Aprendizaje de Máquina - ITAM

Octubre, 2016

MSc. Liliana Millán, liliana.millan@gmail.com

Agenda

- Information Retrieval
- ¿Por qué es importante?
- Métodos/algoritmos para medir relevancia
- Proceso estándar de minado de texto
- Ejemplo de TF/IDF
- Caso de Negocio
- Referencias

Minería de Texto - Information Retrieval (IR)

Information Retrieval es una rama de las Ciencias Computacionales que se define como:

> "Ciencia dedicada a encontrar material digital —generalmente documentos— de una naturaleza no estructurada —generalmente textos— de entre una gran cantidad de colecciones —generalmente almacenada en computadoras— que satisfacen una necesidad de información." *An Introduction to Information Retrieval*

En un sistema de IR hay 3 elementos:

1. Una consulta que puede estar formada por más de un término
2. Una colección de documentos en donde buscar la consulta
3. Ordenar la colección de acuerdo a su relevancia con respecto a la consutla

El buscador de Google es un sistema IR, el primer elemento consiste en las palabras que ponemos en el buscador, el segundo elemento corresponde a todas las páginas web, y el tercer elemento corresponde a la respuesta de documentos que regresa Google ordenada descendentemente por relevancia con base a los términos de consulta.

¿Por qué es importante? ¿En qué se ocupa?

¿Cuánto crees que vale Google? --> **$82,500 millones de dólares** *Mayo 2016, Forbes*

El algoritmo de Pagerank (http://infolab.stanford.edu/pub/papers/google.pdf) de Google (1998) fue una de las aplicaciones de IR que cambió el mundo digital y que atrajo a esta disciplina mucho interés.

- **SEO (Search Engine Optimization)**. Etiquetar el contenido de manera correcta para que sea encontrado por un buscador de páginas (Google) y se muestre como un documento relevante —mientras más arriba más relevante... y mayor probabilidad de que te den clic—
- **Clasificación de texto**. Asignar un texto a una categoría generalmente tópicos, estos tópicos pueden ser jerárquicos.
- **Organización de información**. Generar Taxonomías/Ontologías/Grafos de contenidos
- **Recomendación de contenidos**. Una vez que el contenido ha sigo clasificado nos permite hacer recomendaciones de contenidos similares.

Aproximación al problema

Los diferentes modelos de IR cambian de acuerdo a cómo miden la **relevancia** y generalmente se dividen en modelos algebráicos, probabilísticos y de machine learning.

- Algebráicos: Basados en el *Vector Space Model* TF-IDF, distancia coseno, similitud de Jaccard ...
- Probabilísticos: Basados es en el *Binary Independence Model* (BIM): Okapi BM25, LSI, LDA ...
- *Machine Learning: Deep Learning uan vez que el problema se convierte a clasificación de texto!

Algebraicos

Vector Space Model

Una forma de representar un documento consiste en *romperlo* en términos —generalmente palabras— que lo conforman, de esta manera cada documento tiene asociado esta representación vectorial $\vec{x} = (x_1, \ldots, x_m)$ de acuerdo a los términos que aparecen en él.

A esta representación se le conoce también como **bolsa de palabras** pues al romper el documento en términos suponemos que cada uno ocurre independientemente, sabemos que eso no es cierto y que el orden de las palabras claramente va asociado a un contexto/significado, sin embargo esa sola suposición nos permite generar modelos suficientemente buenos.

En los modelos algebráicos la relevancia se mide de acuerdo a la **similaridad** de los documentos al query.

Jaccard Similarity Coefficient

Mide similitud entre conjuntos

$$J(A,B) = \frac{(A \cap B)}{A \cup B}$$

Ejemplo: A={'una','casa','cama','votos','elefante'}, B={'casa','elefante','leon'} la similitud de Jaccard entre estos dos conjuntos es:

$$J(A,B) = \frac{('casa','elefante')}{('una','casa','cama','votos','elefante','leon')} = \frac{2}{6} = \frac{1}{3} = 0.33$$

El conjunto A y B tienen 0.33 de similitud, mientras el número sea más cercano a 1 mayor será la similitud.

TF/IDF

Es el algoritmo más popular para la recuperación de información debido a que su explicación es intuitiva y es muy fácil de implementar. La relevancia de un documento se mide con relación a la frecuencia que el término de consulta aparece en cada documento de la colección y el inverso del número de documentos de la colección en los que aparece el término

$$tfidf_t = tf_{t,d} \cdot log\frac{N}{df}$$

Donde:

- t: Término de consulta

- tf: Frecuencia del término t en el documento d
- N: Número de documentos en nuestra colección
- df: Número de documentos en los que aparece el término
- El logaritmo permite limitar el peso de tener colecciones de documentos muy grandes (generalmente se ocupa en base 10)

Cuando se tiene más de un término en la consulta el TF/IDF final de un documento se obtiene sumando el TF/IDF obtenido para cada término en la consulta:

$$tfidf_{c,d} = \sum_{t \in c} tfidf_{t,d}$$

Este modelo ocupa una matriz de términos (renglones) y documentos (columnas) en donde almacenamos la frecuencia de aparición de cada término para cada documento en la colección. Esta matriz toma como base la representación vectorial y generalmente es una matriz *sparse* —con muchos 0s—

Probabilísticos

Ocupan la misma representación vectorial de los documentos que los modelos algebraicos pero la relevancia está medida con modelos probabilísticos que estiman que tan probable es que un documento sea relevante a un query —generalmente se ocupa Bayes—

Binary Independence Retrieval Model (BIM)

En este método el vector de un documento toma valores binarios: 1 si el término está en el documento, 0 en otro caso $d = \vec{x} = (x_1, \ldots, x_n) x_i = 1$ si el término está en el documento d.

En este modelo la relevancia de un documento está medida a través Bayes ocupando la incidencia de los términos $\vec{x}$ en la consulta $\vec{q}$

$P(R|d,q)$

Donde:

- R: Relevancia
- d: Documento
- q: Query

Entonces

$$P(R = 1|\vec{x}, \vec{q}) = \frac{P(\vec{x}|R = 1, \vec{q}) P(R = 1|\vec{q})}{P(\vec{x}|\vec{q})}$$
$$P(R = 0|\vec{x}, \vec{q}) = \frac{P(\vec{x}|R = 0, \vec{q}) P(R = 0|\vec{q})}{P(\vec{x}|\vec{q})}$$

- $P(\vec{x}|R = 1, \vec{q})$ y $P(\vec{x}|R = 0, \vec{q})$: La probabilidad de que si un documento regresado es relevante o irrelavante su representación es $\vec{x}$
- $P(R = 1|\vec{q})$ y $P(R = 0|\vec{q})$: La probabilidad a priori de haber regresado un documento relevante o no dado el query

Debido a que un documento solo puede ser relavante o no $P(R = 1|\vec{x}, \vec{q}) + P(R = 0|\vec{x}, \vec{q}) = 1$

Debido a que nos interesa regresar toda la colección ordenada por relevancia: orden descendiente de $P(R=1|\vec{x},\vec{q})$ en lugar de calcular directamente esta probabilidad se ocupan otros valores que regresan el mismo orden.

...

Para ver los detalles de como se paso de una cosa a otra revisar

Introduction to IR, Deriving a ranking function for query terms (http://nlp.stanford.edu/IR-book/html/htmledition/deriving-a-ranking-function-for-query-terms-1.html)

Introduction to IR, Probability estimates in theory (http://nlp.stanford.edu/IR-book/html/htmledition/probability-estimates-in-theory-1.html)

$$log\frac{s+\frac{1}{2}/(S-s+\frac{1}{2})}{df_t-s+\frac{1}{2}/(N-df_t-S+s+\frac{1}{2})}$$

- Se agrega 1/2 para evitar 0's (smoothing)
- En los casos prácticos se asume que los documentos relevantes en una collección son una proporción muy pequeña de la misma y por lo tanto la ecuación anterior toma la forma

$$log\frac{N}{df}$$

Introduction to IR, Probability estimates in practice (http://nlp.stanford.edu/IR-book/html/htmledition/probability-estimates-in-practice-1.html) para la explicación

Es el idf de TF/IDF!

