la clase inicial al multiplicar la matriz A por el estado inicial me dará la probabilidad de ocurrencia de los estados par el dia siguiente y obtenemos el siguiente vector PI(1)

$$P(j|i) = P(i,j)/P(i)$$

$$P(2|3)=1 = C_{32}$$

$$\Pi^{(0)} = [0.4 \quad 0.2 \quad 0.4]$$

$$\begin{matrix} & 1 & 2 & 3 \\ \text{(f)} 1 & \frac{1}{2} & 0 & \frac{1}{2} \\ \text{(c)} 2 & 0 & 1 & 0 \\ \text{(t)} 3 & 0 & 1 & 0 \end{matrix} = A$$

$$\Pi^{(0)} A = \Pi^{(1)}$$

Y esta es la formula general de la Cadena de Markov.

Verificamos haciendo la multiplicación elemento a elemento.

$$\pi^{(t)} A = \pi^{(t+1)}$$

EN general cuantos tienes estados en un tiempo t, y lo multiplicas por la matriz de estados da como resultado la probabilidad de cada estado al dia siguiente, y esta es la formula general de una cadena de markov.

Utilizando el producto cruz (o producto punto) hacemos la multiplicación de ambas matrices y obtenemos el resultado para la probabilidad de los estados para el dia siguiente (Vector $\pi^{(1)}$).

$$P(j|i) = P(i,j)/P(i)$$

$$P(2|3) = 1 = C_{32}$$

$$\pi^{(0)} = [0.4 \quad 0.2 \quad 0.4]$$

$$(f) \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \begin{bmatrix} 1 & 2 & 3 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} = A$$

$$\pi^{(t)} A = \pi^{(t+1)}$$

$$\pi^{(0)} A = \pi^{(1)} = [0.2, 0.6, 0.2]$$



Modelos Markovianos Latentes

HMM: Hidden Markov Model

Sequencia de palabras

$(w_1, w_2, \dots) =$
(Pedro, es, ingeniero)

Sequencia de etiquetas

$(t_1, t_2, \dots) =$
(Sustantivo, Verbo, Sustantivo)

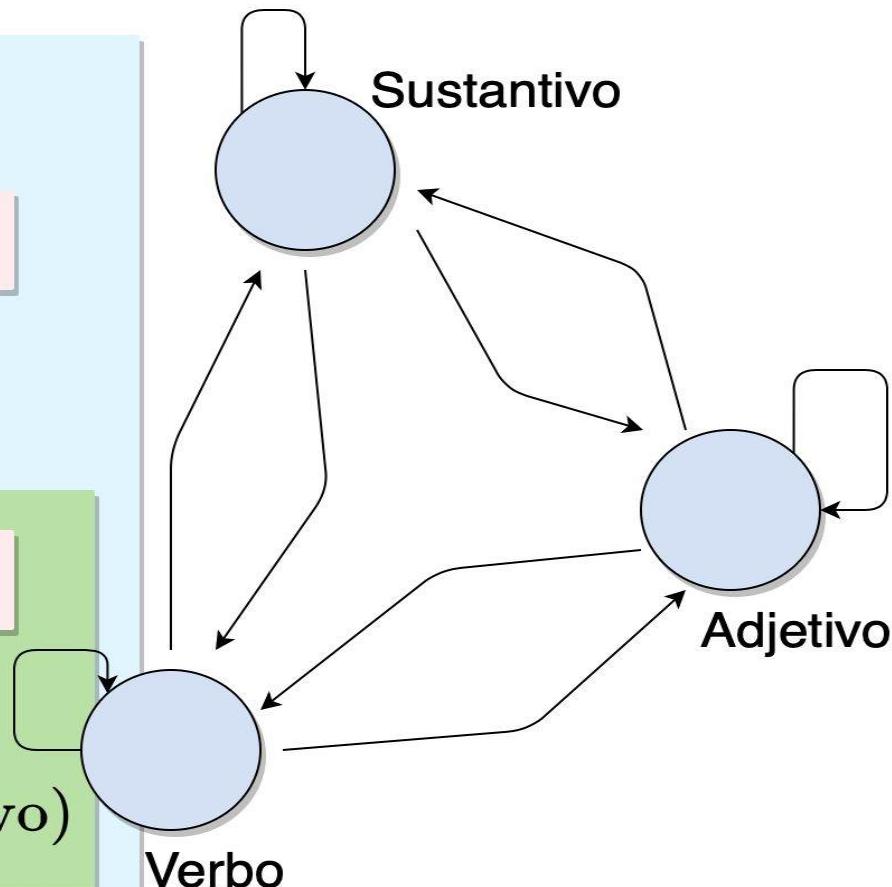
Ahora que vimos las bases de una cadena de markov y sus ingredientes esenciales veamos de manera rápida com se expande esto a una la HMM o Modelo Markoviano Latente (Hidden Markov Model).

La expansion es la siguiente, dejemos de pensar en dias, y en climas cada dia.

Pensemos en secuencia de palabras y secuencia de etiquetas de cada una de esas palabras.

(pedro, es, ingeniero) | (sustantivo, Verbo, Sustantivo)

Esto es una cadena latente (oculta)y el propósito del modelo es descubrir o encontrar cual es esa cadena.



[C6] Entrenando un HMM

Desambiguación y etiquetado
de palabras

Modelos Markovianos

En la clase pasada vimos como funciona una cadena de markov y sus ingredientes esenciales, luego como esto se puede expandir a un modelo markoviano latente considerando una secuencia de palabras y por otro lado una cadena latente u oculta de etiquetas.

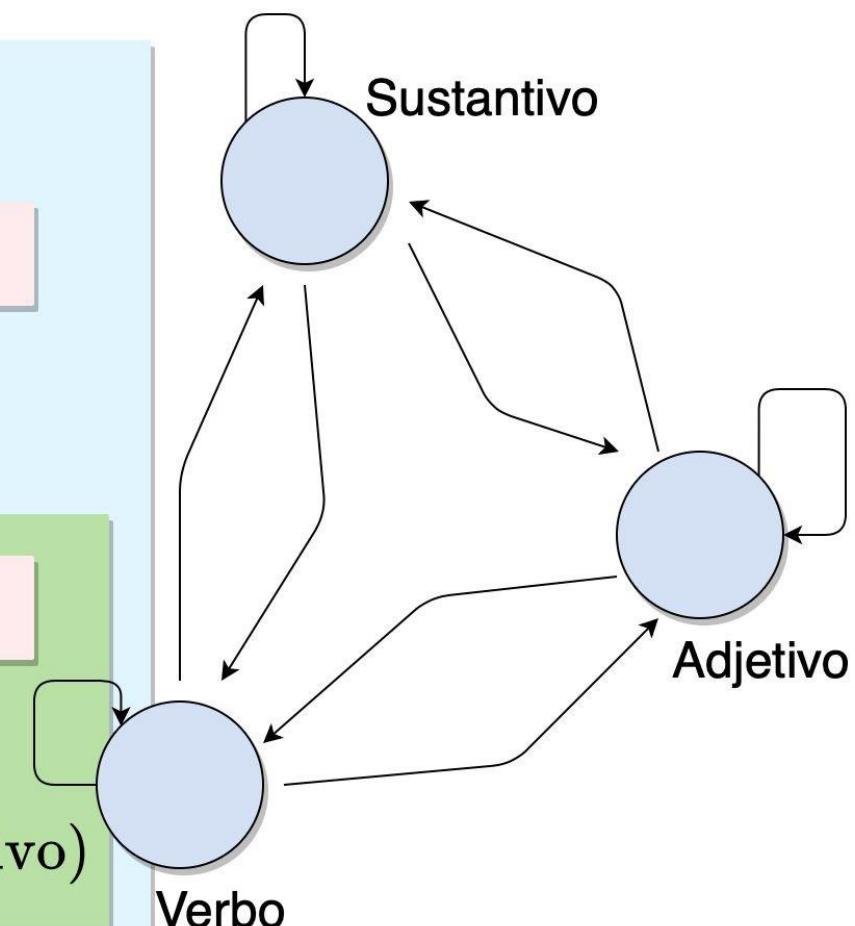
Ahora el modelo markoviano latente tiene como objetivo descubrir cual es esa cadena de etiquetas que le corresponderá a la secuencia de palabras.

HMM: Hidden Markov Model

Sequencia de palabras

$$(w_1, w_2, \dots) = \\ (\text{Pedro}, \text{es}, \text{ingeniero})$$

Sequencia de etiquetas

$$(t_1, t_2, \dots) = \\ (\text{Sustantivo}, \text{Verbo}, \text{Sustantivo})$$


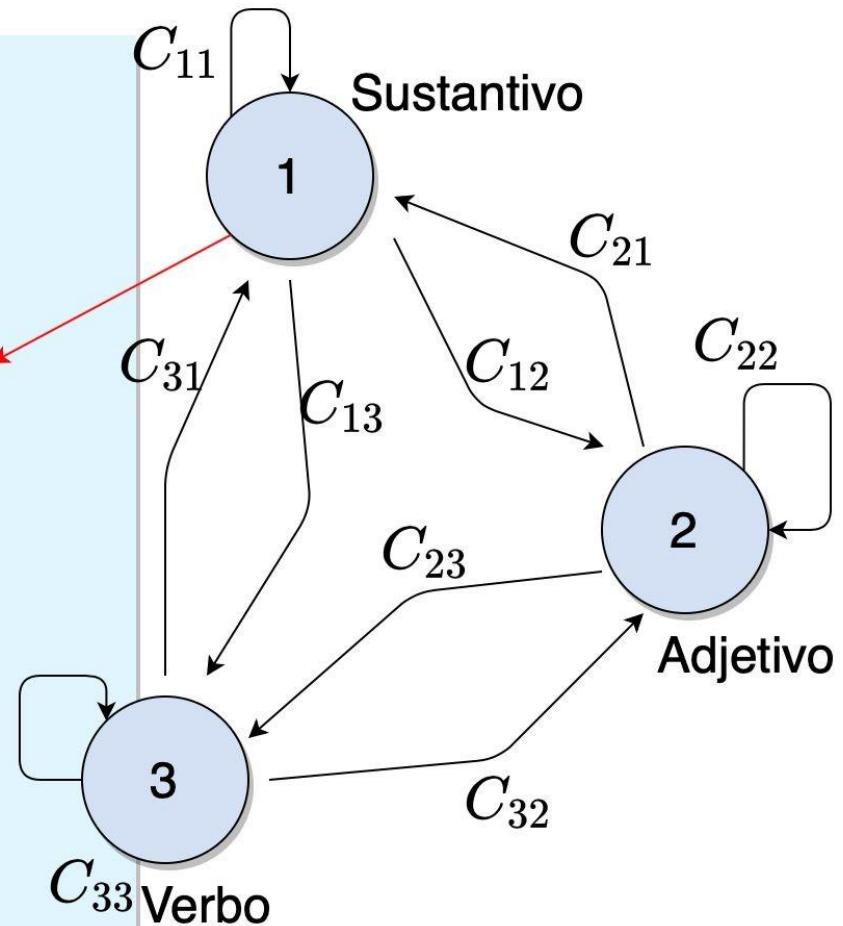
Modelos Markovianos

Probabilidades de emisión

$$(w_1, w_2, \dots) \rightarrow B_i = P(w_i | t_i)$$

$$\begin{aligned} P(\text{Pedro} | \text{Sustantivo}) \\ P(\text{es} | \text{Sustantivo}) \\ \dots \end{aligned}$$

$$(t_1, t_2, \dots) \rightarrow C_{ij} = P(t_i | t_{i-1})$$



Modelos Markovianos

Ingredientes de HMM

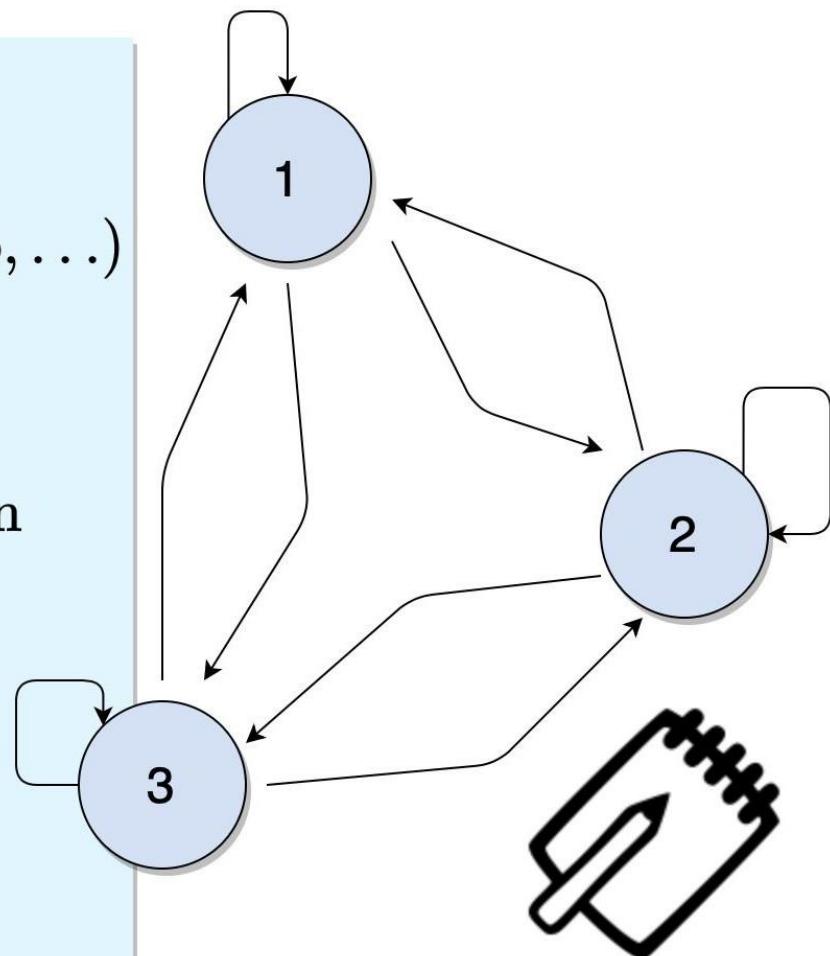
$(t_1, t_2, \dots) = (\text{sustantivo}, \text{verbo}, \dots)$

A : matriz transición

Π : distribución de estados

B : probabilidades de emisión

$\arg \max_{(t^n)} P(t^n | w^n)$



Dado que ahora tenemos una cadena de markov sobre unos estados, que aunque son latentes son estados, sobre ellos podemos definir probabilidades de transición, los cuales están en el recuadro verde C_{ij} los cuales serán la probabilidad de cambiar de una categoría gramatical a que en la siguiente palabra haya otra categoría gramatical.

Ahora las probabilidades serán, dada una categoría grammatical cual es la probabilidad de que esa categoría le corresponda a una cierta palabra como en el ejemplo en el recuadro blanco.

El nodo 1 que es sustantivo puede corresponder a varias palabras, puede ser "pedro", puede ser "es", o en algunos casos no aplica, pero entonces ahí las probabilidades son cero, y eso se calcula con un corpus de palabras, esas probabilidades que son dada una categoría grammatical cual es la probabilidad de que le corresponda una cierta palabra las llamamos probabilidades de emisión y podemos calcular un conjunto de probabilidades de emisión para cada uno de los nodos del mapa de cadena markoviana y eso junto todo corresponde a un modelo markoviano latente de manera que ahora tenemos 3 ingredientes para un HMM:

- Matriz de transición
- Distribución inicial de estados
- Probabilidades de emisión.

