# Prácticas de SAR

Práctica 3: El Mono Infinito

# El mono infinito

## Descripción del problema

*El teorema del mono infinito afirma que un mono pulsando teclas* ***al azar*** *sobre un teclado durante* ***un periodo de tiempo infinito*** *casi seguramente podrá escribir finalmente cualquier libro que se halle en la Biblioteca Nacional de Francia.*

# El mono infinito

## Objetivo de la práctica

*Ya que no tenemos tanto tiempo, crearemos un programa en Python que procese documentos y que utilice la información extraída de ellos para ayudar al mono a escribir su libro.*

***¡ Vamos a crear un modelo de lenguaje !***

# Modelo de lenguaje

# Modelo de lenguaje

Un objetivo importante en el Procesamiento del Lenguaje Natural es asignar una probabilidad a una secuencia de palabras (una frase):

$$P(w) = P(w_1, w_2, \ldots, w_{|w|})$$

Un objetivo muy relacionado con el anterior es el cálculo de la probabilidad de la próxima palabra dadas las palabras anteriores:

$$P(w_n | w_1, w_2, \ldots, w_{n-1})$$

Un modelo de lenguaje es un modelo computacional capaz de calcular ambas probabilidades.

# Modelo de lenguaje

Para calcular la probabilidad de una frase $P(w)$ podemos utilizar la regla de la cadena de Markov:

$$P(B|A) = \frac{P(A,B)}{P(A)}; P(A,B) = P(A) \cdot P(B|A)$$

De forma general:

$$P(w_1, w_2, \ldots, w_{|w|}) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdots P(w_n|w_1, \ldots, w_{n-1}) =$$

$$P(w_1, w_2, \ldots, w_{|w|}) = \prod_{i=1}^{|w|} P(w_i|w_1, \ldots, w_{i-1})$$

**Ejemplo (w = sueña el rey que es rey)**

$$P(w) = P(\textit{sueña el rey que es rey}) = P(\textit{sueña}) \cdot P(\textit{el}|\textit{sueña}) \cdot P(\textit{rey}|\textit{sueña el}) \cdot$$

$$P(\textit{que}|\textit{sueña el rey}) \cdot P(\textit{es}|\textit{sueña el rey que}) \cdot P(\textit{rey}|\textit{sueña el rey que es})$$

# Modelo de lenguaje de *n-gramas*

Una simplificación habitual es hacer depender la probabilidad de una palabra unicamente de las $n-1$ palabras anteriores.

$$P(w_1, w_2, \ldots, w_{|w|}) = \prod_{i=1}^{|w|} P(w_i | w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{|w|} P(w_i | w_{i-(n-1)}, \ldots, w_{i-1})$$

## Ejemplo (w = sueña el rey que es rey)

- Considerando bi-gramas (n=2):

$$P(w) \approx P(sueña) \cdot P(el|sueña) \cdot P(rey|el) \cdot P(que|rey) \cdot P(es|que) \cdot P(rey|es)$$

- Considerando tri-gramas (n=3):

$$P(w) \approx P(sueña) \cdot P(el|sueña) \cdot P(rey|sueña\ el) \cdot$$

$$P(que|el\ rey) \cdot P(es|rey\ que) \cdot P(rey|que\ es)$$

# Modelo de lenguaje de *n-gramas*

Podemos calcular la probabilidad de aparición de una palabra dadas las n-1 palabras anteriores como:

$$P(w_n|w_1, w_2, \ldots, w_{n-1}) = \frac{C(w_1, w_2, \ldots, w_n)}{C(w_1, w_2, \ldots, w_{n-1})}$$

Donde, $C(w_i, w_{i+1}, \ldots, w_{i+k-1})$ es el número de veces que la secuencia $w_i\ w_{i+1} \ldots w_{i+k-1}$ ha aparecido en los textos utilizados para aprender el modelo.

# Modelo de lenguaje de *n-gramas*

Se suele añadir un marcador de inicio y de fin de frase para tener más información. En casos donde $n > 2$, se pueden añadir n-1 símbolos iniciales.