BM25 (Okapi Best Match 25)

Este modelo toma en consideración la longitud de los documentos, la longitud del query y la frecuencia del término en el query en 3 parámetros que permiten 'controlar' la influencia que cada uno de estos tendrá en la métrica de relevancia.

$$\sum_{i \in Q} log \frac{(r_i + 0.5)(N - n_i - R + r_i + 0.5)}{(n_i - r_i + 0.5)(R - r_i + 0.5)} \cdot \frac{(k_1 + 1)f_i}{(K + f_i)} \cdot \frac{(k_2 + 1)qf_i}{k_2 + qf_i}$$

$$K = k_1 \left((1 - b) + b \cdot \frac{dl}{avgdl} \right)$$

- $avgdl$ Longitud promedio de los documentos de la colección, medido en el # de palabras
- dl Longitud de cada documento, medido en el # de palabras
- N # de documentos que hay en la colección
- R # de documentos relevantes en el set, si no contamos con esta información se pone en 0
- r_i # de documentos relevantes que contienen el término i, si no contamos con esta información se pone en 0
- n_i # de documentos que contienen el término i
- f_i La frecuencia del término i en el documento
- qf_i La frecuencia del término i en el query
- k_1 Parámetro que determina cómo la frecuencia del término i afectará a su peso mientras la frecuencia incrementa en cada documento. Si es 0 entonces la frecuencia del término es ignorada. Provoca que f_i no sea lineal y que después de 3 o 4 ocurrencias del término tener más ocurrencias tengan poco impacto en la calificación, $\in [1.0, 2.0]$. De acuerdo a TREC (http://trec.nist.gov/) un valor típico es de 1.2
- k_2 Parámetro que controla el peso del término i en la consulta $\in [0 - 1000]$. De acuerdo a TREC (http://trec.nist.gov/) su valor típico bajo, provoca que la calificación de BM25 sea menos sensible a k2 que a k1 ya que las frecuencias de los términos en la cosulta son menores y menos variables que las frecuencias de los términos en los documentos de la colección.
- K Normaliza el componente de frecuencia del término de acuerdo a la longitud del documento.
- b Regulariza el impacto de la normalización realizada con la longitud del documento, $\in [0, 1]$ Si b es 0 no se toma en cuenta la normalización, de acuerdo a TREC (http://trec.nist.gov/) su valor típico es de 0.75

Latent Semantic Indexing LSI (http://nlp.stanford.edu/IR-book/html/htmledition/latent-semantic-indexing-1.html)

Este método utiliza la matriz de términos y documentos para hacer un producto punto entre todos los términos y documentos para ocupar un Singular Value Decomposition (SVD) para reducir dimensionalidad... la idea que documentos que tienen palabras similiares son del mismo contexto. Fue de los primero métodos que intentan descubrir contexto —semántica— en los documentos basado en una bolsa de palabras.

LDA Latent Dirichlet Allocation LDA Andrew Ng (https://www.cs.princeton.edu/~blei/papers/BleiNgJordan2003.pdf), LDA para mortales (http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/)

Este método a diferencia de los antes mencionados asigna una probabilidad de pertenencia a cada tópico por lo que una mismo documento está asociado con diferentes probabilidades a todos los tópicos disponibles.

- Cada término de un documento es asociado a todos los tópicos con cierta probabilidad (obtenida a través de iterar sobre un EM para norm ales multivariadas —Distribución Dirichlet—)
- Las variables latentes son los tópicos y son definidos con anticipación por el experto
- Permite obtener de manera automática el tópico al que pertenece un documento

Proceso estándar para minado de texto

1. Tokenización: Cada token normalmente es una palabra, aunque también pueden ser n-gramas donde n puede ser palabras (bi-gramas, tri-gramas, etc.) o bien sílabas o letras
    - El enunciado: 'Un enunciado ejemplo de tokenización', estaría formado por 5 tokens unigramas: un, enunciado, ejemplo, de, tokenización
    - Por 4 tokens bigramas: Un enunciado, enunciado ejemplo, ejemplo de, de tokenización
2. Limpieza de tokens:
    - Eliminación de signos de puntuación
    - Eliminación de números (depende del caso de negocio)
    - Eliminación de espacios al inicio/final de palabras
    - Pasar a minúsculas
3. *Stopwords*. Dependiendo de nuestro problema de negocio generamos una *lista negra* de palabras que no queremos tomar en cuenta en nuestra métrica de relevancia. Generalmente en esta lista ponemos artículos, preposiciones, conjunciones y palabras específicas al problema. Existen varias listas ya creadas para diferentes idiomas, generalmente a estas listas se les pueden agregar palabras propias o eliminar algunas —en R, paquete tm (https://cran.r-project.org/web/packages/tm/tm.pdf)—

    - Inglés genérico Snowball (http://snowball.tartarus.org/algorithms/english/stop.txt)
    - Español genérico Snowball (http://snowballstem.org/algorithms/spanish/stop.txt)
    - Francés Snowball (http://snowballstem.org/algorithms/french/stop.txt)
    - Portugués Snowball (http://snowballstem.org/algorithms/portuguese/stop.txt)
    - Italiano Snowball (http://snowballstem.org/algorithms/italian/stop.txt)
    - Ruso Snowball (http://snowballstem.org/algorithms/russian/stop.txt)
    - Sueco Snowball (http://snowballstem.org/algorithms/swedish/stop.txt)
    - Finlandés Snowball (http://snowballstem.org/algorithms/finnish/stop.txt)
    - Danés Snowball (http://snowballstem.org/algorithms/danish/stop.txt)
    - Noruego Snowball (http://snowballstem.org/algorithms/norwegian/stop.txt)
4. *Stemming*. Proceso **heurístico** en el que se eliminan plurales y cambiando algunas derivaciones esperando modificar palabras que son lo mismo que pueden estar presentes de diferentes manera. En inglés generalmente se ocupa el algoritmo de Porter (https://tartarus.org/martin/PorterStemmer/)

    - Claramente el algoritmo de *stemming* depende del lenguaje en el que se encuentra el problema de negocio. Existen varios algoritmos dependiendo del lenguaje stemming español, SNOWBALL (http://snowball.tartarus.org/algorithms/spanish/stemmer.html)
        - cheque --> chequ
        - chequeo --> cheque
        - cheques --> chequ
    - Español y otros idiomas: librería CLIPS (http://www.clips.ua.ac.be/pattern) para python 2
5. *Lemmatization*. Proceso formal en el que se obtiene la raíz de una palabra —lema— realizando análisis morfológico.

    - Depende del lenguaje se ocupan algoritmos diferentes
    - Estos algoritmos están hechos de la mano de lingüístas

Los sistemas de IR son dependientes del idioma por lo que se requiere desarrollar uno específico por lenguaje (inclusive por domino): Inglés americano, Inglés británico, Español mexicano, Español Venezolano, etc.