Con estos tres ingredientes el objetivo de una cadena de markov latente es encontrar dada una secuencia de palabras cual es la secuencia de etiquetas que le corresponde por mayor probabilidad

Tenemos un modelo caracterizado por dos objetos, una matriz de transiciones (A) y otro elemento que contiene las probabilidades de emisión (B).

La entrada de un modelo serán observaciones (O) secuencias de palabras observadas, y la salida serán las categorías asignadas(q).

Cual es la probabilidad (P) de que dada una secuencia de palabras (w^n) cual será la probabilidad de que una secuencia de etiquetas o tags (t^n) le sea asignada, nuestro objetivo es encontrar la cantidad maxima t^n

$$\begin{aligned} \text{HMM} &\rightarrow (A, B) \\ O &\rightarrow \{O_1, O_2, \dots, O_T\} \\ ? &\rightarrow \{q_1, q_2, \dots, q_T\} \end{aligned}$$

$$\arg \max_{t^n} P(t^n | w^n) = \tilde{t}^n$$

Para calcular esta probabilidad usaremos la **regla de Bayes**.


$$\begin{aligned} \text{HMM} &\rightarrow (A, B) \\ O &\rightarrow \{O_1, O_2, \dots, O_T\} \\ ? &\rightarrow \{q_1, q_2, \dots, q_T\} \end{aligned}$$
$$\arg \max_{t^n} P(t^n | w^n) = \tilde{t}^n$$

Regla de Bayes

$$P(t|w) = \frac{P(w|t)P(t)}{P(w)}$$

$$P(t^n | w^n) = P(w^n | t^n)P(t^n)$$

Introduciremos dos hipótesis fundamentales

El máximo de la probabilidad a la izquierda se traduce en el producto de las dos probabilidades de la derecha, ahora es necesario introducir dos hipótesis

$$\tilde{t}^n = \arg \max_{t^n} P(t^n | w^n) = \arg \max_{t^n} P(w^n | t^n) P(t^n)$$

En la hipótesis **Markoviana** el estado actual depende del estado inmediato anterior (ejemplo del clima)

En la hipótesis de **independencia**, las probabilidades de palabras a etiquetas solamente dependen de la misma posición.

hip. Independencia $P(w^n | t^n) = \prod_{i=1}^n P(w_i | t_i)$

hip. Markoviana $P(t^n) = \prod_{i=1}^n P(t_i | t_{i-1})$

$$\tilde{t}^n = \arg \max_{t^n} P(t^n | w^n) = \arg \max_{t^n} P(w^n | t^n) P(t^n)$$

hip. Independencia $P(w^n | t^n) = \prod_{i=1}^n P(w_i | t_i) = P(w_1 | t_1) P(w_2 | t_2) \times \dots$

hip. Markoviana $P(t^n) = \prod_{i=1}^n P(t_i | t_{i-1}) = P(t_1 | t_0) P(t_2 | t_1) \times \dots$

Con esto la formula que sigue un HMM es:

$$\tilde{t}^n = \arg \max_{t^n} \prod_{i=1}^{n-1} P(w_i | t_i) P(t_i | t_{i-1})$$



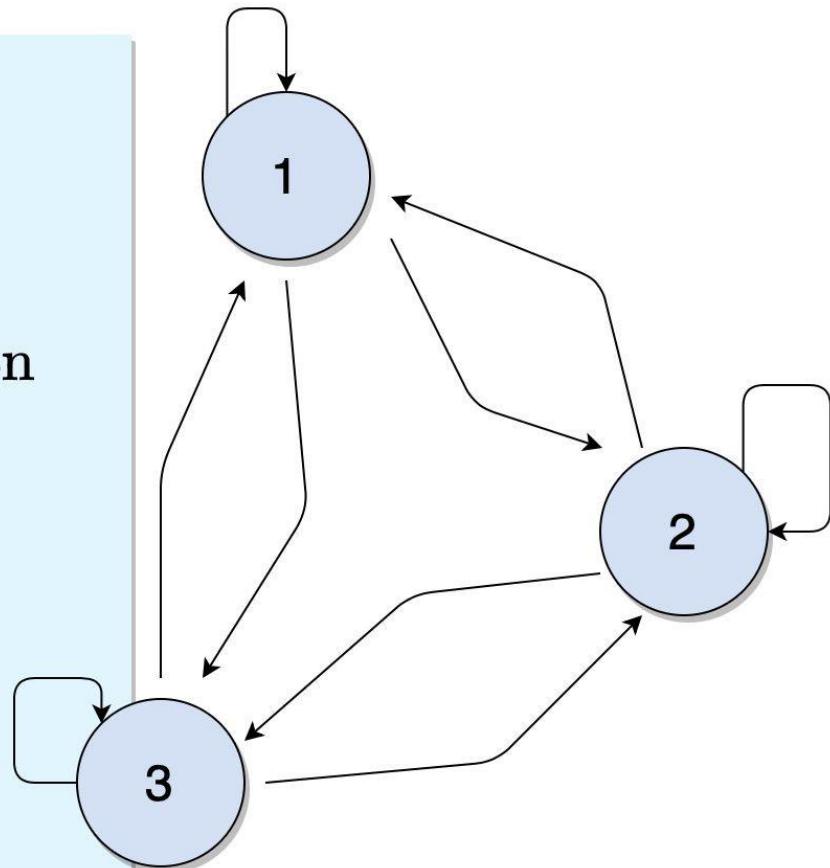
¿Qué significa entrenar un HMM?

Aprender probabilidades ...

A : matriz transición

B : probabilidades de emisión

(A, B)





[C9] El algoritmo de Viterbi

Desambiguación y etiquetado
de palabras

Etiquetado con HMM

En la clase pasada entrenamos un modelo Markoviano latente calculando las probabilidades de transición y emisión a partir de un corpus de textos en español, la idea es ahora entender como podemos usar este modelo para hacer predicciones.

La pregunta es ¿que tipo de predicciones hace un modelo markoviano latente HMM)?

- **Entrenamiento del HMM**

El modelo aprende las probabilidades de emisión y transición.

- **Decodificación**

El modelo calcula la secuencia de etiquetas más probable.

$$\widetilde{t^n} = \arg \max_{(t^n)} \prod_i \underbrace{P(w_i | t_i)}_{emisión} \overbrace{P(t_i | t_{i-1})}^{transición}$$

Etiquetado con HMM

- **Entrenamiento del HMM**

El modelo aprende las probabilidades de emisión y transición.

- **Decodificación**

El modelo calcula la secuencia de etiquetas más probable.

$$\widetilde{t^n} = \arg \max_{(t^n)} \prod_i \underbrace{P(w_i | t_i)}_{emisión} \overbrace{P(t_i | t_{i-1})}^{transición}$$

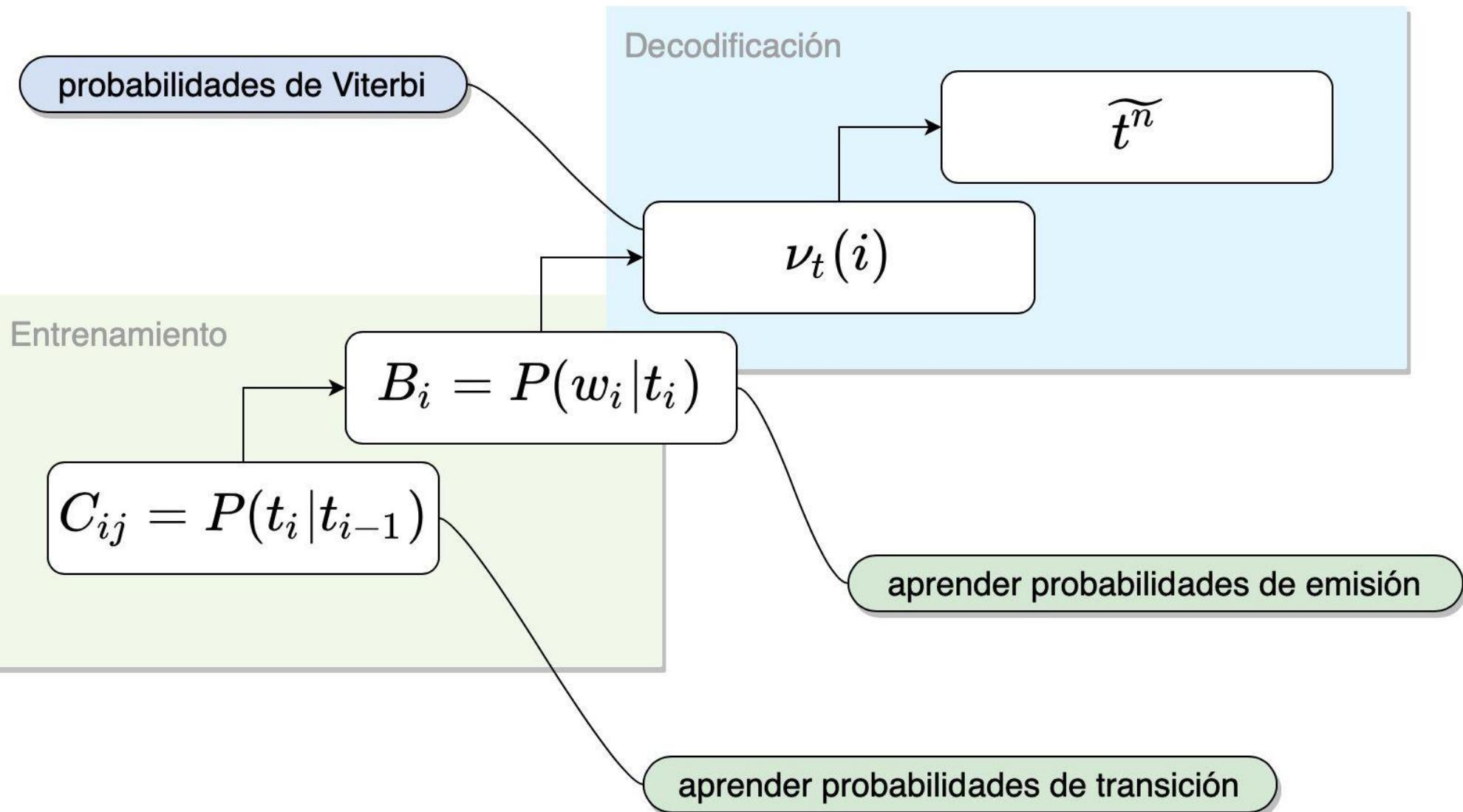
Una vez entrenado, el proceso que denominaremos Decodificación consiste en que dada una secuencia de palabras podamos identificar la secuencia de etiquetas gramaticales mas probable que le corresponda, y esto se hace mediante el algoritmo de Viterbi.

Hay otras alternativas pero primero enfoquemos en este.

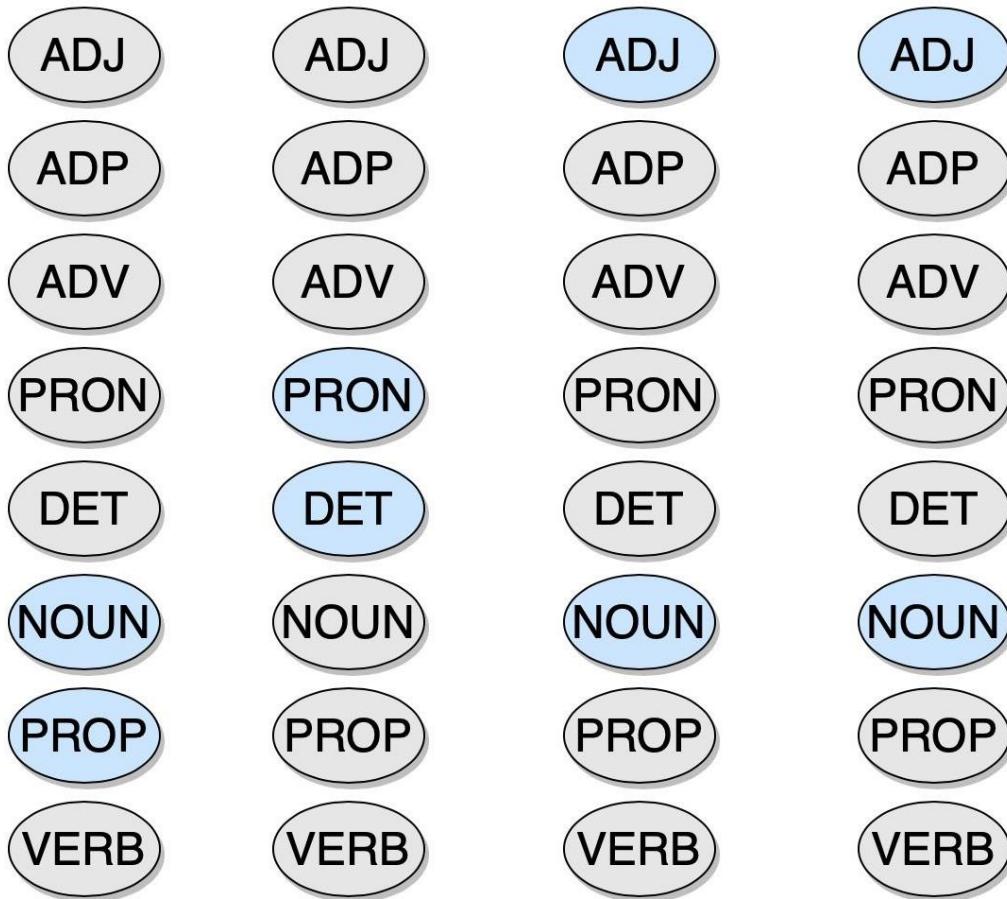
La parte de entrenamiento que programamos consiste en encontrar la matriz A con sus coeficientes C y luego las probabilidades Emisión que son los B dados las probabilidades condicionales (word | tag), luego viene el algoritmo de Viterbi que se va a encargar de encontrar de entre un montón de secuencias la secuencia mas probable esto lo hace asignándole una probabilidad a cada secuencia que llamaremos probabilidad de Viterbi, luego dentro de ese espacio de probabilidades escogemos la mayor y esa seria la que el algoritmo va a retornar como la mas probable y por lo tanto la que debería ser las etiquetas correctas de la secuencia de palabras.

El algoritmo de Viterbi funciona de la siguiente manera.