## Ejemplo (w = sueña el rey que es rey)

- Considerando bi-gramas (n=2):

$$P(w) \approx P(sueña|\$) \cdot P(el|sueña) \cdot P(rey|el) \cdot$$

$$P(que|rey) \cdot P(es|que) \cdot P(rey|es) \cdot P(\$|rey)$$

- Considerando tri-gramas (n=3):

$$P(w) \approx P(sueña|\$\ \$) \cdot P(el|\$\ sueña) \cdot P(rey|sueña\ el) \cdot$$

$$P(que|el\ rey) \cdot P(es|rey\ que) \cdot P(rey|que\ es) \cdot P(\$|es\ rey)$$

# Modelos de lenguaje *neuronales*

En la actualidad los modelos de lenguaje basados en redes neuronales profundas has substituido a los basados en n-gram en la mayoría de casos de uso.
Modelos como **BERT** (Bidirectional Encoder Representations from Transformers), **BART** (Bidirectional and Auto-Regressive Transformers) o **GPT** (Generative Pre-trained Transformer), aprendidos con enormes colecciones de documentos, son el estado del arte para tareas de recuperación de información semántica y procesamiento del lenguaje natural.

# Ejercicio

# ¿Qué debo hacer?

En esta prácticas vamos a hacer tres tareas distintas. Para cada una de ellas se proporciona un programa en python que no se debé modificar:

1. Aprender modelos de lenguaje de n-gramas a partir de una colección de documentos y un valor **n** mayor que 1. `SAR_p3_monkey_learner.py`
2. Mostrar información, por pantalla o en fichero, de los modelos aprendidos. `SAR_p3_monkey_info.py`
3. Generar texto utilizando la información lingüística almacenado en los modelos de lenguaje utilizando opcionalmente una *entradilla*.

   `SAR_p3_monkey_evolved.py`.

# ¿Qué debo hacer?

Los tres programas anteriores utilizan la clase `Monkey` de la librería `SAR_p3_monkey_lib.py`.

Nosotros te proporcionamos una plantilla en la que debes **completar y DOCUMENTAR** los métodos siguientes:

- **compute_lm(self, filenames:List[str], lm_name:str, n:int)**:: recibe una lista de nombres de fichero, los procesa, extrae las frases y llama a `index_sentence` para crear el modelo de lenguaje.
- **index_sentence(self, sentence:str)**: recibe una frase y añade sus estadísticas a los modelos de lenguaje.
- **generate_sentences(self, n:Optional[int], nsentences:int=10, prefix:Optional[str]=None)**: genera frases *aleatorias* utilizando un modelo de lenguaje.

# SAR_p3_monkey_learner.py

```
prompt> python SAR_p3_monkey_learner.py --help

usage: SAR_p3_monkey_learner.py [-h] [-l LM_NAME] -f LM_FILENAME [-n N]
                                files [files ...]

Compute a 3-gram language model from text files.

positional arguments:
  files                 text files.

optional arguments:
  -h, --help            show this help message and exit
  -l LM_NAME, --lm_name LM_NAME
                        name of the language model.
  -f LM_FILENAME, --filename LM_FILENAME
                        name of the file to save the language model.
  -n N                  maximum value of n for n-grams.
```

# SAR_p3_monkey_learner.py

El programa `SAR_p3_monkey_learner.py`:

1. Recibe como argumento: el nombre de uno o más ficheros de texto, un valor para **n** (n>1) y un nombre de fichero de destino. Opcionalmente se le puede dar nombre al modelo de lenguaje.
2. Para cada fichero de entrada, lo divide en frases, las tokeniza y crea un modelo estadístico donde acumula estadísticas de qué palabras siguen a otras.
3. El resultado serán modelos de lenguaje de 2 a **n**-gramas.
4. Guarda los modelos en un fichero binario.

# SAR_p3_monkey_learner.py

**Tokenización:**

- Las frases se separan por “.”, “;”, “!”, “?” o dos saltos de línea.
- Para cada frase:
  1. Se eliminan todos los símbolos no alfanuméricos y se pasa el contenido a minúsculas.
  2. Los tokens son las palabras restantes separadas por espacios.
  3. Se añade un “$” como símbolo final de frase.
  4. Dependiendo del tamaño del n-grama, añadiremos n-1 “$” delante de la frase.

# SAR_p3_monkey_learner.py

**Creación de los modelos de lenguaje:**
En el mismo proceso de aprendizaje se crean varios modelos de lenguaje. Dada una **n** se crearán n-1 modelos distintos: 2-gramas, 3-gramas, ..., n-gramas.

- Cada modelo de i-gramas se guarda como un diccionario Python.

```
for i in range(2, n+1):
    self.info['lm'][i] = {}
```

- Como paso intermedio ese diccionario tendrá la estructura siguiente:
  - Las claves del diccionario serán tuplas de tokens del estilo $(w_1, \cdots, w_{i-1})$.
  - El valor asociado a una clave será otro diccionario para almacenar las veces que cada token aparece en el texto después de la clave.