Ejemplo para TF-IDF

Tenemos una colección C de 5 documentos d_i obtenidos de titulares de New York Times español (http://www.nytimes.com/es/):

d_1 : Dilma Rousseff se defiende ante sus rivales en el senado: 'No me silenciarán'

d_2 : El juicio contra Dilma Rousseff, el próximo evento imperdible en Brasil

d_3 : El Senado de Brasil avanza hacia el juicio de Dilma Rousseff

d_4 : El último intento de Dilma Rousseff por volver a la presidencia

d_5 : La destitución de Rousseff revela actitudes discriminatorias hacia las mujeres en Brasil

In [78]:
```
import sys
import IPython
9
print "python version: " + format(sys.version_info[0:30]) +\
"\nIPython version: " + format(IPython.version_info)
```

```
python version: (2, 7, 9, 'final', 0)
IPython version: (2, 4, 1, '')
```

In [15]:
```
d1 = u"Dilma Rousseff se defiende ante sus rivales en el senado: 'No me silenc
iarán'".encode('utf-8')
d2 = u'El juicio contra Dilma Rousseff, el próximo evento imperdible en Brasi
l'.encode('utf-8')
d3 = u'El Senado de Brasil avanza hacia el juicio de Dilma Rousseff'.encode('u
tf-8')
d4 = u'El último intento de Dilma Rousseff por volver a la presidencia'.encode
tf-8')
d5 = u'La destitución de Rousseff revela actitudes discriminatorias hacia las
 mujeres en Brasil'.encode('utf-8')

#########quitar acentos!, después se vuelve más complicado!
def quitar_acentos(doc):
    a = doc.replace("á", "a")
    e = a.replace("é", "e")
    i = e.replace("í", "i")
    o = i.replace("ó", "o")
    u =o.replace("ú", "u")

    return u

docs_collection = [d1,d2,d3,d4,d5]

docs_collection = [quitar_acentos(doc) for doc in docs_collection]
docs_collection
```

Out[15]:
```
["Dilma Rousseff se defiende ante sus rivales en el senado: 'No me silenciara
n'",
 'El juicio contra Dilma Rousseff, el proximo evento imperdible en Brasil',
 'El Senado de Brasil avanza hacia el juicio de Dilma Rousseff',
 'El ultimo intento de Dilma Rousseff por volver a la presidencia',
 'La destitucion de Rousseff revela actitudes discriminatorias hacia las muje
res en Brasil']
```

In [40]:
```
#tokenizacion
def tokenizacion(doc):
   return doc.split(" ")

terms = [tokenizacion(doc) for doc in docs_collection]
print terms
```

```
[['Dilma', 'Rousseff', 'se', 'defiende', 'ante', 'sus', 'rivales', 'en', 'e
l', 'senado:', "'No", 'me', "silenciaran'"], ['El', 'juicio', 'contra', 'Dilm
a', 'Rousseff,', 'el', 'proximo', 'evento', 'imperdible', 'en', 'Brasil'],
 ['El', 'Senado', 'de', 'Brasil', 'avanza', 'hacia', 'el', 'juicio', 'de', 'D
ilma', 'Rousseff'], ['El', 'ultimo', 'intento', 'de', 'Dilma', 'Rousseff', 'p
or', 'volver', 'a', 'la', 'presidencia'], ['La', 'destitucion', 'de', 'Rousse
ff', 'revela', 'actitudes', 'discriminatorias', 'hacia', 'las', 'mujeres', 'e
n', 'Brasil']]
```

In [41]:
```
#queremos una lista flatten!
def lista_flatten(lista):
    return [word for sentence in lista for word in sentence]

flattened_terms = lista_flatten(terms)
print flattened_terms, len(flattened_terms)
```

```
['Dilma', 'Rousseff', 'se', 'defiende', 'ante', 'sus', 'rivales', 'en', 'el',
 'senado:', "'No", 'me', "silenciaran'", 'El', 'juicio', 'contra', 'Dilma',
 'Rousseff,', 'el', 'proximo', 'evento', 'imperdible', 'en', 'Brasil', 'El',
 'Senado', 'de', 'Brasil', 'avanza', 'hacia', 'el', 'juicio', 'de', 'Dilma',
 'Rousseff', 'El', 'ultimo', 'intento', 'de', 'Dilma', 'Rousseff', 'por', 'vo
lver', 'a', 'la', 'presidencia', 'La', 'destitucion', 'de', 'Rousseff', 'reve
la', 'actitudes', 'discriminatorias', 'hacia', 'las', 'mujeres', 'en', 'Brasi
l'] 58
```

In [43]:
```
#queremos solo palabras unicas
def palabras_unicas(terms):
    return list(set(terms))

unique_terms = palabras_unicas(flattened_terms)
print unique_terms, len(unique_terms)
```

```
['el', 'en', 'La', 'por', 'a', 'hacia', 'ante', 'actitudes', 'intento', 'juic
io', 'revela', "silenciaran'", 'defiende', 'proximo', 'contra', 'Senado', 'su
s', 'presidencia', 'Rousseff,', 'senado:', 'Brasil', 'mujeres', 'evento', 'Di
lma', 'de', 'volver', 'avanza', 'rivales', 'imperdible', 'ultimo', "'No", 'la
s', 'me', 'El', 'la', 'destitucion', 'discriminatorias', 'Rousseff', 'se'] 39
```

In [20]:
```
#quitar puntuación
import string

string.punctuation
```

Out[20]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [27]:
```
def eliminar_puntuacion(term):
    replace_punctuation = string.maketrans(string.punctuation, '
'*len(string.punctuation))
    return term.translate(replace_punctuation)


terms_wo_punctuation = [eliminar_puntuacion(term) for term in unique_terms]
print terms_wo_punctuation
```

```
['el', 'en', 'La', 'por', 'a', 'hacia', 'ante', 'actitudes', 'intento', 'juic
io', 'revela', 'silenciaran ', 'defiende', 'proximo', 'contra', 'Senado', 'su
s', 'presidencia', 'Rousseff ', 'senado ', 'Brasil', 'mujeres', 'evento', 'Di
lma', 'de', 'volver', 'avanza', 'rivales', 'imperdible', 'ultimo', ' No', 'la
s', 'me', 'El', 'la', 'destitucion', 'discriminatorias', 'Rousseff', 'se']
```

In [29]:
```
#quitar números
import re

def eliminar_numeros(term):
    return re.sub(r'\d+','',term)

terms_wo_numbers = [eliminar_numeros(term) for term in terms_wo_punctuation]
print terms_wo_numbers
```

```
['el', 'en', 'La', 'por', 'a', 'hacia', 'ante', 'actitudes', 'intento', 'juic
io', 'revela', 'silenciaran ', 'defiende', 'proximo', 'contra', 'Senado', 'su
s', 'presidencia', 'Rousseff ', 'senado ', 'Brasil', 'mujeres', 'evento', 'Di
lma', 'de', 'volver', 'avanza', 'rivales', 'imperdible', 'ultimo', ' No', 'la
s', 'me', 'El', 'la', 'destitucion', 'discriminatorias', 'Rousseff', 'se']
```

In [31]:
```
#quitar espacios
def eliminar_espacios(term):
    return term.strip()

terms_wo_spaces = [eliminar_espacios(term) for term in terms_wo_numbers]
print terms_wo_spaces
```

```
['el', 'en', 'La', 'por', 'a', 'hacia', 'ante', 'actitudes', 'intento', 'juic
io', 'revela', 'silenciaran', 'defiende', 'proximo', 'contra', 'Senado', 'su
s', 'presidencia', 'Rousseff', 'senado', 'Brasil', 'mujeres', 'evento', 'Dilm
a', 'de', 'volver', 'avanza', 'rivales', 'imperdible', 'ultimo', 'No', 'las',
 'me', 'El', 'la', 'destitucion', 'discriminatorias', 'Rousseff', 'se']
```

In [32]:
```
#todo a minúsculas
def a_minusculas(term):
    return term.lower()

terms_lower = [a_minusculas(term) for term in terms_wo_spaces]
print terms_lower
```

```
['el', 'en', 'la', 'por', 'a', 'hacia', 'ante', 'actitudes', 'intento', 'juic
io', 'revela', 'silenciaran', 'defiende', 'proximo', 'contra', 'senado', 'su
s', 'presidencia', 'rousseff', 'senado', 'brasil', 'mujeres', 'evento', 'dilm
a', 'de', 'volver', 'avanza', 'rivales', 'imperdible', 'ultimo', 'no', 'las',
 'me', 'el', 'la', 'destitucion', 'discriminatorias', 'rousseff', 'se']
```

In [34]:
```
#stop words
my_stop_words = ['el','la','en','de','a','por','se','las','no','me','sus']

#eliminar stopwords
def eliminar_stopwords(term, stopwords):
    return list(set(term) - set(stopwords))


terms_wo_stopwords = eliminar_stopwords(terms_lower, my_stop_words)
print terms_wo_stopwords, len(terms_wo_stopwords)

#stemming
#lemmatization
```

```
['hacia', 'ante', 'actitudes', 'intento', 'juicio', 'imperdible', 'proximo',
 'rousseff', 'contra', 'presidencia', 'defiende', 'mujeres', 'evento', 'silen
ciaran', 'senado', 'volver', 'avanza', 'rivales', 'revela', 'ultimo', 'brasi
l', 'dilma', 'destitucion', 'discriminatorias'] 24
```