Etiquetado con HMM



Decodificación de Viterbi

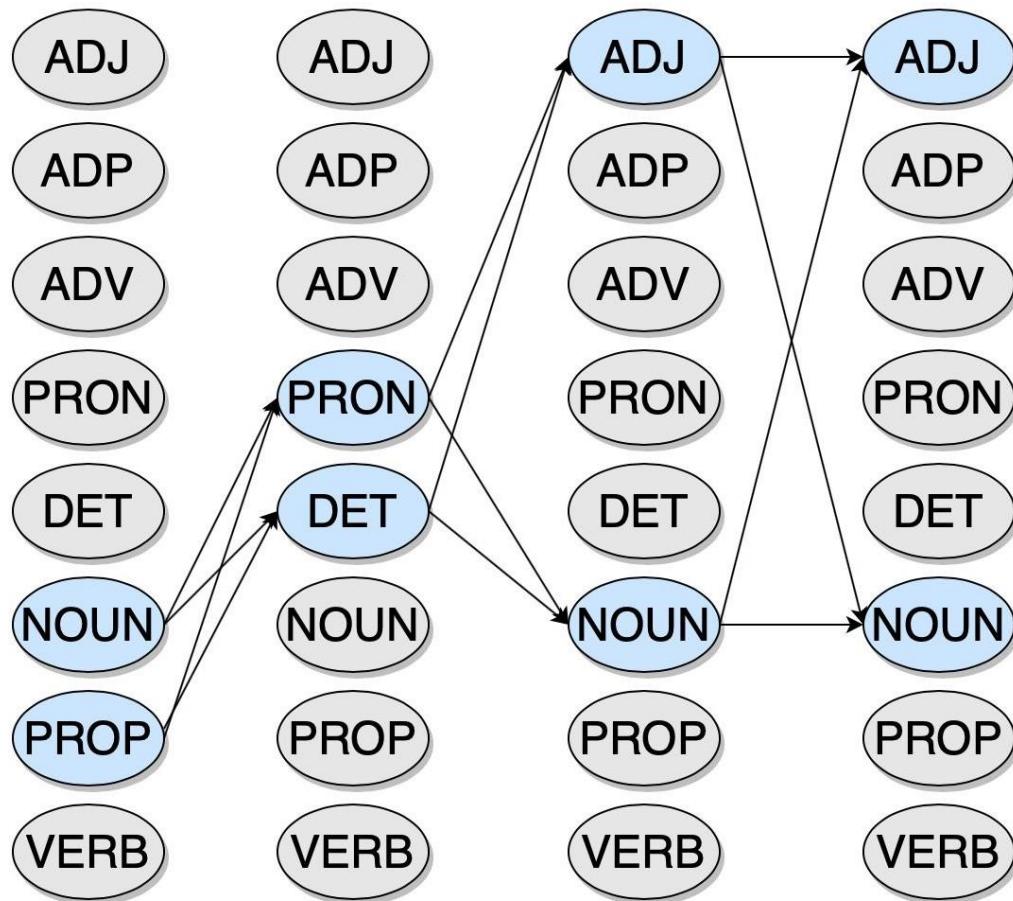


Cada columna son todas las posibles etiquetas que una palabra va a tener, castillo es una persona no un edificio un sustantivo, cada circulo corresponde a una posible categoría gramatical, los círculos en gris tienen una probabilidad cero.

¿Como esto nos ayuda a entender el algoritmo de Viterbi? de la siguiente manera.

Castillo el noble trabajador

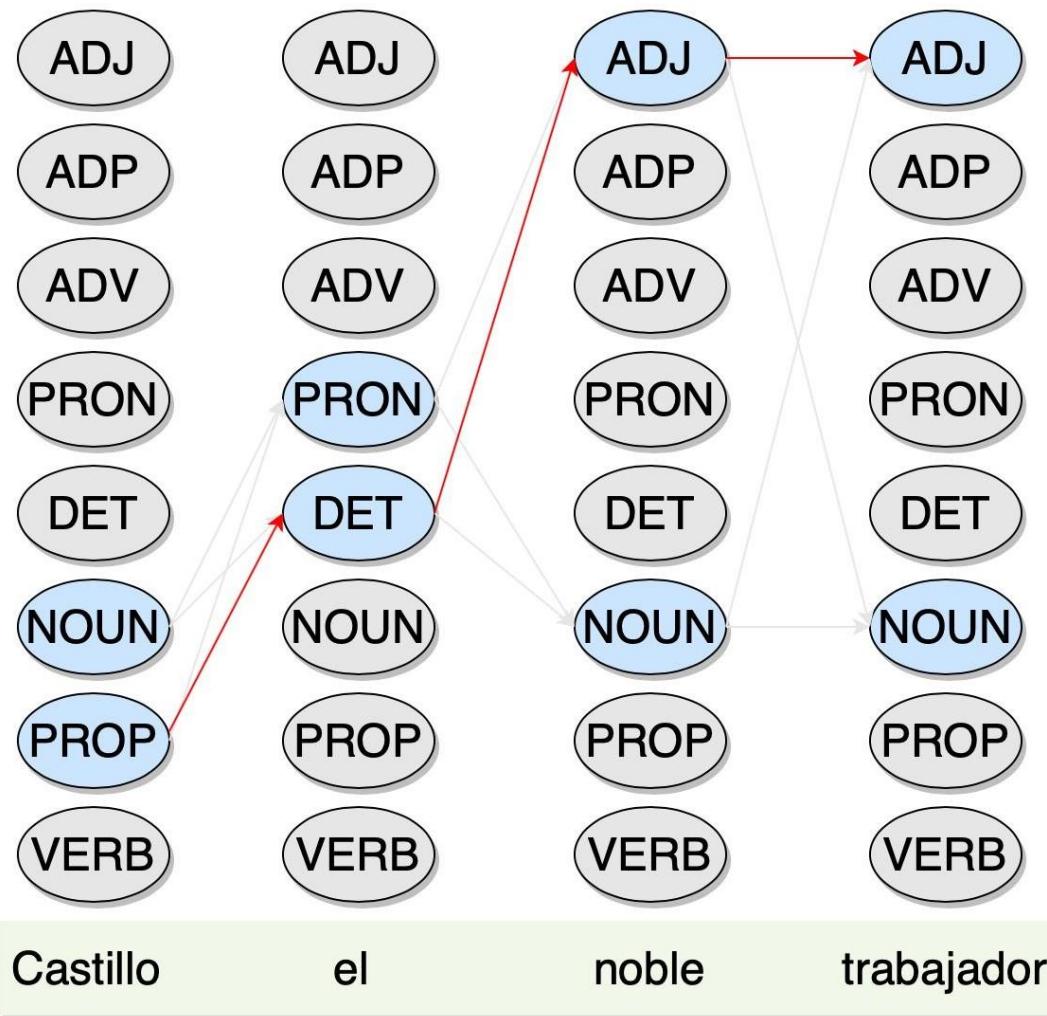
Decodificación de Viterbi



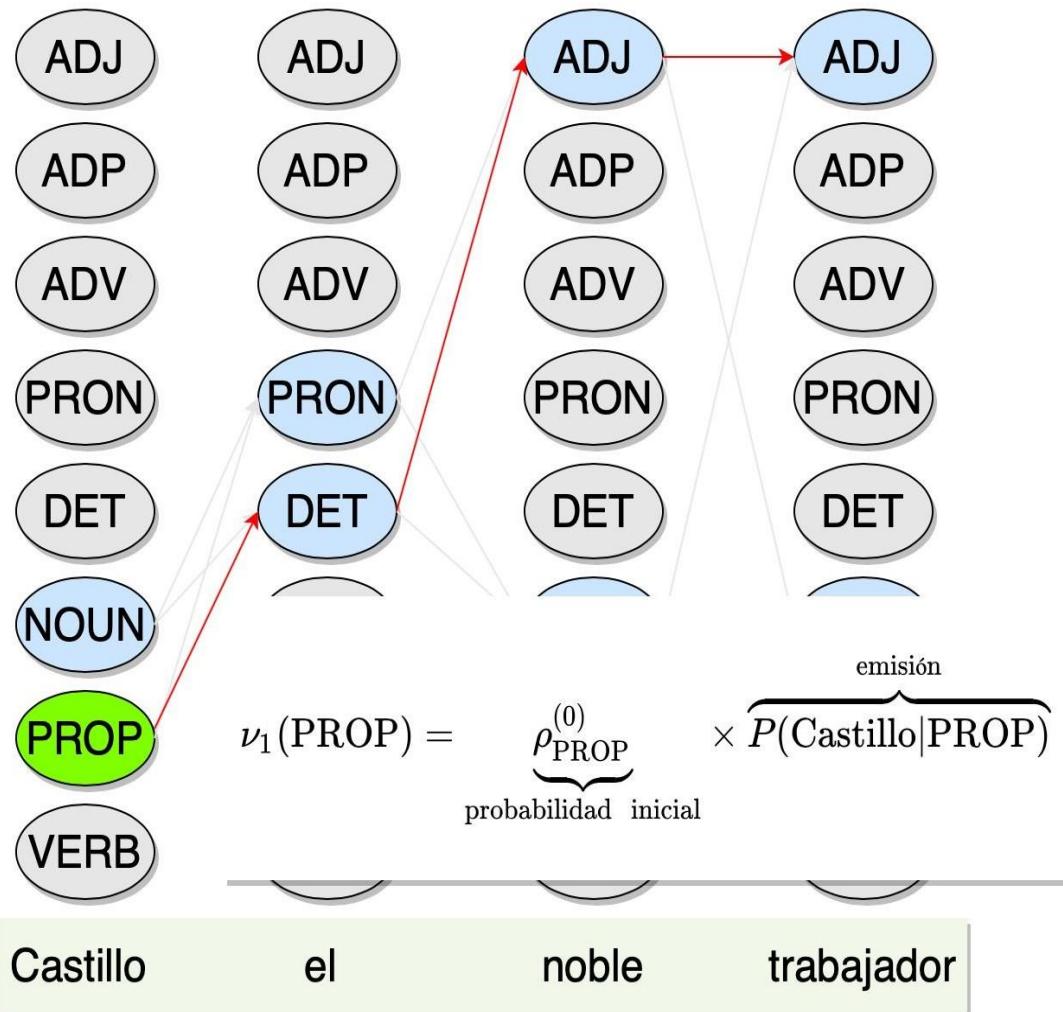
Considerando todas las posibles categorías gramaticales de cada palabra vamos creando caminos creando las etiquetas posteriores, cada camino es recorrer la primer etiqueta posible hasta la siguiente posible y asi hasta llegar a la ultima palabra que contiene la secuencia. De todos esos caminos hay que calcular el mas probable, eso lo hacemos calculando un numero probabilístico que me diga que tan probable es que sea uno de esos caminos y escoger el mayor, eso lo hacemos de la siguiente manera.

Castillo el noble trabajador

Decodificación de Viterbi

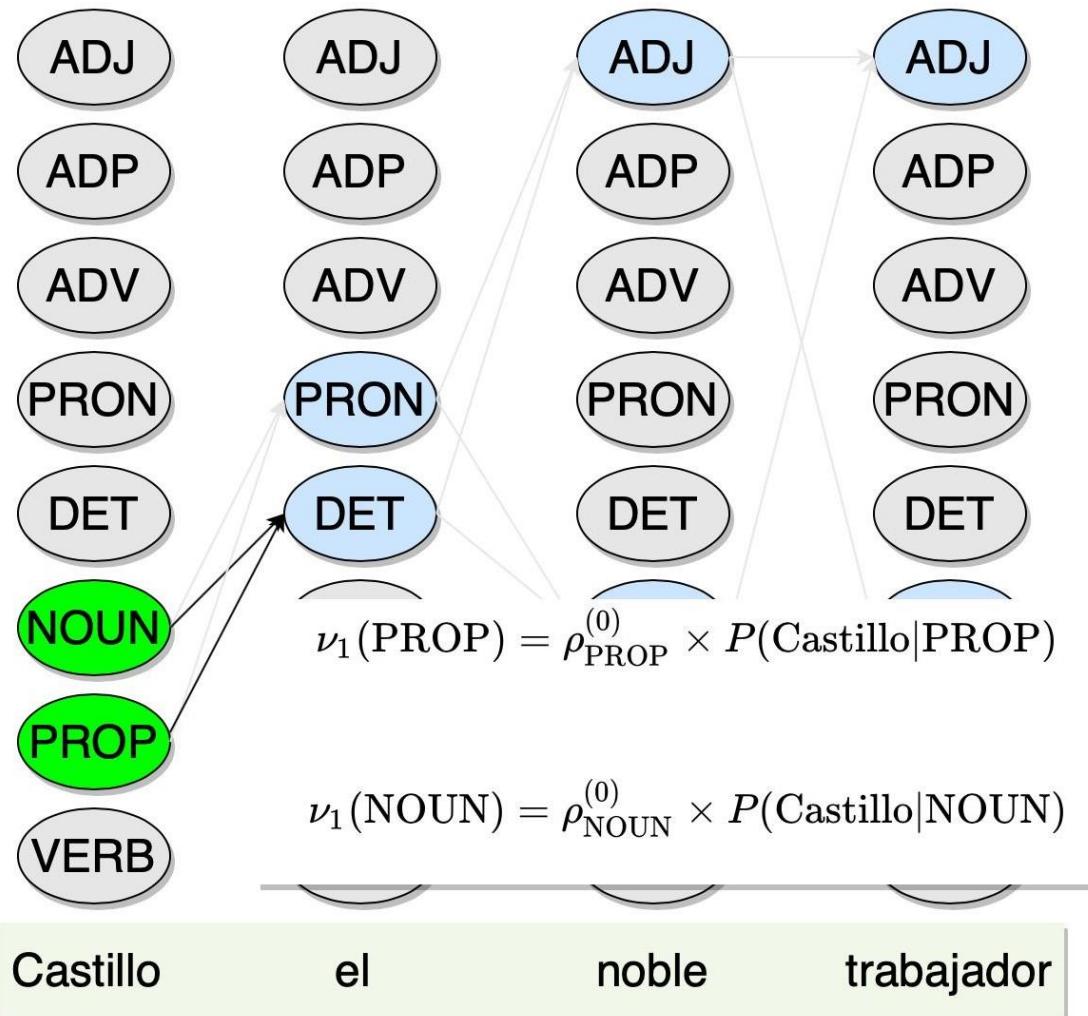


Decodificación de Viterbi

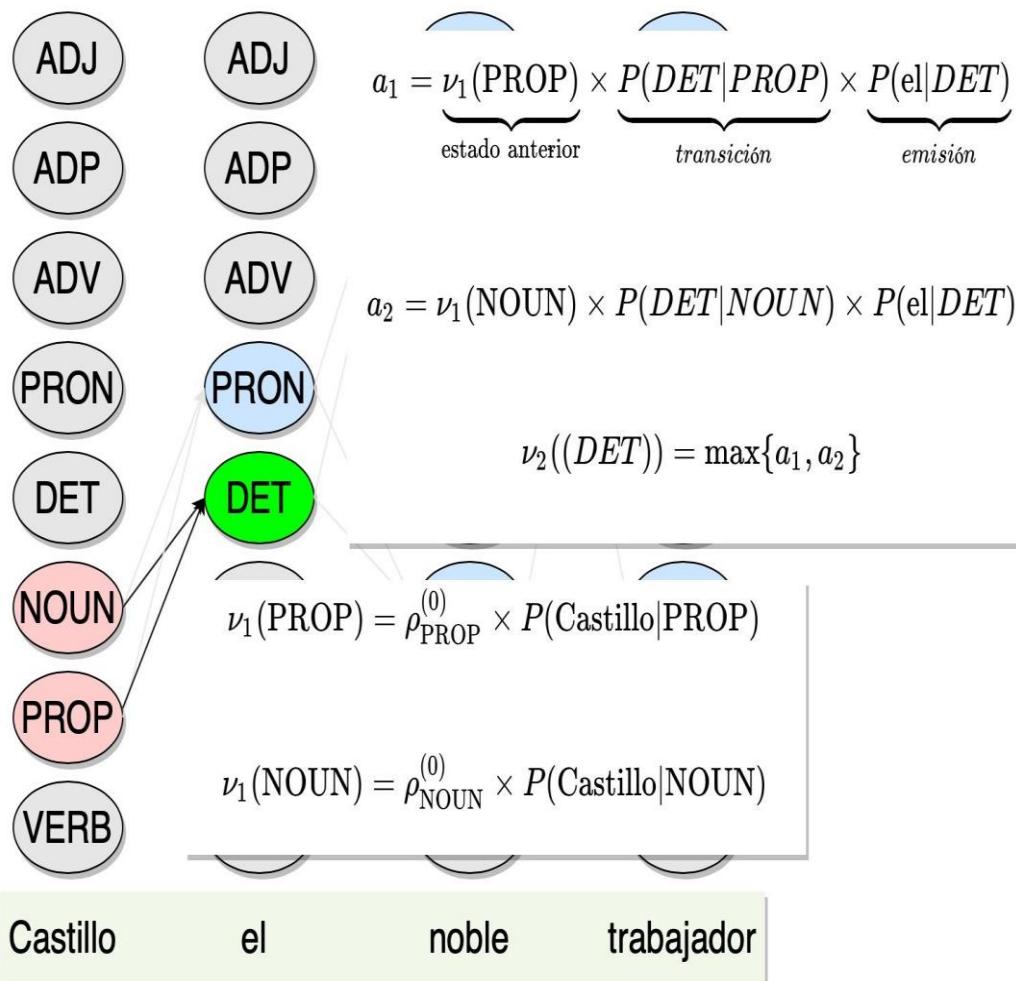


El círculo de sustantivo propio esta en color verde, significa que vamos a analizar lo que sucede en este nodo en particular, vamos a denotar con la letra griega NU, que parece una letra V paréntesis prop estilizada probabilidad de viterbi de que la categoría gramatical de castillo sea "PROP" y eso es igual al "producto de la probabilidad inicial" este multiplicado por una "probabilidad condicional" de que ya que la categoría Inicial es PROP la palabra que este ubicada ahí es castillo, ese calculo lo hacemos para cada una de las celdas de esta columna, de esta manera calculamos la probabilidad de Viterbi para cada uno de los nodos de la primera columna.