# SAR_p3_monkey_learner.py

## Fichero `spam.txt` utilizado en los ejemplos:

```
Egg and Bacon;
Egg, sausage and Bacon;
Egg and Spam;
Spam Egg Sausage and Spam;
Egg, Bacon and Spam;
Egg, Bacon, sausage and Spam;
Spam, Bacon, sausage and Spam;
Spam, Egg, Spam, Spam, Bacon and Spam;
Spam, Spam, Spam, Egg and Spam;
Spam, Spam, Spam, Spam, Spam, Spam, Spam, baked beans, Spam, Spam, Spam and
     Spam;
Lobster Thermidor aux crevettes with a Mornay sauce, garnished with truffle
     pate, brandy and a fried egg on top and Spam
```

# SAR_p3_monkey_learner.py

## Estructura intermedia para self.info['lm'][3] tras el procesado de spam.txt:

```
{
    ('$', '$'): {'egg': 5, 'spam': 5, 'lobster': 1},
    ('$', 'egg'): {'and': 2, 'sausage': 1, 'bacon': 2},
    ('egg', 'and'): {'bacon': 1, 'spam': 2},
    ('and', 'bacon'): {'$': 2}, ('egg', 'sausage'): {'and': 2},
    ('sausage', 'and'): {'bacon': 1, 'spam': 3},
    ('and', 'spam'): {'$': 9},
    ('$', 'spam'): {'egg': 2, 'bacon': 1, 'spam': 2},
    ...,
    ('fried', 'egg'): {'on': 1},
    ('egg', 'on'): {'top': 1},
    ('on', 'top'): {'and': 1},
    ('top', 'and'): {'spam': 1}
}
```

# SAR_p3_monkey_learner.py

**Creación de los modelos de lenguaje:**

La estructura final del diccionario será ligeramente diferente:

- Las claves del diccionario seguirán siendo tuplas de tokens del estilo $(w_1, \cdots, w_{i-1})$.
- El valor asociado a cada clave será una tupla con dos elementos:
  - El primer elemento será el número de veces que la secuencia de tokens de la clave ha aparecido en los textos de entrenamiento.
  - El segundo elemento será una lista ordenada por frecuencia de pares token y número de veces que ese token ha aparecido despúes de la clave.

La estructura definitiva se puede conseguir a partir de la intermedia utilizando la función:

```
def convert_to_lm_dict(d: dict):
    for k in d:
        l = sorted(((y, x) for x, y in d[k].items()), reverse=True)
        d[k] = (sum(x for x, _ in l), l)
```

# SAR_p3_monkey_indexer.py

## Estructura definitiva de self.info['lm'][3] tras la llamada a `convert_to_lm_dict`

```
{
 ('$', '$'): (11, [(5, 'spam'), (5, 'egg'), (1, 'lobster')]),
 ('$', 'egg'): (5, [(2, 'bacon'), (2, 'and'), (1, 'sausage')]),
 ('egg', 'and'): (3, [(2, 'spam'), (1, 'bacon')]),
 ('and', 'bacon'): (2, [(2, '$')]),
 ('egg', 'sausage'): (2, [(2, 'and')]),
 ('sausage', 'and'): (4, [(3, 'spam'), (1, 'bacon')]),
 ('and', 'spam'): (9, [(9, '$')]),
 ('$', 'spam'): (5, [(2, 'spam'), (2, 'egg'), (1, 'bacon')]),
 ('spam', 'egg'): (3, [(1, 'spam'), (1, 'sausage'), (1, 'and')]),
 ...
 ('a', 'mornay'): (1, [(1, 'sauce')]),
 ('mornay', 'sauce'): (1, [(1, 'garnished')]),
 ('egg', 'on'): (1, [(1, 'top')]),
 ('on', 'top'): (1, [(1, 'and')]),
 ('top', 'and'): (1, [(1, 'spam')])
}
```

# SAR_p3_monkey_info.py

```
prompt> python SAR_p3_monkey_info.py --help

usage: SAR_p3_monkey_info.py [-h] [-i INFO_FILENAME] lm_filename

Compute a 3-gram language model from text files.

positional arguments:
  lm_filename           language model file.

optional arguments:
  -h, --help            show this help message and exit
  -i INFO_FILENAME, -info INFO_FILENAME
                        name of the file to save the language model
                        information
```

- Recibe como argumento el nombre de un fichero donde se ha guardado un modelo de lenguaje y opcionalmente el nombre de un fichero de destino y muestra información de los modelos de lenguaje contenidos en el fichero.
- Si no se proporciona un fichero de destino se muestra la información por pantalla.