In [208]:
```
docs_wo_punctuation = [eliminar_puntuacion(doc) for doc in docs_collection]
docs_wo_space = [eliminar_espacios(doc) for doc in docs_wo_punctuation]
docs_wo_lower = [a_minusculas(doc) for doc in docs_wo_space]


#matriz terminos-documentos
def frequency_count(term, words_in_docs):
    return [doc.count(term) for doc in words_in_docs]


#contar la frecuencia de cada termino en cada documento
tf = [frequency_count(term, docs_wo_lower) for term in terms_wo_stopwords]
zip(terms_wo_stopwords, tf)
```

Out[208]:
```
[('hacia', [0, 0, 1, 0, 1]),
 ('ante', [1, 0, 0, 0, 0]),
 ('actitudes', [0, 0, 0, 0, 1]),
 ('intento', [0, 0, 0, 1, 0]),
 ('juicio', [0, 1, 1, 0, 0]),
 ('imperdible', [0, 1, 0, 0, 0]),
 ('proximo', [0, 1, 0, 0, 0]),
 ('rousseff', [1, 1, 1, 1, 1]),
 ('contra', [0, 1, 0, 0, 0]),
 ('presidencia', [0, 0, 0, 1, 0]),
 ('defiende', [1, 0, 0, 0, 0]),
 ('mujeres', [0, 0, 0, 0, 1]),
 ('evento', [0, 1, 0, 0, 0]),
 ('silenciaran', [1, 0, 0, 0, 0]),
 ('senado', [1, 0, 1, 0, 0]),
 ('volver', [0, 0, 0, 1, 0]),
 ('avanza', [0, 0, 1, 0, 0]),
 ('rivales', [1, 0, 0, 0, 0]),
 ('revela', [0, 0, 0, 0, 1]),
 ('ultimo', [0, 0, 0, 1, 0]),
 ('brasil', [0, 1, 1, 0, 1]),
 ('dilma', [1, 1, 1, 1, 0]),
 ('destitucion', [0, 0, 0, 0, 1]),
 ('discriminatorias', [0, 0, 0, 0, 1])]
```

In [206]:
```
def inverse_document_frequency(term, collection):
    bidf = [term in doc for doc in collection]
    return sum(bidf)

df = [inverse_document_frequency(term, docs_wo_lower) for term in terms_wo_sto
pwords]
df
```

Out[206]:
```
[2, 1, 1, 1, 2, 1, 1, 5, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 3, 4, 1, 1]
```

In [207]:
```
N = len(docs_wo_lower)*1.0

def idf(term, N):
    return math.log10(N/term)


idf_values = [idf(term, N) for term in df]
zip(terms_wo_stopwords, idf_values)
```

Out[207]:
```
[('hacia', 0.3979400086720376),
 ('ante', 0.6989700043360189),
 ('actitudes', 0.6989700043360189),
 ('intento', 0.6989700043360189),
 ('juicio', 0.3979400086720376),
 ('imperdible', 0.6989700043360189),
 ('proximo', 0.6989700043360189),
 ('rousseff', 0.0),
 ('contra', 0.6989700043360189),
 ('presidencia', 0.6989700043360189),
 ('defiende', 0.6989700043360189),
 ('mujeres', 0.6989700043360189),
 ('evento', 0.6989700043360189),
 ('silenciaran', 0.6989700043360189),
 ('senado', 0.3979400086720376),
 ('volver', 0.6989700043360189),
 ('avanza', 0.6989700043360189),
 ('rivales', 0.6989700043360189),
 ('revela', 0.6989700043360189),
 ('ultimo', 0.6989700043360189),
 ('brasil', 0.2218487496163564),
 ('dilma', 0.09691001300805642),
 ('destitucion', 0.6989700043360189),
 ('discriminatorias', 0.6989700043360189)]
```

Nota que los términos que aparecen en todos los documentos de la colección tienen asociada una relevancia de 0!! :)

In [256]:
```
tf_idf = []
for i in range(len(idf_values)):
   tf_idf.append([(doc*1.0)/idf_values[i] if idf_values[i] > 0 else 0.0 for do
c in tf[i]])

zip(terms_wo_stopwords, tf_idf)
```

Out[256]:
```
[('hacia', [0.0, 0.0, 2.51294159473206, 0.0, 2.51294159473206]),
 ('ante', [1.430676558073393, 0.0, 0.0, 0.0, 0.0]),
 ('actitudes', [0.0, 0.0, 0.0, 0.0, 1.430676558073393]),
 ('intento', [0.0, 0.0, 0.0, 1.430676558073393, 0.0]),
 ('juicio', [0.0, 2.51294159473206, 2.51294159473206, 0.0, 0.0]),
 ('imperdible', [0.0, 1.430676558073393, 0.0, 0.0, 0.0]),
 ('proximo', [0.0, 1.430676558073393, 0.0, 0.0, 0.0]),
 ('rousseff', [0.0, 0.0, 0.0, 0.0, 0.0]),
 ('contra', [0.0, 1.430676558073393, 0.0, 0.0, 0.0]),
 ('presidencia', [0.0, 0.0, 0.0, 1.430676558073393, 0.0]),
 ('defiende', [1.430676558073393, 0.0, 0.0, 0.0, 0.0]),
 ('mujeres', [0.0, 0.0, 0.0, 0.0, 1.430676558073393]),
 ('evento', [0.0, 1.430676558073393, 0.0, 0.0, 0.0]),
 ('silenciaran', [1.430676558073393, 0.0, 0.0, 0.0, 0.0]),
 ('senado', [2.51294159473206, 0.0, 2.51294159473206, 0.0, 0.0]),
 ('volver', [0.0, 0.0, 0.0, 1.430676558073393, 0.0]),
 ('avanza', [0.0, 0.0, 1.430676558073393, 0.0, 0.0]),
 ('rivales', [1.430676558073393, 0.0, 0.0, 0.0, 0.0]),
 ('revela', [0.0, 0.0, 0.0, 0.0, 1.430676558073393]),
 ('ultimo', [0.0, 0.0, 0.0, 1.430676558073393, 0.0]),
 ('brasil',
  [0.0, 4.507575551943847, 4.507575551943847, 0.0, 4.507575551943847]),
 ('dilma',
  [10.31885115851617,
   10.31885115851617,
   10.31885115851617,
   10.31885115851617,
   0.0]),
 ('destitucion', [0.0, 0.0, 0.0, 0.0, 1.430676558073393]),
 ('discriminatorias', [0.0, 0.0, 0.0, 0.0, 1.430676558073393])]
```

Si tenemos el query de consulta **"Arrestan al diputado que estuvo detrás del juicio político contra Dilma"** (El Financiero (http://www.elfinanciero.com.mx/mundo/arrestan-al-diputado-que-estuvo-detras-del-juicio-politico-contra-dilma.html)) nuestro TF/IDF devolvería...