Decodificación de Viterbi



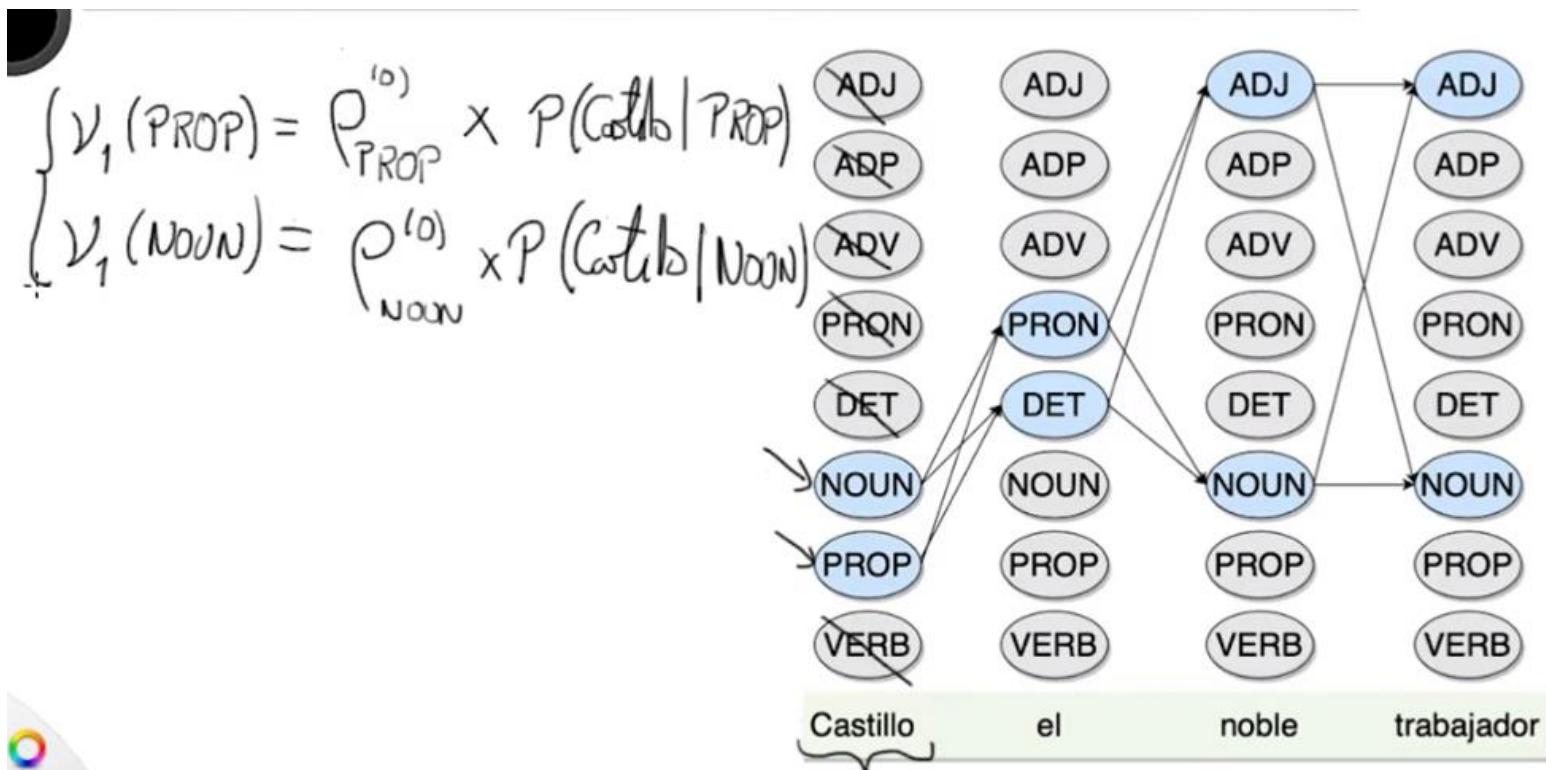
Decodificación de Viterbi



El único nodo que tiene probabilidad no nula en la segunda columna es el de categoría determinante (DET), para calcular la probabilidad de este nodo lo que vamos a hacer es considerar todos los posibles caminos que pasan por ese nodo, vemos que son dos NOUN-DET Y PROP-DET, cada uno de los números tiene una probabilidad asignada que consiste en tomar la probabilidad del estado anterior ($\nu_1(\text{PROP})$), multiplicar por la probabilidad condicional de la etiqueta anterior y cual sera la etiqueta siguiente que en este caso sera de que dado prop la siguiente sea DET y esto multiplicado por una probabilidad de emisión, que dado que la categoría es DET cual sera la palabra el, y que tan probable es eso, aplicamos lo mismo a otra ruta con NOUN, y de esos dos números calculamos la probabilidad y tomamos esa como la nueva probabilidad de Viterbi del nodo que esta en verde en este momento, y asi subsecuentemente para todos los nodos en la columna, el proceso estará completo cuando hayamos calculado las probabilidades de cada uno de los elementos de esta matriz.

En la clase anterior vimos como funciona el algoritmo de Viterbi, este asigna probabilidades a cada elemento en una matriz que combina filas de categorías gramaticales y columnas de palabras en una secuencia, la idea es que con este algoritmo que con estas probabilidades el puede determinar cual es la secuencia de etiquetas mas probable para esa secuencia que nosotros le estamos dando al modelo como entrada. Veamos ahora como en profundidad son estos cálculos de probabilidades y entender la matriz mencionada.

Retomamos el ejemplo anterior donde castillo es el apellido de una persona, donde los nodos contienen las etiquetas gramaticales, pero solo los azules tienen probabilidad no nula, de esta manera en la primer columna solo vamos a calcular 2 probabilidades que corresponden con los nodos que indican la categoría de sustantivo (noun) o sustantivo propio (prop).

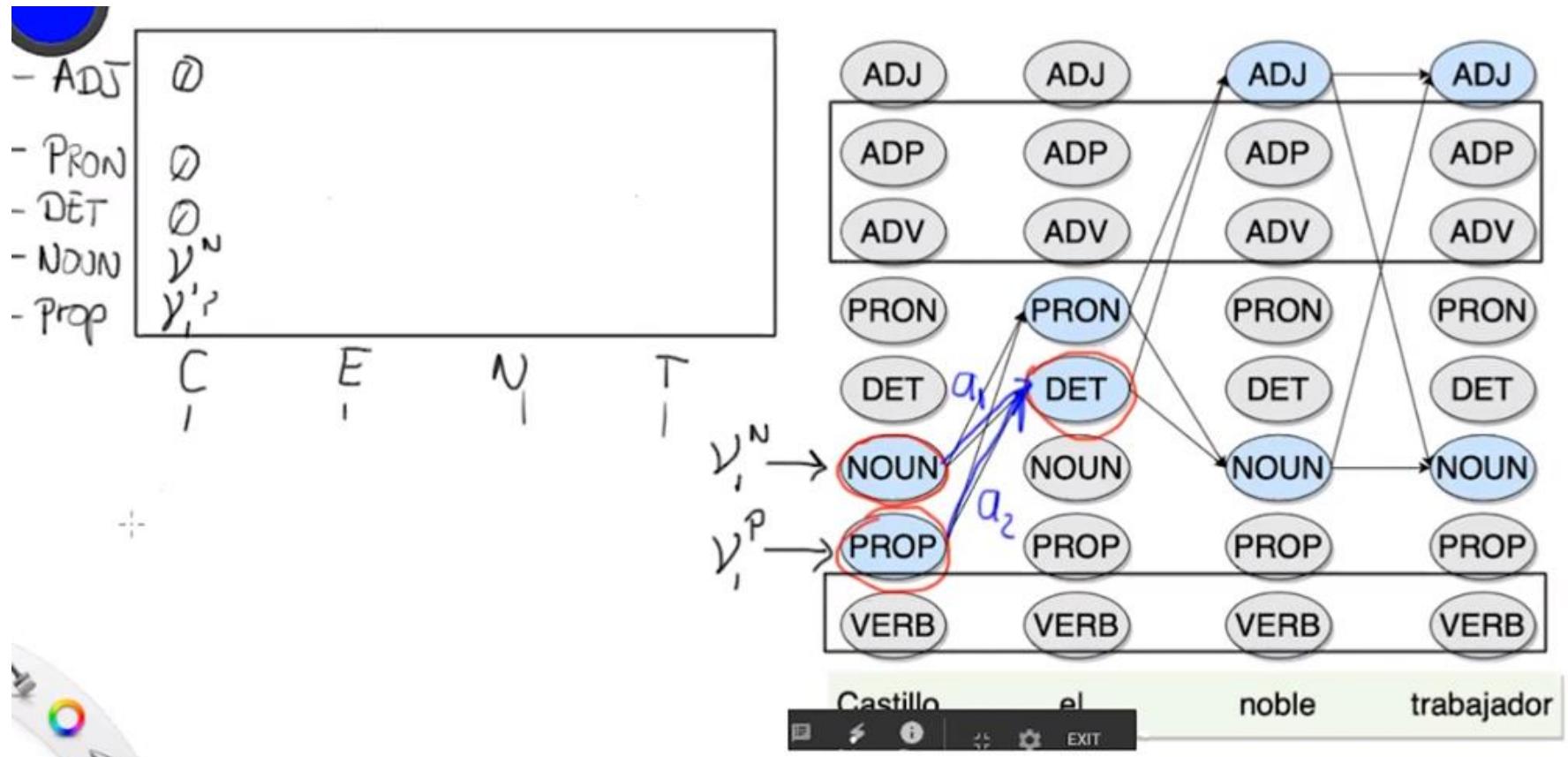


Una vez calculadas las probabilidades de cada columna, podemos empezar a visualizar la matriz, donde cada palabra de la frase sea una columna, y las filas serán las etiquetas gramaticales.

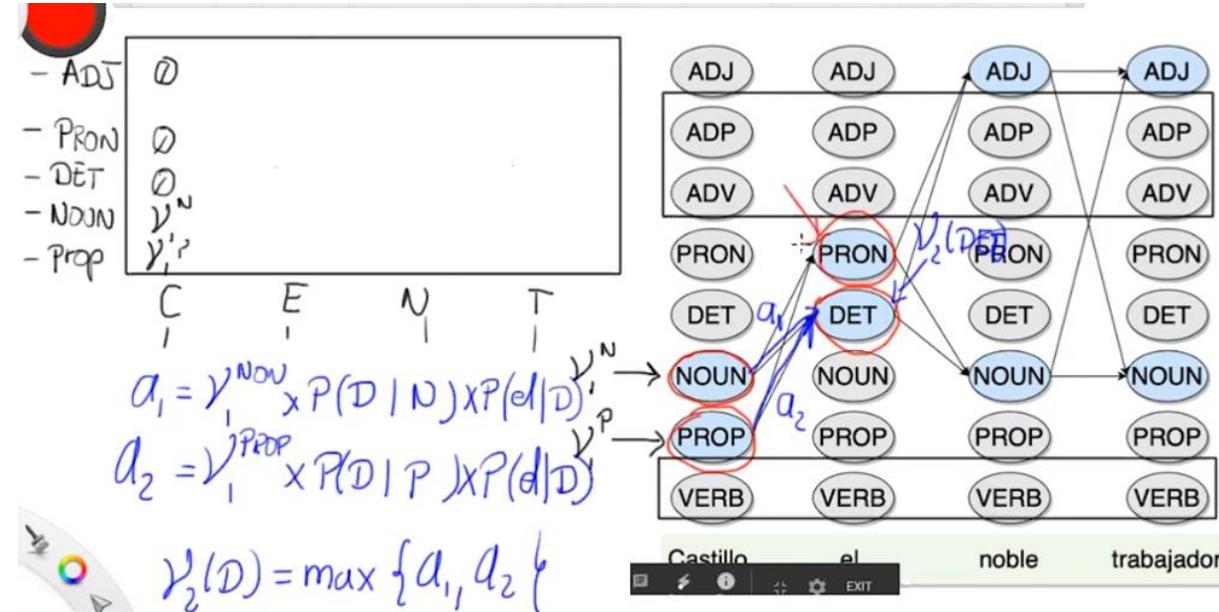
- ADJ	Ø
- PRON	Ø
- DET	Ø
- NOUN	γ^N
- Prop	γ^P

C	E	N	T
I	I	I	I

El siguiente paso es asignar las probabilidades a la columna 2 primer elemento en sus dos caminos para llegar al nodo



La probabilidad a seria igual a Probabilidad de viterbi de la etiqueta por la probabilidad de transición, por la probabilidad de emisión.



Mientras que la probabilidad de Viterbi V2 sera la mayor entre a1 y a2, lo siguiente es aplicar el mismo concepto al nodo PRON, hecho esto asignamos los valores a la matriz

- ADJ	Ø	Ø	..
- PRON	Ø	V_2^P	..
- DET	Ø	V_2^D	..
- NOUN	Y^N	Ø	..
- Prop	Y'_1'	Ø	..
C	E	N	T

El algoritmo de viterbi termina cuando hemos calculado las probabilidades de todos los elementos de esta matriz.

En resumen el algoritmo de Viterbi mediante la búsqueda de posibles caminos de etiquetas calcula una probabilidad a cada elemento de una matriz donde esas probabilidades las llamamos probabilidades de Viterbi, el objetivo de encontrar la secuencia mas probable consiste en encontrar el camino cuyas probabilidades de Viterbi son mas grandes.

