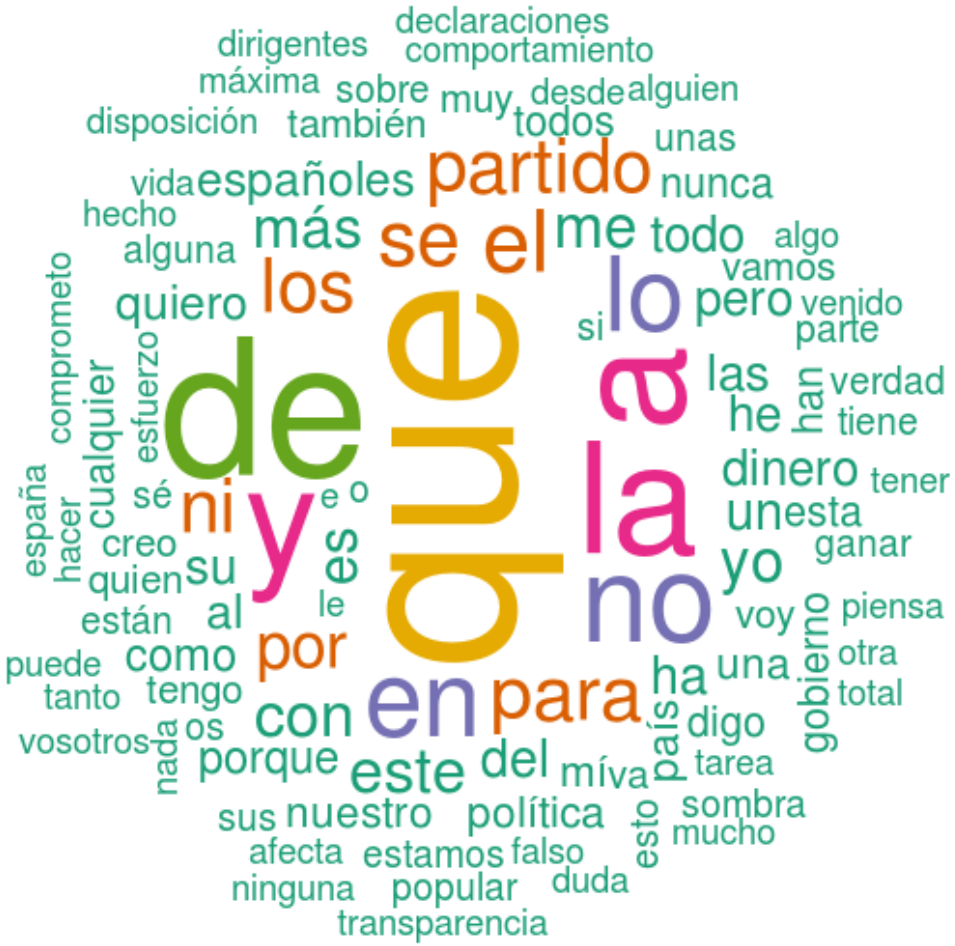
Un lematizador desambiguado con R

Carlos J. Gil Bellosta
cgb@datanalytics.com

Mayo 2013

Contenidos

Una nube de palabras: un primer ensayo



Quitamos las palabras *irrelevantes* (*stopwords*)

El siguiente problema: variedad morfológica

- Queremos que las variantes morfológicas *voten* agregadamente
- P.e., que la nube agrupe "español", "española", "españoles" y "españolas"
- Necesitamos rescatar la **raíz** que transmite significado, el **lema**, de cada término

Lematización

Es el proceso que transforma

- "comiéramos" en "comer"
- "zanjas" en "zanja", etc.

eliminando las marcas morfológicas.

Lematización

Los algoritmos de lematización pueden categorizarse según se basen en

- reglas,
- diccionarios, o
- probabilidades

El que presentaré más adelante es una mezcla de los dos últimos.