- Arrestan = 0
- al = 0 --> deberemos agregarla al stopword!
- diputado = 0
- que = 0
- estuvo = 0
- detras = 0
- del = 0
- juicio = [0.0, 2.51294159473206, 2.51294159473206, 0.0, 0.0]
- politico = 0
- contra = [0.0, 1.430676558073393, 0.0, 0.0, 0.0]
- dilma = [10.31885115851617,10.31885115851617,10.31885115851617,10.31885115851617,0.0]

d1 = $0.0 + 0.0 + 10.31 = 10.31$

d2 = $2.51 + 1.43 + 10.31 = 14.25$

d3 = $2.51 + 0.0 + 10.31 = 12.86$

d4 = $0.0 + 0.0 + 10.31 = 10.31$

d5 = $0.0 + 0.0 + 0.0 = 0.0$

Los documentos ordenados por relevancia con respecto a nuestro query:

1. d2: "El juicio contra Dilma Rousseff, el próximo evento imperdible en Brasil"
2. d3: "El Senado de Brasil avanza hacia el juicio de Dilma Rousseff"
3. d1: "Dilma Rousseff se defiende ante sus rivales en el senado: 'No me silenciarán'"
4. d4: 'El último intento de Dilma Rousseff por volver a la presidencia'
5. d5: 'La destitución de Rousseff revela actitudes discriminatorias hacia las mujeres en Brasil'

In [259]:
```
q='Arrestan al diputado que estuvo detrás del juicio político contra Dilma'

#eliminar acentos
q_wo_accents = quitar_acentos(q)
#tokenizar
q_tokens = tokenizacion(q_wo_accents)
q_unique_tokens = palabras_unicas(q_tokens)
#eliminar puntuacion
q_wo_punctuation = [eliminar_puntuacion(term) for term in q_unique_tokens]
#eliminar numeros
q_wo_numbers = [eliminar_numeros(term) for term in q_wo_punctuation]
#eliminar espacios
q_wo_spaces = [eliminar_espacios(term) for term in q_wo_numbers]
#a minusculas
q_wo_lower = [a_minusculas(term) for term in q_wo_spaces]
```

Ahora si... Scikit learn ya tiene todo esto listo para usarse :P ... PERO los stop words que trae por default son solo para inglés

(ノ °□°)ノ ⌒ ┻━┻ ... pero puedes pasarle tu propia lista de stop words

In [241]:
```
import sklearn as sl
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_setting = TfidfVectorizer(encoding='utf-8', strip_accents='unicode',
                analyzer='word', lowercase=True, stop_words=my_stop_words,
                use_idf=True, min_df=0)

tfidf_matrix = tfidf_setting.fit_transform(docs_collection)
terms = tfidf_setting.get_feature_names()
```

In [242]:
```
print tfidf_matrix
```

```
  (0, 21)       0.438724228779
  (0, 20)       0.353959945346
  (0, 19)       0.209054445715
  (0, 18)       0.438724228779
  (0, 7)        0.247169577713
  (0, 5)        0.438724228779
  (0, 1)        0.438724228779
  (1, 19)       0.200575845628
  (1, 16)       0.420930934444
  (1, 13)       0.339604427513
  (1, 11)       0.420930934444
  (1, 9)        0.420930934444
  (1, 7)        0.237145146058
  (1, 4)        0.420930934444
  (1, 3)        0.281902352559
  (2, 20)       0.406162115569
  (2, 19)       0.239885888381
  (2, 13)       0.406162115569
  (2, 10)       0.406162115569
  (2, 7)        0.283622257004
  (2, 3)        0.337151246047
  (2, 2)        0.503427473235
  (3, 23)       0.469093038892
  (3, 22)       0.469093038892
  (3, 19)       0.223525346452
  (3, 15)       0.469093038892
  (3, 12)       0.469093038892
  (3, 7)        0.2642788356
  (4, 19)       0.189446450964
  (4, 17)       0.397574650037
  (4, 14)       0.397574650037
  (4, 10)       0.320760724316
  (4, 8)        0.397574650037
  (4, 6)        0.397574650037
  (4, 3)        0.266260376685
  (4, 0)        0.397574650037
```

La representación es mucho más eficiente pues en lugar de tener una matriz con muchos 0s solo guardamos el lugar y valor de valores diferentes a 0.

In [243]:
```
for col in tfidf_matrix.nonzero()[1]:
    print terms[col], ' - ', tfidf_matrix[0, col]
```

```
silenciaran  -  0.438724228779
senado  -  0.353959945346
rousseff  -  0.209054445715
rivales  -  0.438724228779
dilma  -  0.247169577713
defiende  -  0.438724228779
ante  -  0.438724228779
rousseff  -  0.209054445715
proximo  -  0.0
juicio  -  0.0
imperdible  -  0.0
evento  -  0.0
dilma  -  0.247169577713
contra  -  0.0
brasil  -  0.0
senado  -  0.353959945346
rousseff  -  0.209054445715
juicio  -  0.0
hacia  -  0.0
dilma  -  0.247169577713
brasil  -  0.0
avanza  -  0.0
volver  -  0.0
ultimo  -  0.0
rousseff  -  0.209054445715
presidencia  -  0.0
intento  -  0.0
dilma  -  0.247169577713
rousseff  -  0.209054445715
revela  -  0.0
mujeres  -  0.0
hacia  -  0.0
discriminatorias  -  0.0
destitucion  -  0.0
brasil  -  0.0
actitudes  -  0.0
```

Probemos con nuestro query de consulta **"Arrestan al diputado que estuvo detrás del juicio político contra Dilma"**

In [246]:
```
q_wo_accents
```

Out[246]: 'Arrestan al diputado que estuvo detras del juicio politico contra Dilma'

In [244]:
```
response = tfidf_setting.transform([q_wo_accents])
print response
```

```
  (0, 13)       0.575062556088
  (0, 7)        0.401565123442
  (0, 4)        0.712775215773
```

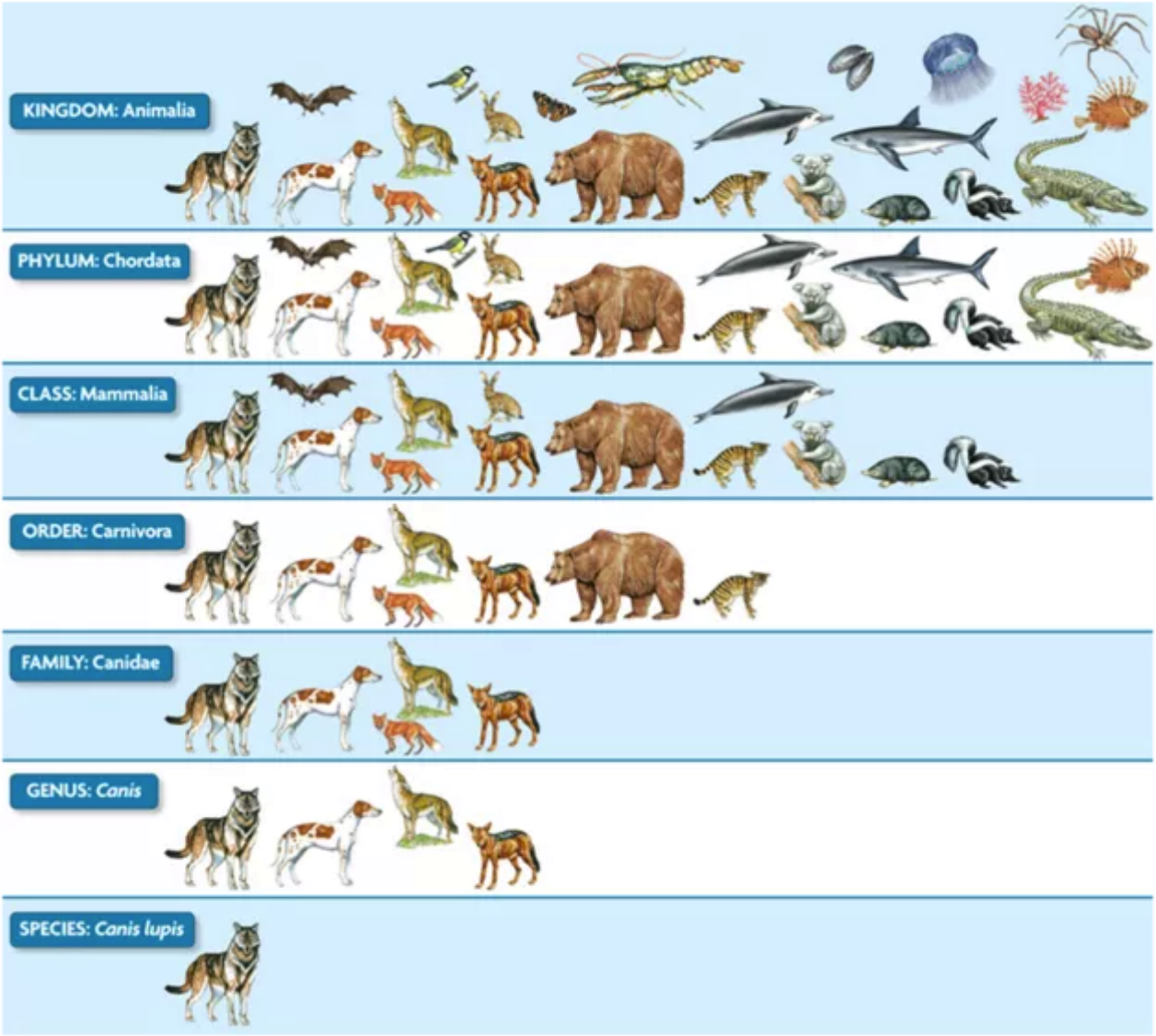
In [245]:
```
for col in response.nonzero()[1]:
    print terms[col], ' - ', response[0, col]
```

```
juicio  -  0.575062556088
dilma  -  0.401565123442
contra  -  0.712775215773
```