Decodificación de Viterbi

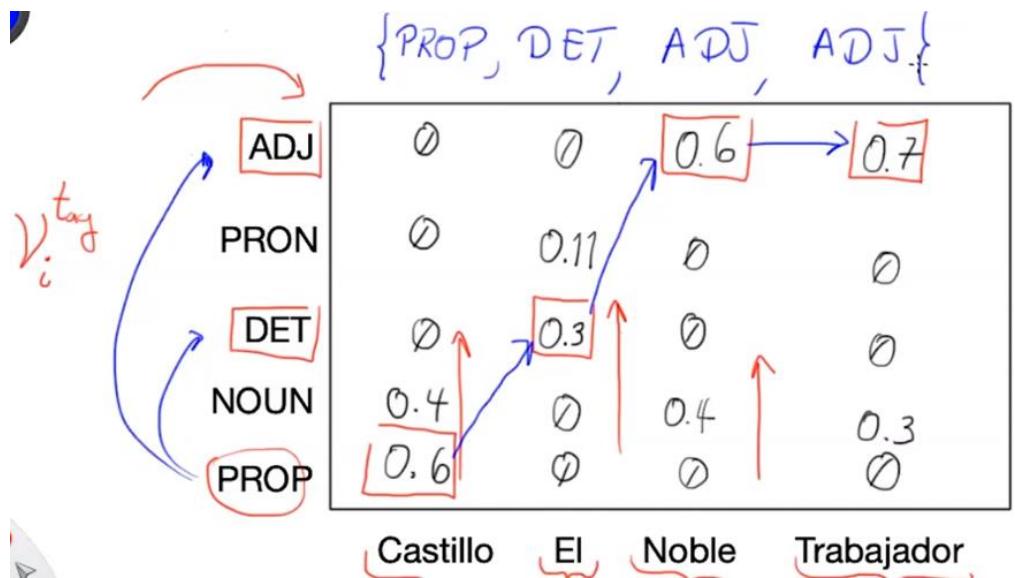
$$\tilde{t^n} = \arg \max_{(t^n)} \prod_i \underbrace{P(w_i|t_i)}_{emisión} \overbrace{P(t_i|t_{i-1})}^{transición}$$

$$\nu_t(j) = \max_i \{\nu_{t-1}(i) \times C_{ij} \times P(\text{palabra}|j)\}$$

Una vez calculadas todas las probabilidades de viterbi de todas las etiquetas posibles la matriz nos queda de la siguiente manera.

ADJ	0	0	0.6	0.7
PRON	0	0.11	0	0
DET	0	0.3	0	0
NOUN	0.4	0	0.4	0.3
PROP	0.6	0	0	0

Castillo El Noble Trabajador

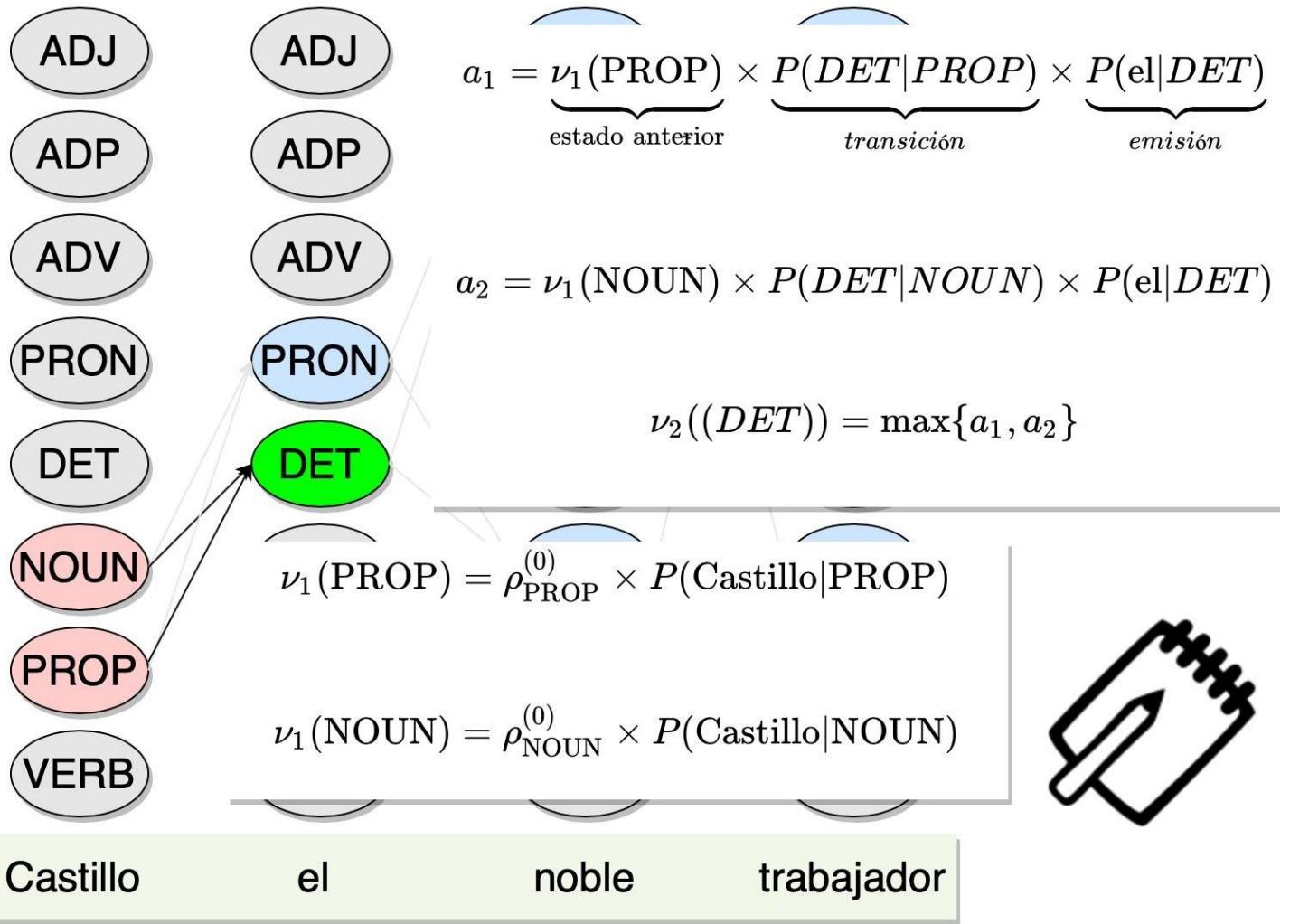




[C10] Cálculo de las probabilidades de Viterbi

Desambiguación y etiquetado de palabras

Decodificación de Viterbi





Decodificación de Viterbi

$$\tilde{t^n} = \arg \max_{(t^n)} \prod_i \underbrace{P(w_i | t_i)}_{emisión} \overbrace{P(t_i | t_{i-1})}^{transición}$$

$$\nu_t(j) = \max_i \{\nu_{t-1}(i) \times C_{ij} \times P(\text{palabra}|j)\}$$

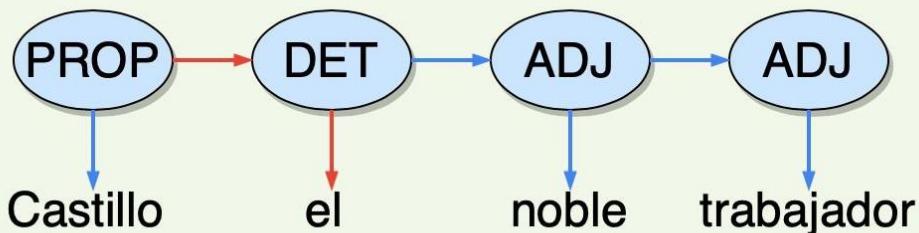


[C14] Modelos Markovianos de Máxima Entropía

Desambiguación y etiquetado
de palabras

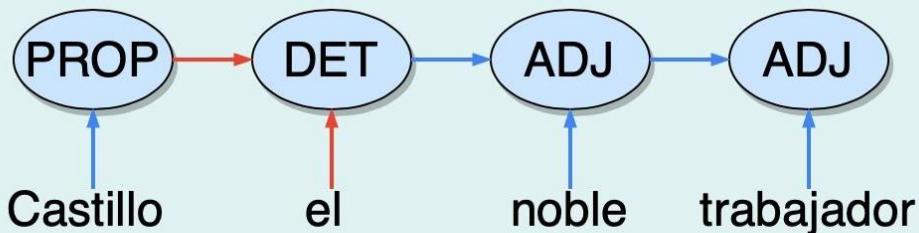
HMM

$$\tilde{t}^n = \arg \max_{(t^n)} \prod_i P(w_i | t_i) P(t_i | t_{i-1})$$



En este punto tenemos los conocimientos suficientes para entender diversos algoritmos de clasificación, específicamente de etiquetado que tienen cierta afinidad o similitud con los modelos markovianos latentes, la idea de esta sección es que el profesor propone un reto en el cual tendrás que usar todo lo aprendido hasta este momento pero desarrollando código basado en un nuevo modelo propuesto en esta clase, derivado de los HMM.

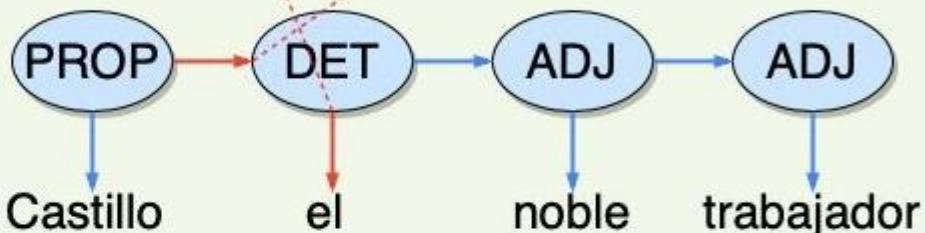
MEMM



$$\tilde{t}^n = \arg \max_{(t^n)} \prod_i P(t_i | w_i, t_{i-1})$$

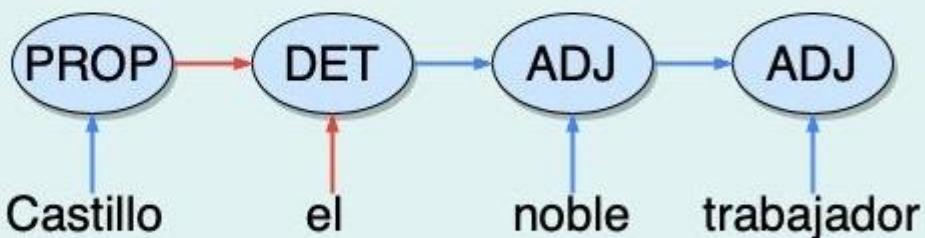
HMM

$$\tilde{t^n} = \arg \max_{(t^n)} \prod_i P(w_i | t_i) P(t_i | t_{i-1})$$



En un modelo Markoviano latente ya conocemos la formula, la secuencia mas probable de etiquetas se calcula como aquella donde dada una secuencia de palabras cual es la secuencia de etiquetas mas probable, usábamos bayes para convertir probabilidad condicional en el producto de dos probabilidades que llamábamos probabilidades de transición y probabilidades de emisión representados con flechas rojas en HMM

MEMM

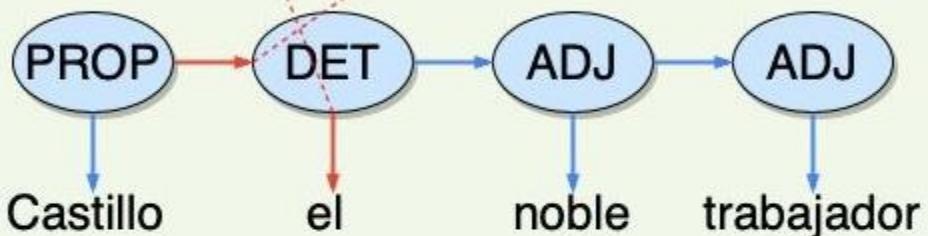


El nuevo modelo hace una diferencia, las etiquetas siguen teniendo flechas que van dirigidas desde la etiqueta anterior a la posterior, pero las flechas que conectan palabras y etiquetas van hacia arriba (de la palabra a la etiqueta).

$$\tilde{t^n} = \arg \max_{(t^n)} \prod_i P(t_i | w_i, t_{i-1})$$

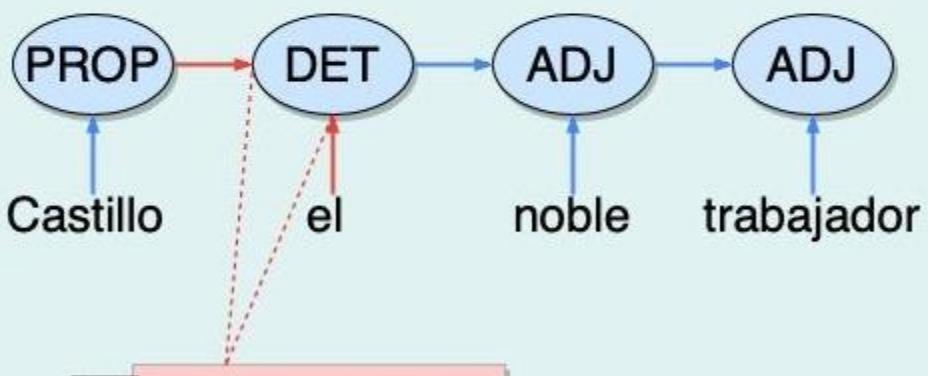
HMM

$$\tilde{t^n} = \arg \max_{(t^n)} \prod_i P(w_i | t_i) P(t_i | t_{i-1})$$



En resumen un Modelo Markoviano de Maxima Entropia se define como idéntico a HMM pero no usamos la regla de bayes para convertir esto en probabilidades de transición y emisión, sino que la probabilidad inicial es la única que vamos a considerar directamente y vamos a crear nuevas dependencias.

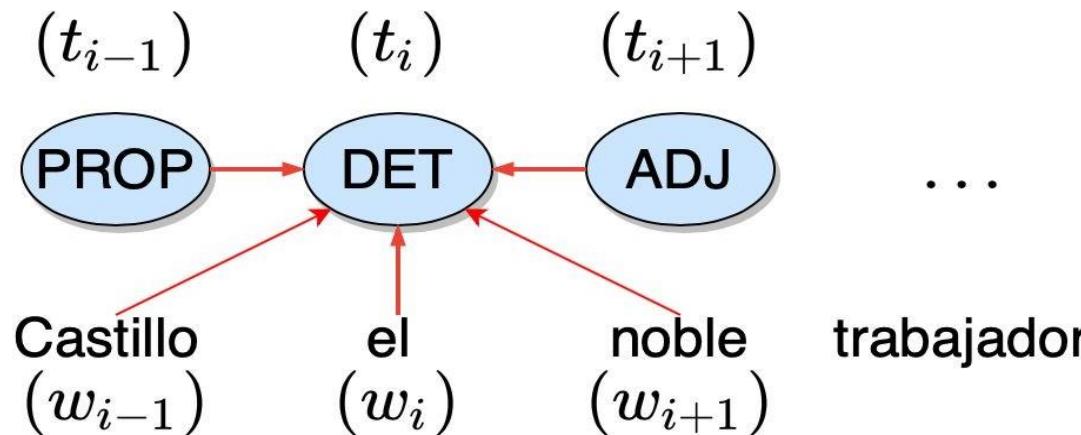
MEMM



$$\tilde{t^n} = \arg \max_{(t^n)} \prod_i P(t_i | w_i, t_{i-1})$$

En la formula inferior estamos calculando la probabilidad de que dada una palabra en la posición i, y una etiqueta en la posición i-1, cual será la etiqueta en la posición i, sin embargo esta probabilidad no se descompone en transición - emisión como en el HMM, esta es una util pero gran diferencia, el performance del modelo será distinto.