Lematización basada en reglas

- El más famoso es *snowball* (de M. Porter)
- Pensado originalmente para el inglés
- Extendido a otros idiomas (incluido el español)

Una regla de *snowball* para el español

Busca el más largo de entre los sufijos

me se sela selo selas selos la le lo las les los nos

y bórralo si sigue a

- *iéndo ándo ár ér ír*
- *ando iendo ar er ir*
- *yendo (si va detrás de u)*

Resultado del algoritmo de Porter

El código, por referencia

```
library(tm)
library(Snowball)

rajoy <- readLines("discurso_rajoy.txt")
rajoy <- paste(rajoy, collapse = " ")
rajoy <- Corpus(
           DataframeSource(data.frame(docs = rajoy)),
           readerControl = list(language = "es") )
rajoy <- tm_map(rajoy, removePunctuation)
rajoy <- tm_map(rajoy, tolower)
rajoy <- tm_map(rajoy, stemDocument)   # snowball!
```

El código, por referencia (cont.)

```
tdm <- TermDocumentMatrix(rajoy,
            control = list(removePunctuation = T,
            stopwords = T,
            wordLengths = c(2, Inf)))

m <- as.matrix(tdm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(word = names(v),freq=v)

wordcloud( d$word, d$freq, min.freq = 3,
           scale=c(4,.5),
           colors=brewer.pal(6,"Dark2"),
           random.order=FALSE)
```

Snowball puede ser útil

Tener un diccionario con todas las parejas (término, lema)

- Basado en diccionarios de parejas `(término, lema)`.
- Lematizar consiste en buscar el término y devolver el lema correspondiente.

La ambigüedad, el gran problema:

- `(casas, casar)` (contraer matrimonio)
- `(casas, casa)` (edificio)

Y como este, muchos más casos (más de los que uno piensa).

Uno de los lematizadores de Molino de Ideas

Petición:

```
https://store.apicultur.com/api/lematiza-molino/1.0.0/casas
```

Respuesta:

```
{ "palabra": "casas",
  "lemas": [
    { "lema": "casar", "categoria": "5" },
    { "lema": "casa",  "categoria": "4" }
   ]
}
```

Resultado aplicando un lematizador basado en diccionarios

Un modelo probabilístico para palabras y etiquetas

- Vamos a ver cómo asociar a cada término w una etiqueta t
- Las etiquetas pueden ser:
  - La raíz (potencial) de dicho término
  - Su categoría morfológica (verbo, sustantivo,...)
  - Si son palabras comunes o nombres de personas, calles, empresas,...
- Dada una frase $w_1, \ldots, w_n$, buscamos la secuencia de etiquetas $t_1, \ldots, t_n$ más probable
- Es decir, queremos maximizar $P(t_1, \ldots, t_n | w_1, \ldots, w_n)$

Aplicamos el cálculo de probabilidades (pura álgebra)

$$P(t_1, \ldots, t_n | w_1, \ldots, w_n) = P_w(t_1, \ldots, t_n) =$$
$$P_w(t_n | t_1, \ldots, t_{n-1}) P_w(t_1, \ldots, t_{n-1}) =$$
$$\prod_i P_w(t_i | t_{i-1}, \ldots, t_1) =$$
$$\prod_i P(t_i | t_{i-1}, \ldots, t_1, w_1, \ldots, w_n)$$

¡*Sólo* necesitamos calcular $P(t_i | t_{i-1}, \ldots, t_1, w_1, \ldots, w_n)$!