# SAR_p3_monkey_info.py

## python SAR_p3_monkey_info.py spam.lm

```
####################
#       INFO       #
####################
language model name: None
filenames used to learn the language model: ['spam.txt']
####################
####################
#     2-GRAMS      #
####################
'$' =>  11  =>  spam:5, egg:5, lobster:1
'a' =>  2   =>  mornay:1, fried:1
'and'   =>  12  =>  spam:9, bacon:2, a:1
.
.
####################
#     4-GRAMS      #
####################
'$ $ $' =>  11  =>  spam:5, egg:5, lobster:1
'$ $ egg'   =>  5   =>  bacon:2, and:2, sausage:1
'$ $ lobster'   =>  1   =>  thermidor:1
.
.
```

# SAR_p3_monkey_evolved.py

```
prompt> python SAR_p3_monkey_evolved.py --help

usage: SAR_p3_monkey_evolved.py [-h] -f LM_FILENAME [-n N] [-s SENTENCES]
                                [-p PREFIX]

Generate sentences based on a language model.

optional arguments:
  -h, --help            show this help message and exit
  -f LM_FILENAME, --filename LM_FILENAME
                        name of the file with the language model.
  -n N                  n to use in the ngram model.
  -s SENTENCES, --sentences SENTENCES
                        number of sentences to produce.
  -p PREFIX, --prefix PREFIX
                        prefix to use in the sentence generation.
```

# SAR_p3_monkey_evolved.py

- **Funcionalidad**

1. Recibe el nombre de un fichero de modelos de lenguaje y, opcionalmente:
   - un valor para **n**, por defecto el mayor número posible.
   - un número de frases, por defecto 10,
   - un prefijo, por defecto cadena vacía.
2. Utiliza el modelo de lenguaje de **n**-gramas para generar frases.
3. Si se proporciona un prefijo, todas las frases comienzan por él.

# SAR_p3_monkey_evolved.py

## Ejemplo de ejecución

```
> python SAR_p3_monkey_evolved.py -f spam.lm -n 2 -s 3 -p "spam egg"
spam egg on top and spam spam spam
spam egg bacon sausage and spam spam egg sausage and spam
spam egg bacon

> python SAR_p3_monkey_evolved.py -f spam.lm -n 4 -s 3
egg and spam
spam spam spam and spam
egg bacon sausage and spam

> python SAR_p3_monkey_evolved.py -f spam.lm -n 5 -s 3 -p "spam egg"
spam egg sausage and spam
spam egg sausage and spam
spam egg spam spam bacon and spam
```

# SAR_p3_monkey_evolved.py

## ¿Cómo se inicia la generación de cada frase?

Se genera una historia inicial como una tupla de talla n-1.

- Si **no** se proporciona prefijo, se inicia con una tupla de n-1 '$'.
  - n=4 y prefijo = "" $\longrightarrow$ ini = ("$", "$", "$")
- Si se proporciona prefijo, se limpia, se separa por palabras y se crea el inicio rellenando con '$' por la izquierda.
  - n=4 y prefijo = "spam, egg" $\longrightarrow$ ini = ("$", "spam", "egg")

## ¿Cómo se elige cada palabra siguiente?

- Dado un inicio, se eligen la siguiente palabra de forma "**aleatoria ponderada**" entre las sucesoras posibles teniendo en cuenta el número de veces que ha aparecido.
- Se actualiza el inicio descartando la primera palabra y añadiendo la nueva.

# SAR_p3_monkey_evolved.py

## ¿Cuándo se termina una frase?

Cuando la palabra siguiente elegida es la marca de final de frase “$”, o se llega a un número máximo de palabras, 50 en nuestro caso.

# Cosas útiles

# Guardar objetos python en un fichero

## pickle

```
import pickle

def save_object(obj, filename):
    with open(file_name, 'wb') as fh:
        pickle.dump(obj, fh)

def load_object(filename):
    with open(file_name, 'rb') as fh:
        obj = pickle.load(fh)
    return obj
```

# Números “aleatorios” en python

## librería random

```
import random

random.randint(a, b)
  Return a random integer N such that a <= N <= b.

random.choice(seq)
  Return a random element from the non-empty sequence seq. If seq
     is empty, raises IndexError.

choices(population, weights=None, *, cum_weights=None, k=1)
  Return a k sized list of population elements chosen with
     replacement.
  If the relative weights or cumulative weights are not specified,
  the selections are made with equal probability.
```