En el siguiente diagrama vemos la forma en que funciona un Modelo Markoviano de Maxima Entropia con dependencias tan arbitrarias como se deseé.



$$\tilde{t^n} = \arg \max_{(t^n)} \prod_i P(t_i | w_{i-1}, \dots, w_{i+1}, t_{i-1}, t_{i+1})$$

En un HMM estamos calculando la probabilidad de que dada una palabra le corresponda una cierta etiqueta (probabilidades posteriores en bayes) y esto lo descomponímos en la probabilidad de transición (dada una etiqueta, cual será la que le corresponde en la siguiente posición) y la probabilidad de emisión (dada una etiqueta cual será la palabra que le corresponda), aquí debemos pensar de forma distinta con probabilidades posteriores, la probabilidad seria la siguiente:

Dado un contexto alrededor de una etiqueta particular que yo quiero calcular o determinar cual será la probabilidad de que esa categoría sea un verbo/sustantivo/etc.

Esto quiere decir que tomaremos información de las etiquetas anterior y posterior con las palabras que están antes, y después, con esto caemos en el concepto de las redes neuronales donde tenemos muchas entradas y la red debe inferir una salida, haciendo uso de modelos de clasificación.

La formula nos indica que la secuencia de palabras a la cual le vamos a asignar una secuencia de etiquetas donde la probabilidad sea máxima esta dado solo por esa probabilidad, donde dada una palabra actual, palabra posterior y palabra anterior, etiqueta posterior, y etiqueta anterior quiero saber la etiqueta actual, lo cual es un contexto completo.

COMPARACIÓN ENTRE MODELOS:

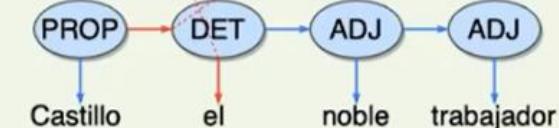
HMM:

$$P(t_i | t_{i-1}) = \frac{C(t_i | t_{i-1})}{C(t_{i-1})}$$

$$P(w_i | t_i) = \frac{C(t_i | w_i)}{C(t_i)}$$

HMM

$$\tilde{t}^n = \arg \max_{(t^n)} \prod_i P(w_i | t_i) P(t_i | t_{i-1})$$



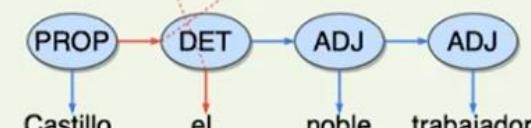
MEMM:

Cual es la probabilidad de que dada una palabra, y una etiqueta anterior, cual es la probabilidad de que corresponda una etiqueta en la posición actual, deberá ser un conteo donde observo la palabra y las dos etiquetas consecutivas, dividido entre el numero de veces que veo la palabra junto con la etiqueta en la posición inmediatamente anterior.

$$P(t_i | w_i, t_{i-1}) = \frac{C(w_i, t_i, t_{i-1})}{C(t_i, t_{i-1})}$$

HMM

$$\tilde{t}^n = \arg \max_{(t^n)} \prod_i P(w_i | t_i) P(t_i | t_{i-1})$$



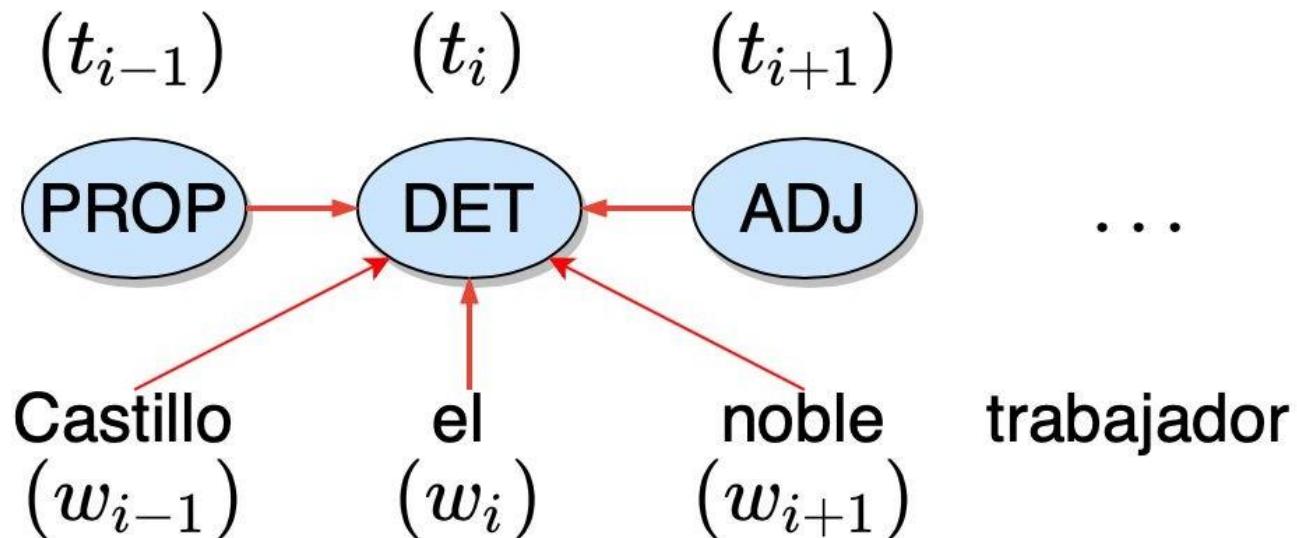
Esa es la forma en que deberíamos calcular matemáticamente las probabilidades para el MEMM





[C15] Algoritmo de Viterbi para MEMM

Desambiguación y etiquetado de palabras



$$\widetilde{t^n} = \arg \max_{(t^n)} \prod_i P(t_i | w_{i-1}, \dots, w_{i+1}, t_{i-1}, t_{i+1})$$

En este slide vimos como construir un MEMM

Donde yo puedo predecir la categoría a la que pertenece cierta palabra considerando todo el contexto que rodea a esa categoría en términos de las categorías y palabras que se encuentran a los costados, y la palabra que corresponde a esa categoría

Decodificación de Viterbi

$$\widetilde{t^n} = \arg \max_{(t^n)} \prod_i P(t_i | w_i, t_{i-1}, \dots)$$

$$\nu_t(j) = \max_i \{\nu_{t-1}(i) \times P(j|palabra, i, \dots)\}$$

Teniendo en cuenta que solo calculamos probabilidades posteriores, dado un contexto de palabras y etiquetas cual es la probabilidad de que le corresponda una cierta etiqueta. Ya no utilizamos probabilidades de transición ni emisión solo una probabilidad posterior, y el algoritmo de Viterbi tiene que adaptarse a esa filosofía, la formula debajo del slide indica como seria: "la probabilidad de Viterbi para la columna t para una categoría j es igual al máximo de todas las posibles probabilidades donde cada una de esas probabilidades es el producto de la probabilidad de Viterbi de la columna anterior t-1 de una categoría i multiplicado por la probabilidad posterior, que dado ese contexto cual debe ser la categoría j que debería corresponder a la palabra.

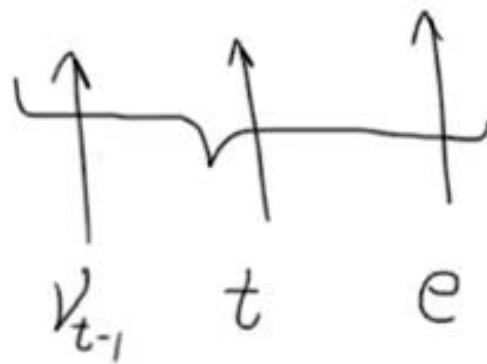
HMM

$$\tilde{t^n} = \arg \max_{(t^n)} \prod_i \underbrace{P(w_i | t_i)}_{emisión} \overbrace{P(t_i | t_{i-1})}^{transición}$$

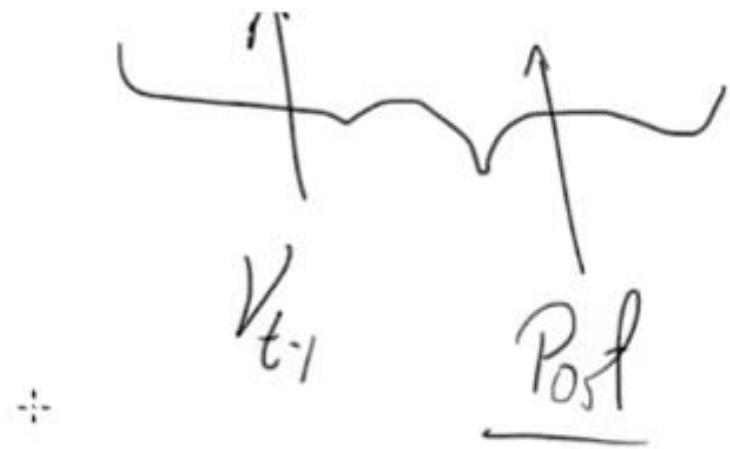
MEMM

$$\tilde{t^n} = \arg \max_{(t^n)} \prod_i P(t_i | w_i, t_{i-1}, \dots)$$

$$\nu_t(j) = \max_i \{\nu_{t-1}(i) \times C_{ij} \times P(\text{palabra}|j)\}$$



$$\nu_t(j) = \max_i \{\nu_{t-1}(i) \times P(j|\text{palabra}, i, \dots)\}$$



Si observas todo es muy similar al algoritmo y código anterior, donde eliminamos la probabilidad de transición y en lugar de la probabilidad de emisión calculamos la probabilidad posterior de dado un contexto nos da la categoría



[C16] RETO 1: Construye un **MEMM en Python**

Desambiguación y etiquetado
de palabras

```
import nltk
```

```
... .
```

```
import pickle
```

```
... .
```

“

**Vamos a un notebook
en Google Colab ...**



”



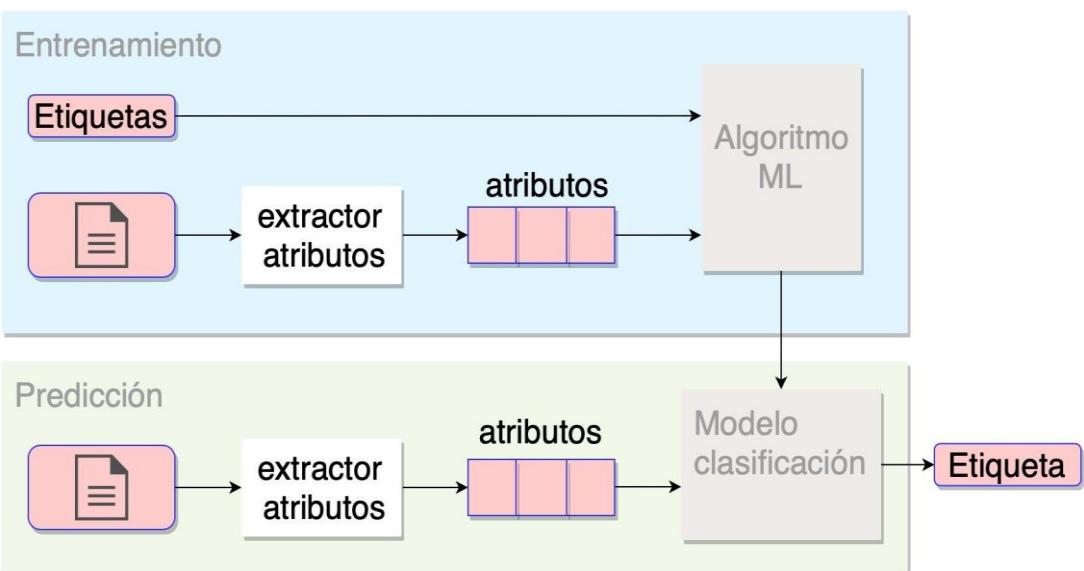
[C17] El problema general de la clasificación de texto

Clasificación de texto

Clasificación: ML supervisado

La tarea general de clasificación en Machine learning es mas amplio que etiquetar palabras por categoría gramatical (un caso particular).

El clasificado en ML se observa en el siguiente diagrama.



En general tu tienes un documento con texto donde extraes atributos para predecir (etiquetas) categorías de clasificación (tema de conversación, sentimiento, prioridad,) la idea del algoritmo de IA es hacer un preprocessamiento para extraer atributos, una vez procesados y vectorizados hasta cierto punto con el algoritmo lo entrenamos para que sepa que categoría o etiquetas corresponden de acuerdo a los data points de ese conjunto o documento, el algoritmo se entrena de forma supervisada.

Una vez terminada la fase de entrenamiento sigue y tenemos formalmente un modelo de clasificación, podemos tomar otro documento que el algoritmo no conoce y aplicamos el mismo procesamiento para extraer atributos y a partir de ellos, en teoría, si el modelo es bueno, el debe predecir la etiqueta de ese documento, de esta manera es como funciona la clasificación en general.

Técnicas de clasificación

- Basadas en teoría de la probabilidad
- Basadas en teoría de la información
- Basadas en espacios vectoriales

Clasificación de palabras

- Identificación de género de nombres
- Etiquetado POS (categorías gramaticales): ya tratadas
- Bloqueo de palabras ofensivas

Clasificación de documentos

- Análisis de sentimiento
- Tópicos de conversación
- Priorización en CRMs



[C18] Tareas de clasificación con NLTK

Clasificación de texto

Clasificación de género

Por palabra

EN la clase pasada vimos que existen dos grandes categorías para la clasificación de texto en cuanto a la granularidad (palabras y documentos).

Para la clasificación por palabras podemos asignar categorías, clasificarlas por genero, etiquetas gramaticales, etc.

Los documentos los clasificamos con análisis de sentimientos, respecto al tópico, o tema de conversación, o priorización (spam no spam).



```
import nltk, random
from nltk.corpus import names

def atributos(palabra):
    return { 'letra_f': palabra[-1] }

tagset = [
    (n, 'M'/'F') for n in names.words('.txt')
]

random.shuffle(tagset)
...
```

Primer Ejercicio

Nuestra primer tarea será la clasificación de palabras para determinar el genero, para ello tomaremos esta plantilla de pseudo código en Python

En nuestro caso definimos una función sencilla ya que la clasificación de nombres será un caso de prueba y error, nos retorna la ultima letra del nombre.

Luego de eso hacemos una lista de tuplas tomando todas las palabras de nuestro archivo de texto.

Creamos una lista random shuffle a nuestra lista para tener iguales probabilidades de que la lista tenga ambos géneros, ya que al inicio primero serán masculinos los nombres seguido de todos los nombres femeninos.

Una vez que tenemos nuestra lista de tuplas, donde cada tupla es (nombre, genero) nos damos cuenta de que el algoritmo no es el que lee directamente esos datos, el algoritmo como vimos en la clase pasada toma los atributos del texto, esto nos obliga a crear otra lista, esta será igual en estructura pero ahora las tuplas son dos elementos donde ahora el primer elemento ya no es el nombre sino los atributos del nombre y el segundo será el genero.

Una vez que tenemos la lista la dividimos en train y test.

Una vez tenemos los datasets crearemos una instancia de un clasificador, sin embargo nuestro clasificador al haber utilizado solo un atributo no tendrá un performance muy bueno.

```
...  
  
fset = [  
    (atributos(n), 'g') for (n, 'g') in tagset)  
]  
  
train, test = fset[num:], fset[:num]  
  
classifier =  
    nltk.NaiveBayesClassifier.train(train)
```

Ingeniería de atributos



¿Cómo escoger los atributos relevantes de un nombre?