Algunas simplificaciones

- En general, $P(t_i|t_{i-1},\ldots,t_1,w_1,\ldots,w_n)$ es incognoscible
- Buscamos simplificaciones que
  - No sean excesivas
  - Que tengan sentido lingüístico
  - Que den lugar a expresiones calculables

Típicamente, se supone que t_i es independiente de $t_{i-3}, t_{i-4}, \ldots$, con lo que

$$P(t_i|t_{i-1},\ldots,t_1,w_1,\ldots,w_n) \approx P(t_i|t_{i-1},t_{i-2},w_1,\ldots,w_n)$$

Modelos lineales generalizados (GLM)

- Son modelos en los que $P(t_i|t_{i-1}, t_{i-2}, w_1, \ldots, w_n)$ se aproxima por $\exp(\sum \lambda_i f_i(t, t_1, t_2, w_1, \ldots, w_n))$
- Las funciones f_i responden preguntas del tipo: ¿la etiqueta anterior es un artículo femenino y la palabra termina con "a"?
- Los pesos λ_i se ajustan maximizando la verosimilitud de un conjunto de entrenamiento.

Nótese cómo maximizar el producto

$$\prod_i P(t_i|t_{i-1}, \ldots, t_1, w_1, \ldots, w_n)$$

equivale a maximizar

$$\sum_{i,j} \lambda_i f_i(t_j, t_{j1}, t_{j2}, w_1, \ldots, w_n)$$

Modelos ocultos de Markov (HMM)

Son modelos en los que se realiza la siguiente aproximación:

$$\begin{aligned} P(t_i|t_{i-1}, t_{i-2}, w_1, \ldots, w_n) &\approx \\ P(t_i|t_{i-1}, t_{i-2}, w_i) &\approx \\ P(t_i|t_{i-1}, t_{i-2})P(t_i|w_i) & \end{aligned}$$

- $P(t_i|t_{i-1}, t_{i-2})$: transiciones de un modelo de Markov (de segundo orden); p.e., $P(\text{adj.}|\text{sust.}, \text{art.})$
- $P(t_i|w_i)$: indica lo probable que es la etiqueta t_i cuando se observa la palabra w_i; p.e., $P(\text{verbo}|\text{ascensor})$

Algoritmo de Viterbi para HMM

El modelo anterior se califica de *oculto* porque

- los estados de la cadena de Markov no son observables
- lo observable son etiquetas *emitidas* probabilísticamente por dichos estados

El **algoritmo de Viterbi** permite calcular la secuencia de estados de la cadena que maximiza la probabilidad de las etiquetas observadas.

Los elementos para implementar el algoritmo de Viterbi

Si hay n etiquetas y m términos, se necesitan:

- Un array $n \times n \times n$ de transición entre etiquetas.
- Un diccionario $m \times n$ que indique la probabilidad de que un término tenga una etiqueta dada
- Una lista de matrices $n \times n$ que almacenen las combinaciones más probables en los pasos $i - 1$ e $i - 2$ de la cadena.

La lista de matrices se construye paso a paso recorriendo la cadena de inicio a fin.
Luego hay que recorrerla de fin a inicio para recuperar las etiquetas que han dado lugar a la traza de máxima probabilidad.

Modelo híbrido diccionarios / HMM

- Es el que he implementado para Molino de Ideas

Algoritmo de lematizacíon *desambiguada*

- Usar **diccionarios** para sugerir raíces de términos
- Usar **HMM** para etiquetar morfológicamente
- Usar la etiqueta sugerida por el HMM para **desambiguar** las opciones del diccionario

- **Ventajas:** decenas de etiquetas morfológicas pero miles de posibles lemas
- **Limitaciones:** fue: ¿ir o ser? ¡ambos son verbos!

Motor del algoritmo (en R)

```
pila <- function(i, pk){
  tmp   <- log.wuv +
             outer(pk, p.ter.etq[[i]],"+")
  probs <- apply(tmp, 2:3, max)

  res <- if(i == length(frase))  # fin cadena
           { devolver que maximizan probs }
         else                          # iteramos
           pila(i + 1, probs)

    return(c(arg.max(probs, res), res))
}

pila(1, prob.ini)
```

Lematización en español

- El español no es un idioma como el inglés
- En español hacen falta herramientas específicas
- Creo que hay que pasar por lematizadores basados en diccionarios
- Los cálculos son complejos y las necesidades de memoria, elevadas
- Una solución basada en APIs, ¿adecuada?