Caso de Negocio

Taxonomía

Organización de temas de un dominio específico



Ontología

Es una taxonomía en donde además definimos tipos, propiedades y relaciones entre los miembros de la taxonomía... se ocupan mucho en NLP para definir contextos y estructuras. Por ejemplo, partes de una oración.

Problema

En una empresa de contenido editorial que tiene marcas de deportes, moda, negocios, noticias, estilo de vida y noticias de celebridades a.k.a. *chismes*, se requiere de:

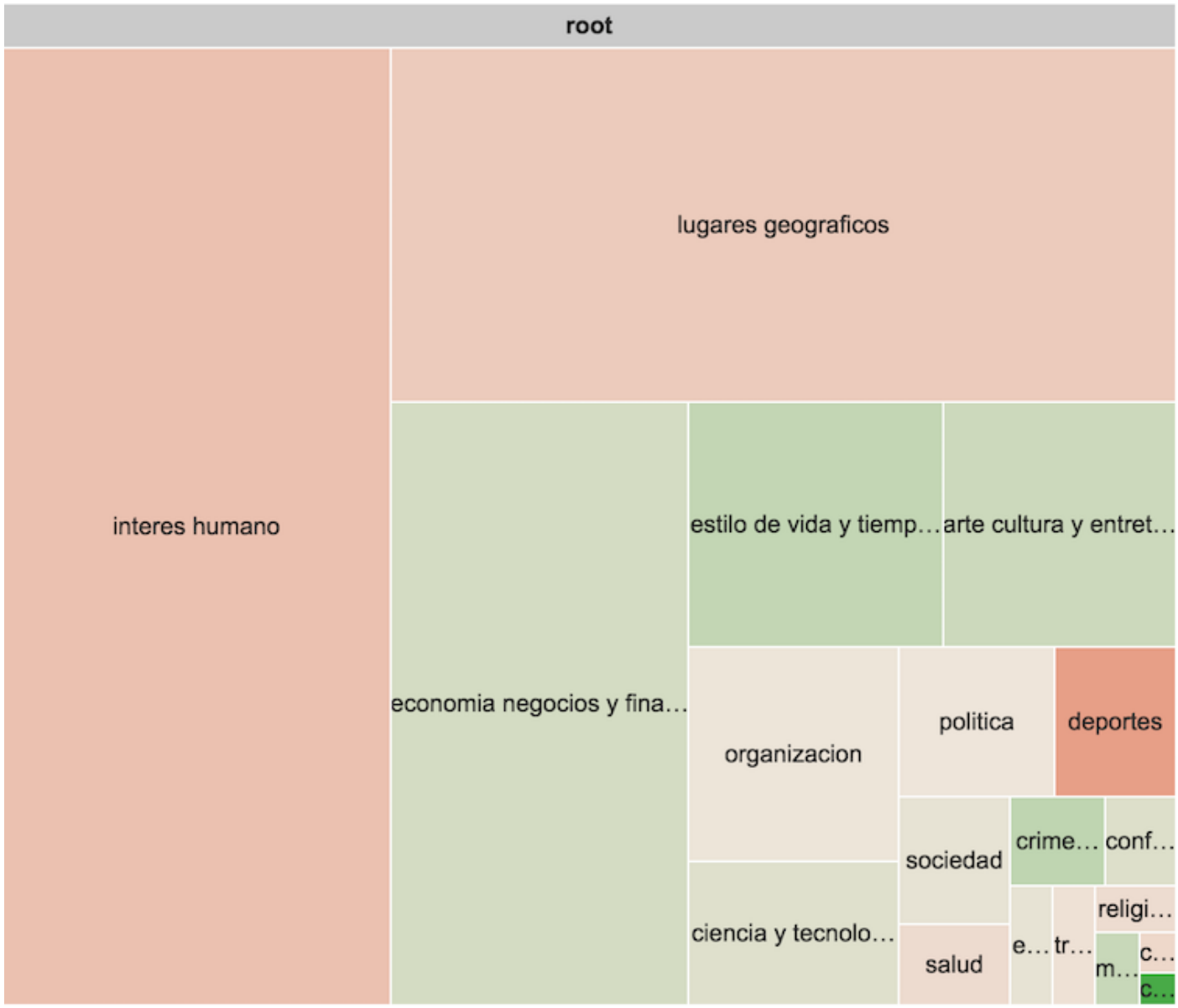
- etiquetar el contenido para mejorar el SEO basado en las etiquetas generadas por los editores
- recomendar las etiquetas asociadas a un contenido nuevo al momento que el editor guarda el contenido

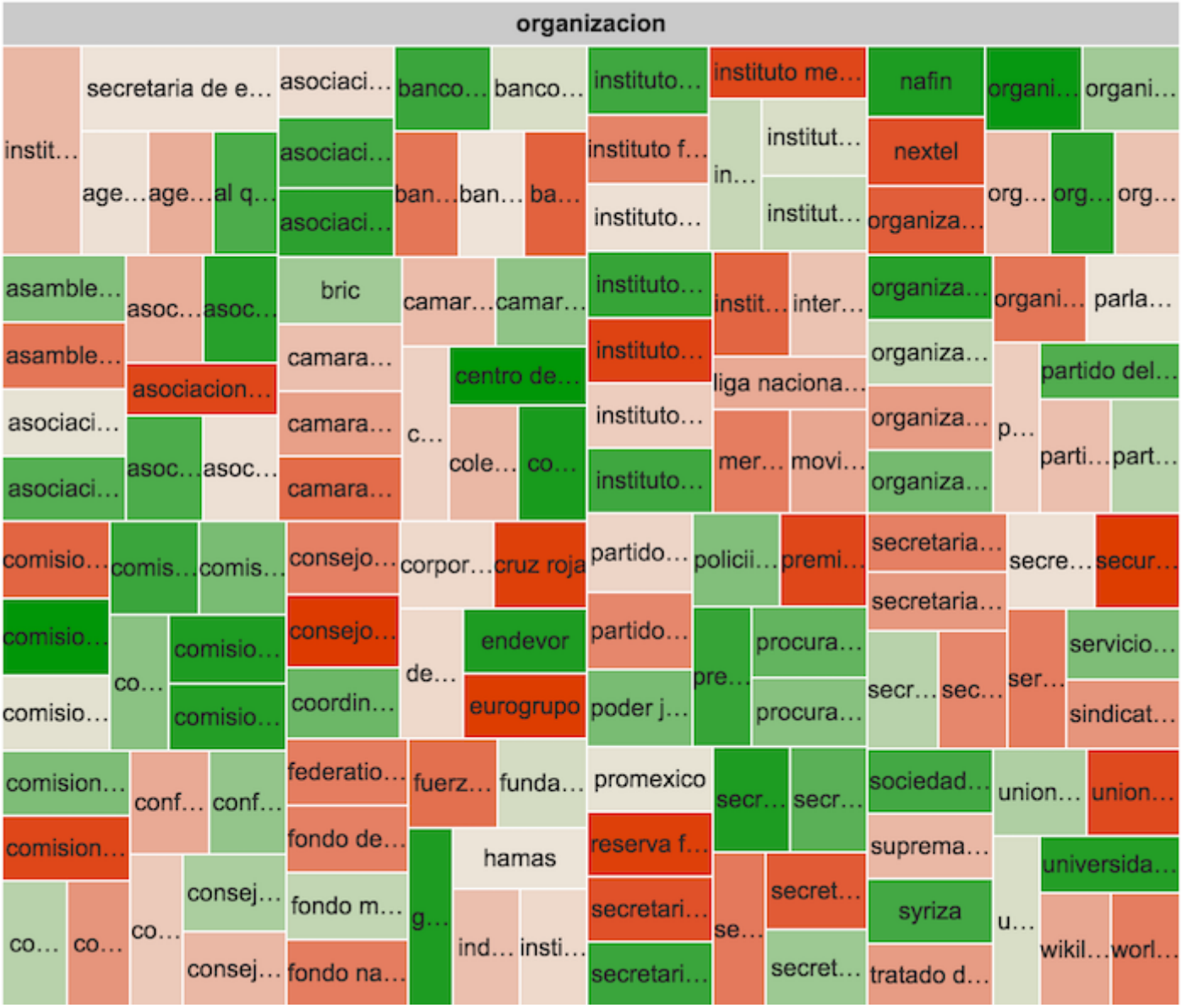
- identificar el tema en el que es mejor 'podar' una etiqueta para que sea lo *suficientemente* genérico
- clasificar de manera automática las notas editoriales que ya existen
- clasificar de manera automática las notas editoriales al momento de creación —sugerencia de etiquetas—
- organización del contenido editorial existente
  - evitar duplicidad
  - aprovechar investigaciones de contenidos escritos anteriormente
  - ontología de todo el contenido de casa editorial
- ... predicción de éxito en notas (por tema asociado)
- ... brinda más información para identificación de audiencias
- ... sistema de recomendación de contenidos, al lector
- ... sistema de recomendación de contenidos, al editor