Es difícil saber a-priori saber cuales son los atributos de un objeto texto, puede ser un string muy largo que determine que esa palabra o documento caiga en una categoría especifica de manera optima para la mayoría de los casos, para todos los casos debemos utilizar prueba y error, y eso es conocido como ingeniería de atributos.

La pregunta central es como escoger los atributos mas relevantes.

El siguiente paso será definir una nueva función de atributos.

```
import nltk, random
from nltk.corpus import names

def atributos(palabra):
    '??'
    return { 'atr_1': ?, 'atr_2': ?, ... }

...
```

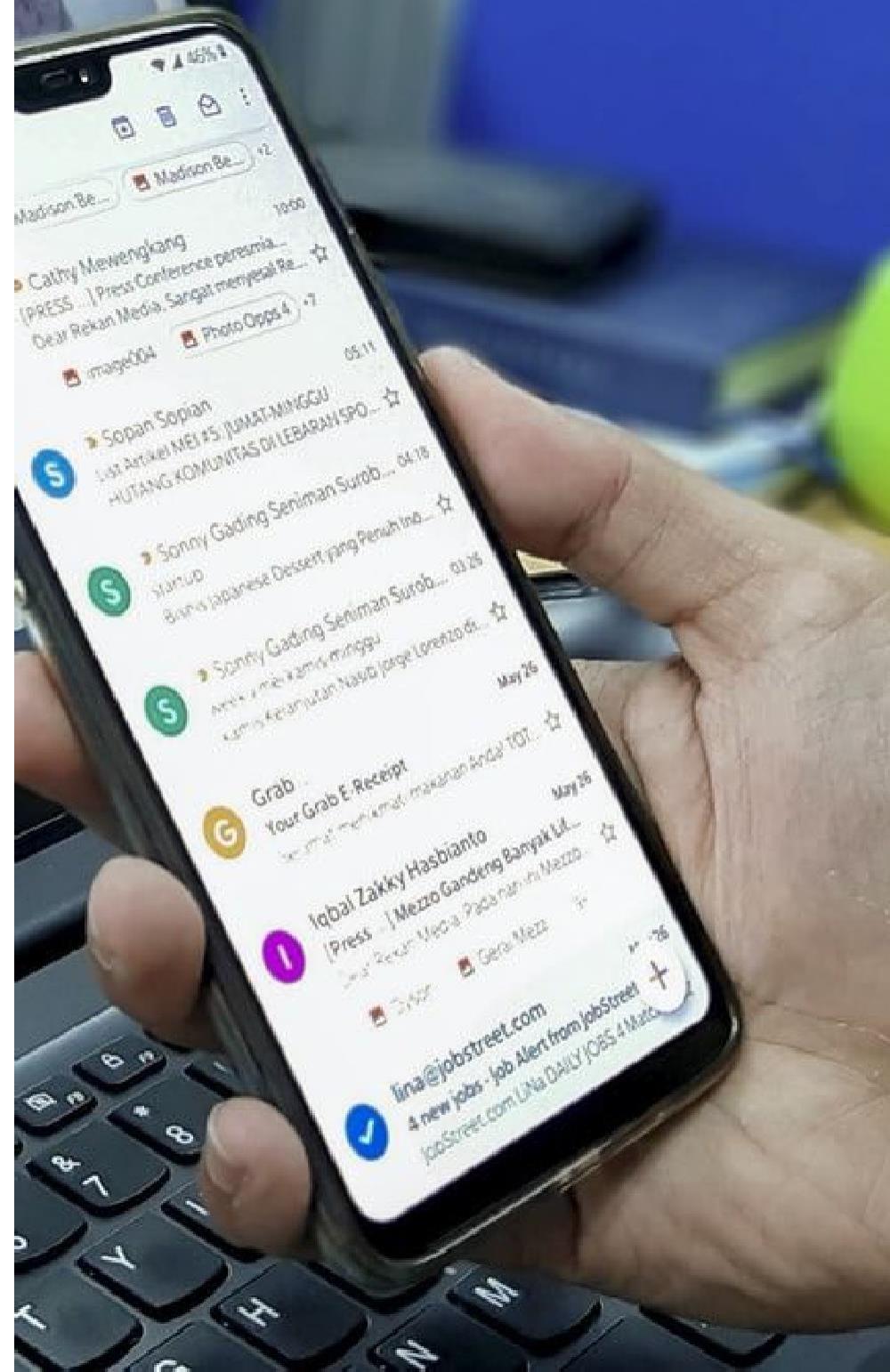
Esta función recibe una palabra pero aun no tenemos determinado que realizara, y el resultado será un diccionario con n atributos para ganar mas precisión.

Segundo Ejercicio

El segundo ejercicio será clasificar documentos que representan emails y las categorías de clasificación será si el correo es spam o no lo es.

Clasificació n de emails

Por documento



¡Alto!

Hasta el final del curso





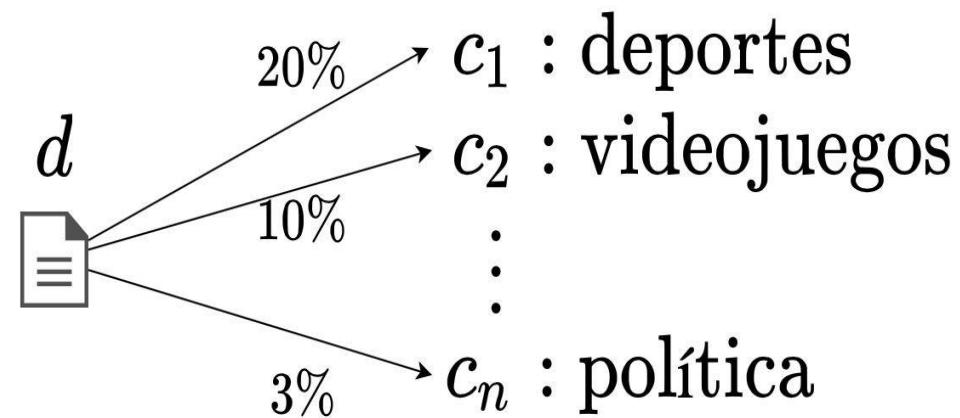
[C21] Naive Bayes

Clasificación de texto

NB: clasificador probabilístico

Vamos a profundizar la lógica matemática del algoritmo Naive Bayes, pero antes es necesario entender el concepto de clasificador probabilístico

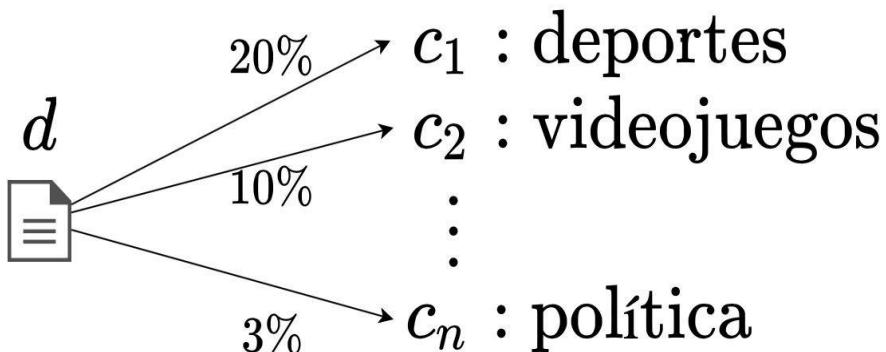
Dado un documento cualquiera el considera todas las categorías asignándoles una probabilidad, como vemos en la imagen un documento cualquiera puede estar refiriéndose a deportes, videojuegos o política, no lo sabemos, pero el modelo con la formula que observas calcula internamente una probabilidad para cada una de esas categorías, se interpreta de la siguiente manera:



La probabilidad condicional de que dado un documento "d" le corresponda una categoría "c", y al igual que con las cadenas de markov calculamos cual es la máxima de esas probabilidades, y la categoría correspondiente es la que predice el modelo porque es la que tiene la máxima probabilidad.

$$\hat{c} = \arg \max_{(c)} P(c|d)$$

NB: clasificador probabilístico



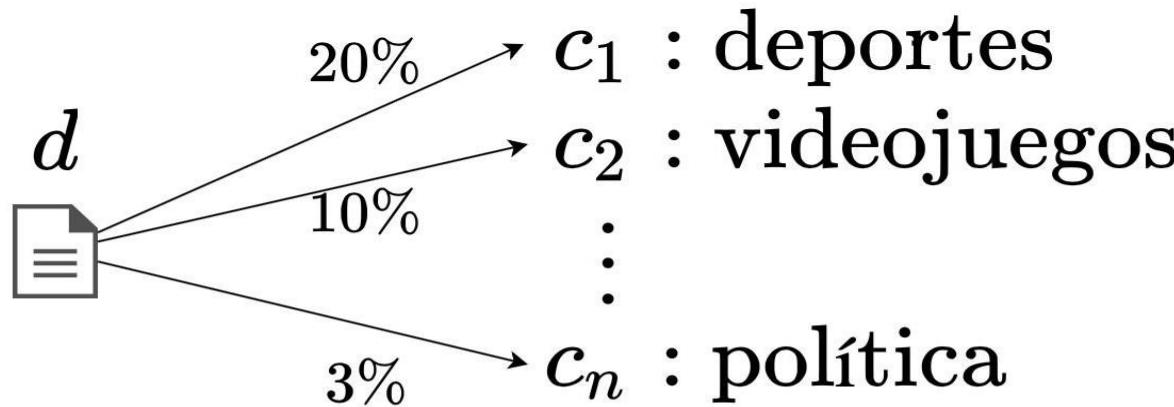
$$\hat{c} = \arg \max_{(c)} \underbrace{\frac{P(d|c)P(c)}{P(d)}}_{\text{Regla de Bayes}}$$

Por debajo, el modelo se llama Naive Bayes porque usa la regla de Bayes, con ella hace referencia a que la probabilidad llamada posterior se calcula en términos de otras probabilidades.

Como observamos en la imagen la idea es considerar **la posibilidad de que dada una categoría "c" le corresponda un documento "d"** por **la probabilidad de encontrar esa categoría "c" en el corpus de datos** y todo dividido por la probabilidad de encontrar el documento "d" en todo el corpus de datos nuevamente.

Aquí hay un truco adicional para en vez de calcular 3 probabilidades calcular solo 2 de ellas.

NB: clasificador probabilístico



$$\hat{c} = \arg \max_{(c)} \underbrace{P(d|c)P(c)}_{P(d) \text{ igual para toda clase}}$$

Esto porque estamos comparando el máximo de muchas probabilidades, y todas esas probabilidades están divididas por el mismo numero (la probabilidad del documento), lo único que cambia es la probabilidad de la categoría, (solo quitamos el denominador, el más grande sigue siendo el mas grande, el mas pequeño sigue siendo el mas pequeño), y matemáticamente la relación de orden se preserva reduciendo esto a el calculo de solo dos probabilidades, la primera es la Probabilidad condicional prior en la cual dada una categoría le corresponda un documento, la segunda es probabilidad de encontrar esa categoría en el corpus de datos, el problema de Naive Bayes se reduce a eso, ahora veamos como se desglosa. La probabilidad como vimos tenemos el producto de una distribución de probabilidad condicional, por el producto de una probabilidad sobre las categorías, la idea es encontrar la categoría donde su probabilidad es máxima, la pregunta a responder es como calcular esas probabilidades

$$P(d|c) = ?$$

$\underbrace{P(d|c)P(c)}_{\rightarrow} \rightarrow P(c) = ?$

Primero encontrar la probabilidad de que dada una categoría le corresponda un documento, y luego la probabilidad de encontrar la categoría en el corpus

La primera categoría es esta condicional, un documento estará compuesto por un conjunto de features "fn", estos serán atributos.

$$P(d|c) = P(f_1, f_2, \dots, f_n | c)$$

n - variables

Vamos a usar la "hipótesis de Naive Bayes" que consiste en decir que existe cierta independencia en la probabilidad conjunta, diciendo que en realidad el factor se puede reducir al producto de varias probabilidades.

$$P(d|c) = P(f_1, f_2, \dots, f_n | c) = P(f_1 | c)P(f_2 | c) \times \dots$$

n - variables

Así tomaremos el valor máximo de esa serie de cálculos de probabilidades, y ello lo podemos expresar con la siguiente fórmula utilizando PI mayúscula para denotar al conjunto de productos de probabilidad.

$$\hat{c} = \arg \max_{(c)} P(c) \prod_{i=1}^n P(f_i | c)$$

Sin embargo aun hay que tomar en cuenta un factor adicional, el elemento PI Mayúscula (productorio en matemáticas) puede significar que estamos multiplicando demasiados elementos, esto porque el numero de features n puede ser muy grande, y si nuestras probabilidades son números relativamente pequeños por ejemplo:

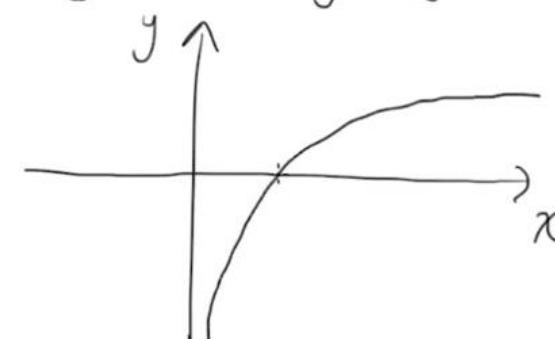
$$\hat{c} = \arg \max_{(c)} P(c) \prod_{i=1}^n P(f_i | c)$$

$n \rightarrow \text{grande}$
 $0.1 \times 0.2 \times 0.03$

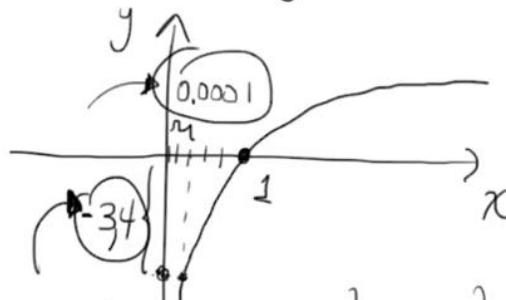
En general cuando multiplicas números menores a la unidad el numero resultante sera cada vez mas pequeño, computacionalmente no es bueno ya que la maquina puede llegar a interpretar estos números como un cero, para evitar utilizamos el truco del logaritmo.

$$\hat{c} = \arg \max_{(c)} P(c) \prod_{i=1}^n P(f_i | c)$$

$n \rightarrow \text{grande}$ $y = \log(x)$



Recuerda que en la función logaritmo los números cercanos a cero para y serán números negativos



$$\log(ab) = \log(a) + \log(b)$$

Y que también los logaritmos tienen propiedades. De esta forma la formula se convierte en lo siguiente.

$$\begin{aligned}
 & \log \left\{ P(c) \prod_i P(f_i | c) \right\} \\
 &= \log(P(c)) + \underbrace{\log \left(\prod_i (P(f_i | c)) \right)}_{= \log P(c) + \log P(f_1 | c) + \log(P(f_2 | c)) + \dots} \quad \log(ab) = \log(a) + \log(b)
 \end{aligned}$$

$$\textcircled{1} \quad \arg \max_{(c)} P(c) \prod_{i=1}^n P(f_i | c) = \log P(c) + \sum_{i=1}^n \log P(f_i | c)$$

Con la ultima expresión evitamos el fenómeno de underflow que es cuando sobrepasamos la precisión de la maquina utilizando números cada vez mas pequeños.

Solo nos falta recordar que cada una de estas propiedades se calcula haciendo conteos sobre el dataset como en los modelos anterior.

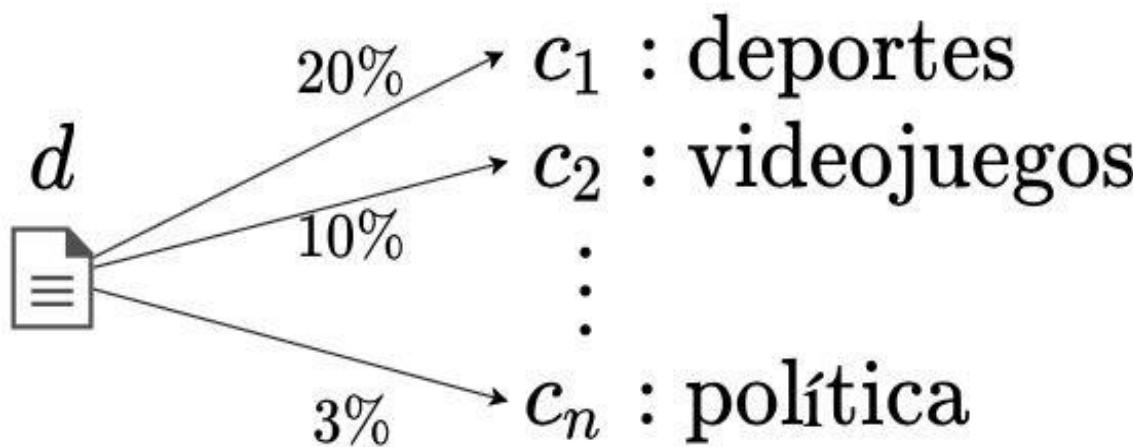




[C23] Naive Bayes en python: construcción del modelo

Clasificación de texto

NB: clasificador probabilístico



$$\hat{c} = \arg \max_{(c)} \log P(c) + \sum_{i=1}^n \log P(f_i | c)$$

“

**Vamos a un notebook
en Google Colab ...**



”



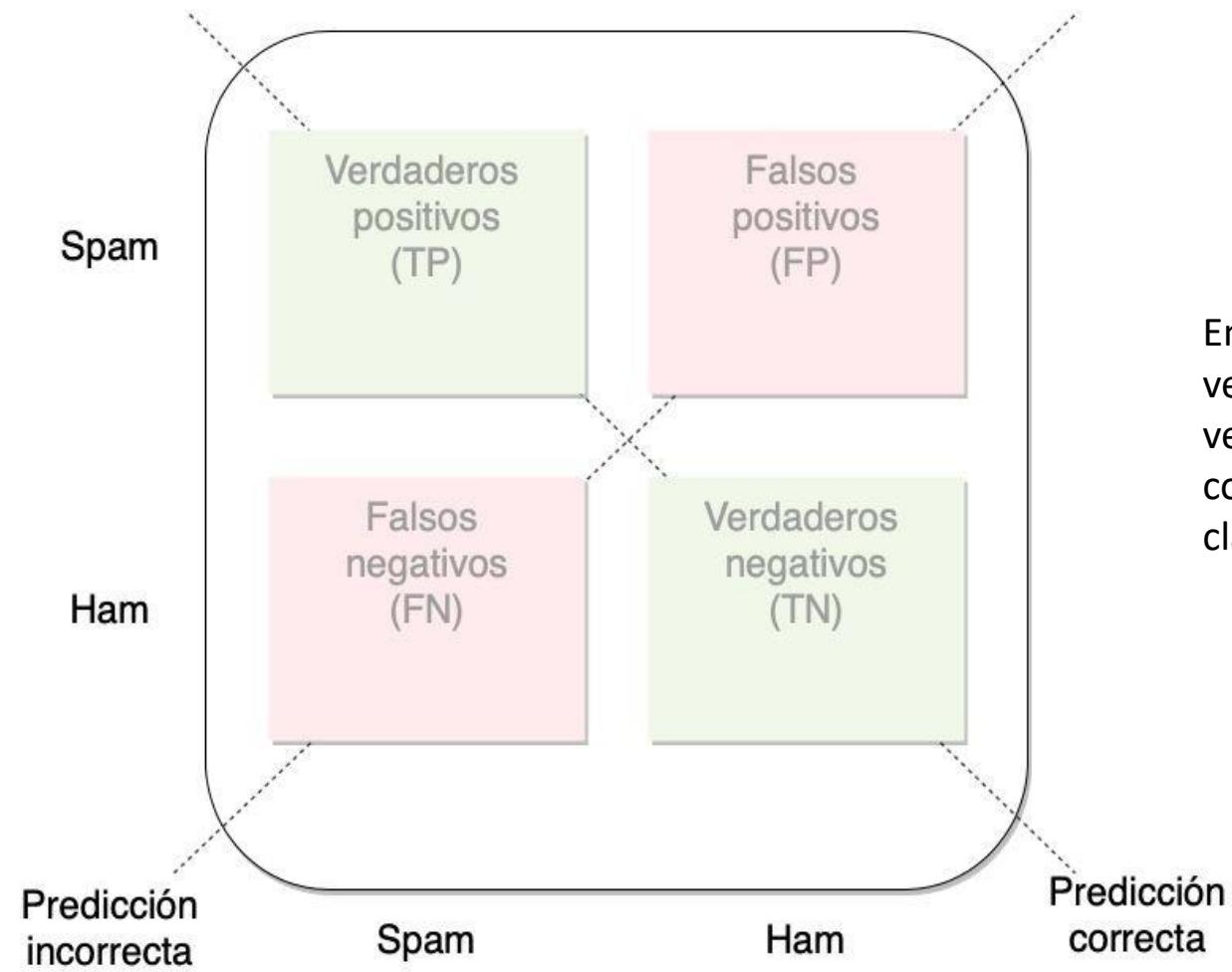
[C25] Métricas para algoritmos de clasificación

Clasificación de texto

Accuracy

Es simplemente la proporción entre el numero de predicciones que fueron correctas y el numero total de predicciones que fueron realizadas, el accuracy máximo es 1, cuando ambos números son iguales.

$$\text{Accuracy}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^n \delta_{y_i, \hat{y}_i}$$



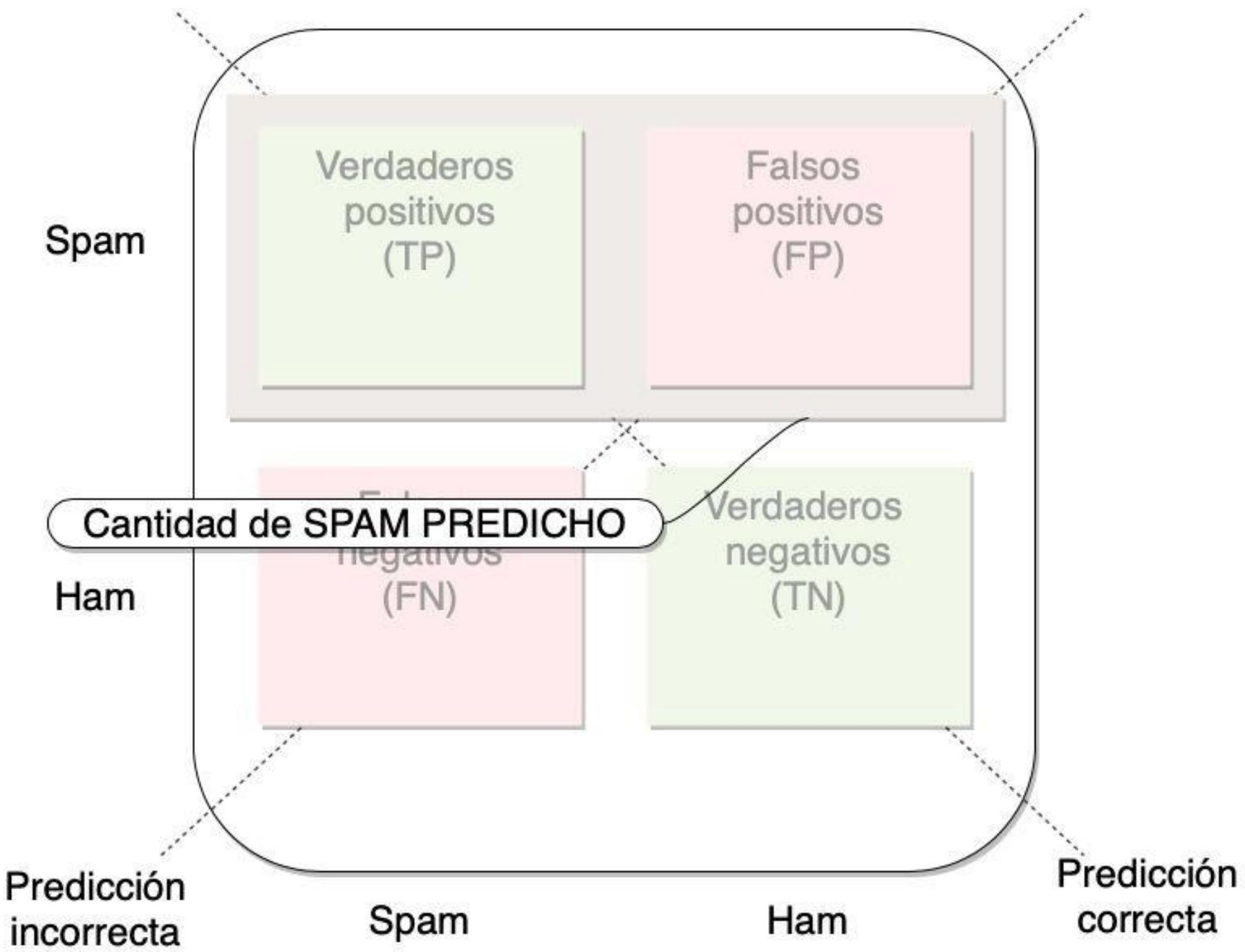
En verde tenemos la suma de los verdaderos positivos y los verdaderos negativos, es decir correos que eran spam o ham clasificados correctamente.

Precision

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

¿Cuántos de los correos clasificados como spam, realmente lo son?

Nos responde básicamente lo siguiente, cuando estoy clasificando correos como spam mi modelo arroja un cierto numero de correos diciendo esto es spam, pero en la realidad solo una fracción de predicciones son verdaderas, y la métrica precisión nos dice eso.

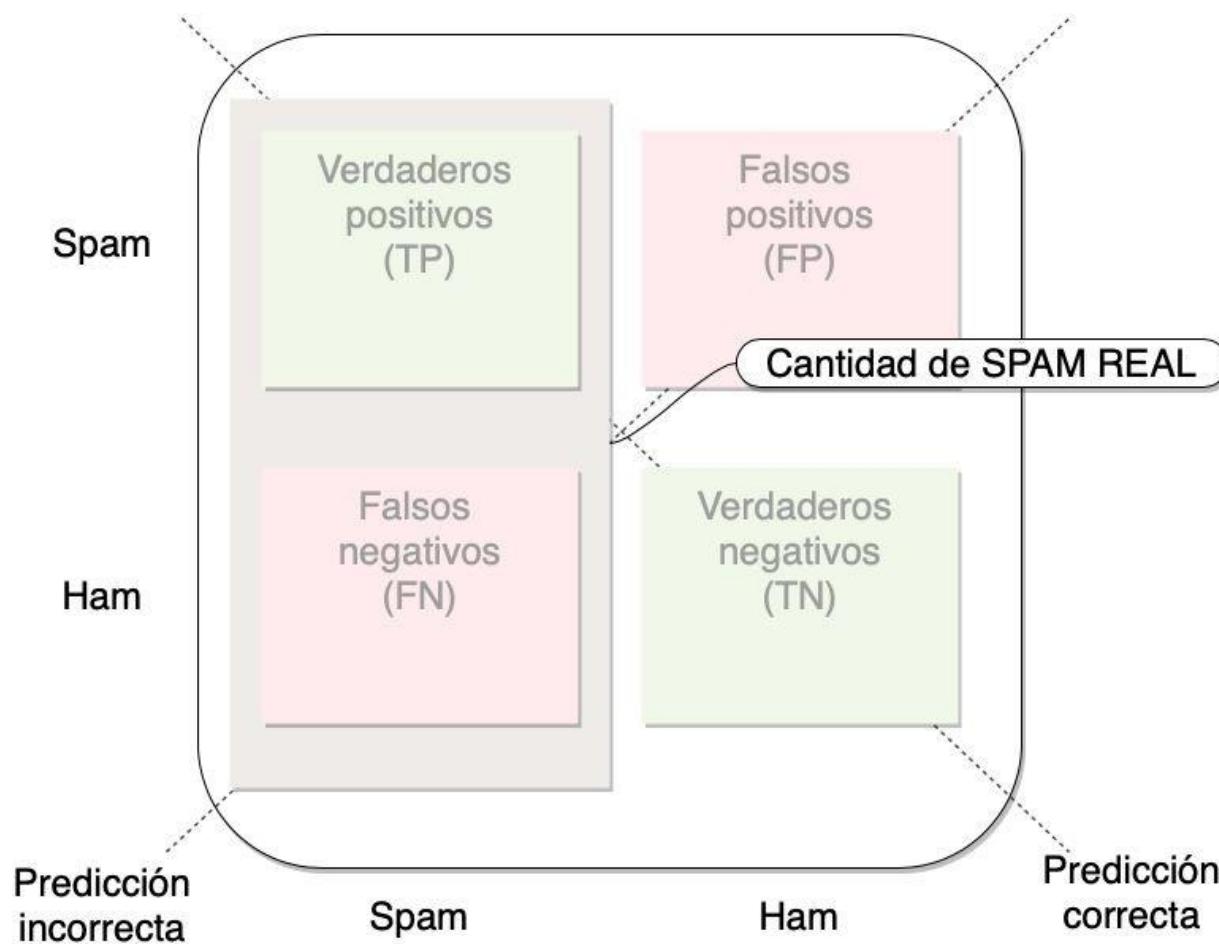


Recall

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

¿Cuántos correos SPAM logramos identificar?

Básicamente dice, en la vida real este conjunto de datos es spam, pero de esa realidad usted solo logró identificar una fracción.



En la tabla lo vemos como lo que era spam, y la fracción que yo logre capturar como spam.

“

**Vamos a un notebook
en Google Colab ...**



”



[C26] Reto Final: Construye un modelo de sentimiento

Clasificación de texto

```
from nltk.corpus import movie_reviews as  
mr  
  
docs =[  
    (list(mr.words(fileid)), category)  
    for category in mr.categories()  
    for fileid in mr.fileids(category) ]  
  
random.shuffle(docs)
```

```
from nltk.corpus import movie_reviews as  
mr  
  
all_words = nltk.FreqDist(  
    w.lower() for w in mr.words()  
)  
  
word_features = list(all_words)[:2000]  
  
...
```

Clasificación de sentimiento

- Dataset:
<http://www.cs.jhu.edu/~mdredze/datasets/sentiment/>
- Naive Bayes, Árboles de decisión, Máxima entropía, etc.





Imágenes de uso libre

unsplash.com

pexels.com

[Pixabay.com](https://pixabay.com)

Íconos de uso libre

thenounproject.com

flaticon.com

Gifs

giphy.com

Íconos de Redes sociales