Árbol de etiquetas

- Los editores de la casa editorial hicieron un árbol jerárquico de etiquetas basados en los temas de los que escriben en sus diferentes marcas y en el árbol estándar internacional de noticias IPTC (http://show.newscodes.org/index.html?newscodes=medtop&lang=en-GB&startTo=Show)
- Tenemos 3,942 etiquetas con 19 temas generales
- Máximo 8 niveles
- En promedio 4 niveles, depende del tema

~10k contenidos de diferentes marcas existentes desde el año 2010: moda, noticias, estilo de vida, deportes, *chismes*


root
interes humano
lugares geograficos
economia negocios y fina…
estilo de vida y tiemp…
arte cultura y entret…
organizacion
politica
deportes
sociedad
crime…
conf…
ciencia y tecnolo…
salud
e…
tr…
religi…
m…
c…
c…



Elementos del IR

1. ¿Cuál es la colección?
2. ¿Qué tamaño tiene?
3. ¿Cuál sería el query de consulta?
4. ¿Qué modelo ocupar para medir relevancia?
5. Justificación de decisiones tomadas...

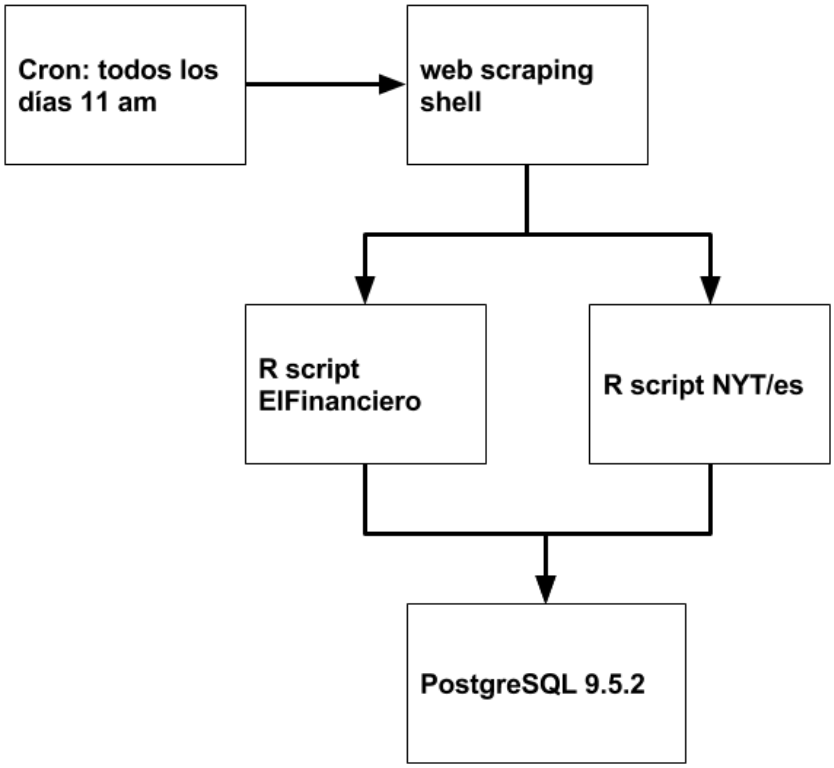
1. La colección consiste en el árbol jerárquico de etiquetas generado por los editores por lo que cada etiqueta se vuelve un documento
2. 3,942 **documentos**
3. El query es el contenido que queremos etiquetar
4. BM25

5. Siempre JUSTIFIQUEN sus tomas de decisiones y déjenlo plasmado en un documento ejecutivo (Rmd). En nuestro caso decidimos *voltear* qué es la colección y qué es el query ya que la tasa de crecimiento del árbol de etiquetas es mucho más lenta que la generación de contenidos, por cada nuevo contenido hay que buscar en ~4k documentos la relevancia en 1 solo query, al revés, por cada nuevo contenido tengo que volver a querear los ~4k etiquetas en >~10k documentos...

Si hubieramos tomado el query como la etiqueta estaríamos encontrando los documentos que probablemente están más relacionados a la misma sin embargo, nuestro cambio busca por documento cuál es la etiqueta más relevante... es una sutil diferencia pero ayuda a concebir el problema de manera más orgánica y es muy fácil de comprender para los editores y tomadores de decisión a.k.a. el consejo... CxO's.

El prototipo fue realizado en R mientras que la puesta en producción utilizó Hadoop y Spring XD como orquestador con un pequeño cluster de 4 máquinas con 8 GB cada una. El tiempo total para etiquetar todo el contenido fue de ~45 min.

Si bien en nuestro caso de negocio nuestro contenido ya existía *limpio* (obvio no!) y digitalizado, en muchas ocasiones tendrás que obtener el contenido tu mismo haciendo web scraping SelectorGadget (https://chrome.google.com/webstore/detail/selectorgadget/mhjhnkcfbdhnjickkkdbjoemdmbfginb). R tiene la librería rvest (https://cran.r-project.org/web/packages/rvest/rvest.pdf) y Python el módulo Beautiful Soup (https://www.crummy.com/software/BeautifulSoup/bs4/doc/).



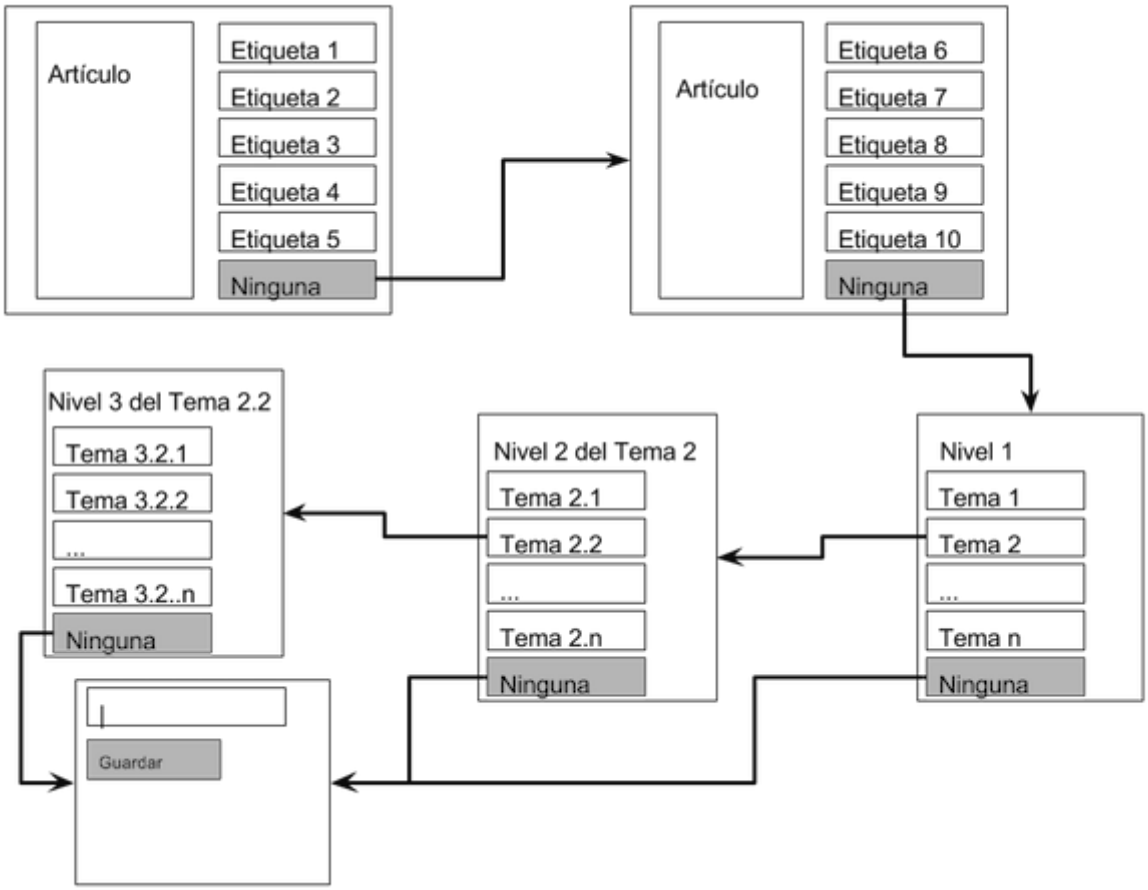
¿Cómo sabemos que las etiquetas sugeridas fueron correctas?

Una vez que el contenido ha sido etiquetado, se requiere de humanos que verifiquen si la etiqueta y abstracción del tema fue el adecuado.

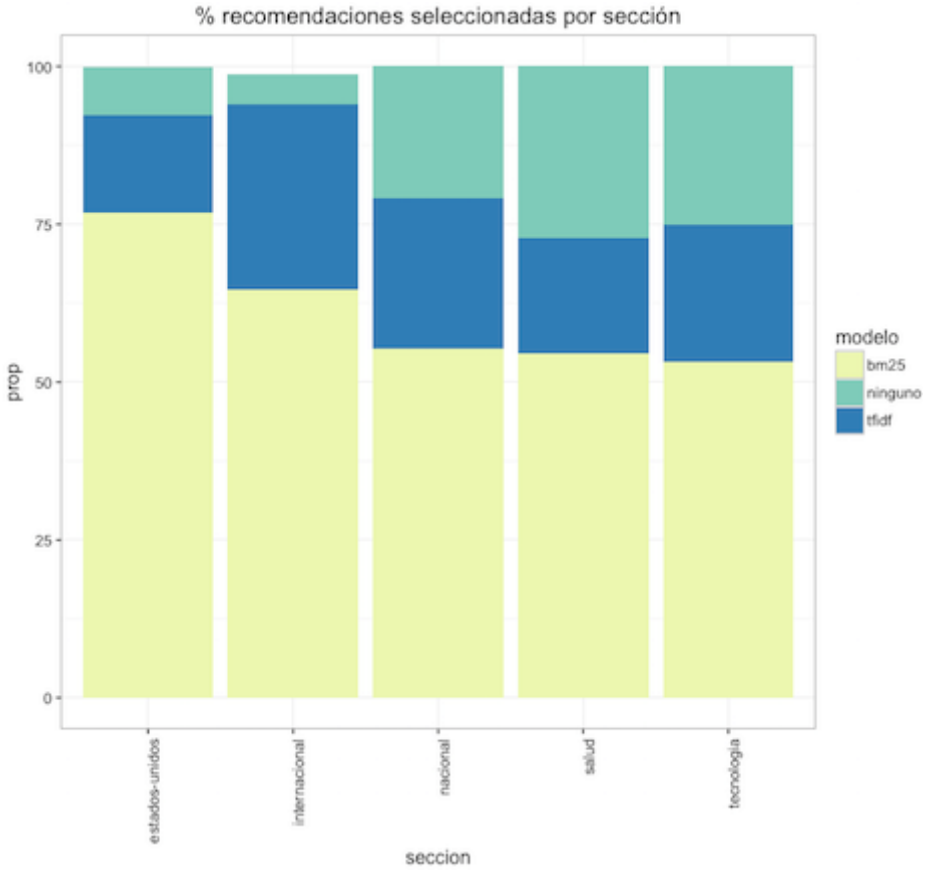
- En la vida real: No hay personas dispuestas a leer contenidos y ver clasificaciones sin incentivo alguno (generalmente económico) y es por eso que se recomienda implementar al menos 2 modelos y comparar resultados... el mismo equipo debe estar incluido en la revisión de los resultados... y presentar las conclusiones a los principales interesados del producto.
- En nuestro caso: Hicimos revisiones aleatorias de cada marca sin embargo, hasta que el editor consultaba algún contenido específico (por otras razones) verificaba la etiqueta y manda sus comentarios al equipo... O.o
- En el caso ideal: Tendríamos a los expertos de negocio (número impar!) ayudándo a verificar si la etiqueta y abstracción fue la correcta para realizar un voto de expertos... y hacer adecuaciones sobre todo a la parte de abstracción de los temas.

¿Cómo mejorar el modelo?

- Verificar los parámetros de tuneo del modelo (if any!)
- Permitir tener una opción digital que registre las recomendaciones del dueño del producto una vez que lo está usando o revisando
- Generar un clasificador de texto basado en algoritmos de machine learning... una vez que generamos el set de datos etiquetado a través de un sistema de IR podemos ocupar estos documentos como información para un clasificador y por lo tanto se pueden ocupar los algoritmos de ML correspondientes a clasificación


Artículo
Etiqueta 1
Etiqueta 2
Etiqueta 3
Etiqueta 4
Etiqueta 5
Ninguna
Artículo
Etiqueta 6
Etiqueta 7
Etiqueta 8
Etiqueta 9
Etiqueta 10
Ninguna
Nivel 3 del Tema 2.2
Tema 3.2.1
Tema 3.2.2
...
Tema 3.2..n
Ninguna
Nivel 2 del Tema 2
Tema 2.1
Tema 2.2
...
Tema 2.n
Ninguna
Nivel 1
Tema 1
Tema 2
...
Tema n
Ninguna
Guardar

Resultados



Referencias

- The Anatomy of a Large-Scale Hypertextual Web Search Engine (http://infolab.stanford.edu/pub/papers/google.pdf)
- Introduction to Information Retrieval (http://nlp.stanford.edu/IR-book/)
- The Stanford Natural Language Processing Group (http://nlp.stanford.edu/)
- Laboratorio de Lenguaje Natural y Procesamiento de Texto, IPN (http://nlp.cic.ipn.mx/lab-Spanish.htm)
- Grupo de Ingeniería Lingüística (GIL), UNAM (http://grupos.iingen.unam.mx/iling/es-mx/Paginas/default.aspx)
- Deep learning para Clasificación de texto con tensor flow (http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/)

Datasets para jugar

- Hillary Clinton emails, Kaggle (https://www.kaggle.com/kaggle/hillary-clinton-emails)

In [ ]: