**ESCUELA POLITÉCNICA NACIONAL**

**ASIGNATURA DE RECUPERACIÓN DE LA INFORMACIÓN**

**PROYECTO FINAL: MODELO PROBABILÍSTICO**

**INTEGRANTES**

Armijos Macas Wilman Alberto

Caiza Jiménez Boris Javier

Contreras Andrade Milan Leonardo

Jiménez Reino Sergio Andrés

Taco Apolo César Paolo

Tejada Bustillo Jonathan Moisés

Vivas Corrales Gabriel Alexander

**Quito, 07 marzo de 2022**

**Tabla de contenido**

[Introducción 3](#_Toc97568220)

[Marco teórico 3](#_Toc97568221)

[Desarrollo 4](#_Toc97568222)

[Instalación de librerías 4](#_Toc97568223)

[Importación de librerías 4](#_Toc97568224)

[Variables globales 4](#_Toc97568225)

[Limpieza 5](#_Toc97568226)

[Función Limpiar 5](#_Toc97568227)

[Función contar\_terminos 5](#_Toc97568228)

[Función ponerId 5](#_Toc97568229)

[Función indexarDocumentos 5](#_Toc97568230)

[Modelo 6](#_Toc97568231)

[Función encontrar\_ci 6](#_Toc97568232)

[Función modelo\_probabilistico 7](#_Toc97568233)

[Función ponerId2 7](#_Toc97568234)

[Función recuperarDocumentos 7](#_Toc97568235)

[Función consulta\_usuario 8](#_Toc97568236)

[Uso del Modelo 8](#_Toc97568237)

[Resultados 8](#_Toc97568238)

[Obtención de documentos 9](#_Toc97568239)

[Creación de diccionario 9](#_Toc97568240)

[Cálculo de CI 9](#_Toc97568241)

[Consulta de prueba 1 10](#_Toc97568242)

[Consulta de prueba 2 10](#_Toc97568243)

[Consulta de prueba 3 11](#_Toc97568244)

[Bibliografía 11](#_Toc97568245)

# Introducción

El modelo probabilístico es una representación abstracta de procesos el cual a partir de una colección de documentos trata de predecir si un documento es relevante o no y en qué grado. Este modelo toma en cuenta la incertidumbre que se añade a cada modelo en las fases de adaptación de documentos y consulta, a su vez calculando la probabilidad de que un documento sea relevante dada una consulta.

# Marco teórico

**Modelos de Recuperación de Información**

Los modelos de recuperación de información se conforman esencialmente de un método para hallar los documentos relevantes existentes en una colección en relación con cada necesidad informativa de los usuarios expresada mediante una consulta al Sistema de Recuperación de Información (SRI).

Para poder llevar a cabo esta tarea, cada modelo necesita desarrollar los siguientes aspectos básicos:

* Una representación para los documentos de la colección [1].
* Una representación de las consultas.
* Un algoritmo o proceso que permita discernir los documentos relevantes de los irrelevantes en relación con cada consulta, y su posible ordenación.

Los principales modelos de Recuperación de Información

* Modelo Binario
* Modelo Vectorial
* Modelo Probabilístico

**Modelo Probabilístico**

A la hora de representar los documentos de la colección, el modelo probabilístico es también un modelo binario, pues solamente considera dos posibilidades en relación con cada término:

* Valor 0 si el término está ausente en el documento.
* Valor 1 si el término está presente en el documento, independientemente de su frecuencia en dicho documento.

En consecuencia, los documentos de la colección son representados mediante una serie ordenada de ceros y unos.

En el modelo probabilístico, una consulta Q está compuesta de la enumeración de los términos de indización que deseamos encontrar en los documentos de la respuesta. Por tanto, una consulta Q tiene la misma representación que un documento de la colección; esto es, una serie ordenada de ceros y unos [1]. Los unos corresponderán a los términos de la colección que deseamos hallar en los documentos de la respuesta.

Por último, el proceso de recuperación consiste en los siguientes pasos:

* Cálculo de los coeficientes iniciales (o valor del estado de recuperación; RSV o Retrieval Status Value en inglés) correspondientes a cada término de la consulta.
* Cálculo de la similitud entre cada documento de la colección y la consulta, mediante la suma de los coeficientes iniciales correspondientes a los términos 28 presentes simultáneamente en el documento y en la consulta:
  + SIM(Di , Q) = Σ ci
* Mostrar al usuario los documentos ordenados en orden decreciente de su similitud con la consulta.
* Nuevo cálculo de los coeficientes conforme a la clasificación en relevantes e irrelevantes efectuada por el usuario.
* Nuevo cálculo de similitud entre cada documento de la colección y la consulta.
* Mostrar al usuario los documentos ordenados en orden decreciente de su similitud con la consulta [1].

# Desarrollo

El modelo fue desarrolla en el lenguaje de programación **Python** en su versión 3, este lenguaje fue seleccionado por su facilidad de aprendizaje, amplia cantidad de librerías de terceros, entendimiento del lenguaje de parte del equipo de trabajo y por la capacidad de procesamiento.

## Instalación de librerías

1. Instalación de librería spacy en entorno virtual.

!pip install spacy

1. Instalación de complementos de español.

!python -m spacy download es\_core\_news\_md

1. Instalación de complementos de inglés.

!python -m spacy download en\_core\_web\_md

1. Instalación de modulos en español.

!python -m spacy download es

1. Conexión con google drive (en caso de usar Google Colab este paso se puede omitir).

from google.colab import drive

drive.mount('/content/drive')

## Importación de librerías

El modelo debe recibir documentos que ya hayan sido normalizados y limpiados para evitar fallos al momento de realizar búsquedas.

Por este motivo se utilizó las librerías:

* **Spacy: “**biblioteca de código abierto para el procesamiento del lenguaje natural en Python” [2].
* **Nltk: “**Es una plataforma líder para crear programas de Python para trabajar con datos de lenguaje humano con un conjunto de bibliotecas de procesamiento de texto para clasificación, tokenización, lematización, etiquetado, análisis y razonamiento semántico**”** [3]**.**
* **Math:** Librería de Python que proporciona acceso a las funciones matemáticas definidas en el estándar de C [4].
* **SnowballStemmer:** Para invocar lematizadores si no conoce el lenguaje que se derivará en tiempo de ejecución. Alternativamente, si ya sabes el idioma, entonces puede invocar el lematizador específico del idioma directamente [5].

import spacy

import nltk

import math

from nltk import SnowballStemmer

## Variables globales

* Cambiar el valor de variables spanishstemmer y npl en caso de requerir otro idioma.
* Cambiar la variable path para establecer correctamente la ruta de los documentos.

spanishstemmer=SnowballStemmer('spanish')

nlp = spacy.load('es')

path = ""

## Limpieza

### Función Limpiar

Datos de entrada: cadena de caracteres a ser procesada.

Datos de salida: cada uno de los caracteres procesados.

Descripción: el texto de los documentos es lematizado y tokenizado a través de la librería y sus configuraciones de **spacy** y **ntlk**.

def limpiar(text):

  doc = nlp(text)

  tokens = [tok.lemma\_.lower() for tok in doc if not tok.is\_punct | tok.is\_stop | len(tok) < 3]

  stems = [spanishstemmer.stem(token) for token in tokens]

  return stems

### Función contar\_terminos

Datos de entrada: diccionario con los términos y sus documentos.

Datos de salida: lista con cada uno de los términos y las veces que aparece en cada documento.

Descripción: base a una lista de términos en un documento se cuenta cuantas veces se repite cada término.

def contar\_terminos(data):

  result=dict.fromkeys(data,0)

  for item in data:

    if item not in result.keys():

      result[item] = 1

    else:

      result[item] += 1

  return result

### Función ponerId

Datos de entrada: lista de los documentos extraídos.

Datos de salida: lista de los documentos con un identificador.

Descripción: agregar un identificador en este caso un número a cada documento para identificar la ubicación de las palabras en ese documento.

def ponerId(listaDocuemntos):

  id\_mas\_docuemntos= dict.fromkeys(documentos)

  for i in range(len(documentos)):

    id\_mas\_docuemntos[documentos[i]]=i+1

  return id\_mas\_docuemntos

### Función indexarDocumentos

Datos de entrada: la lista de los documentos extraídos.

Datos de salida: un diccionario con todos los términos y los documentos donde aparecen.

Descripción: se encarga de obtener los distintos documentos en los que se encuentra un término.

def indexarDocumentos(documentos):

  terminos=[]

  terminos\_doc=[]

  terminos\_docuementos = dict.fromkeys(documentos)

  for doc in documentos:

    terminos\_doc.clear

    with open(path+doc+".txt", 'r') as file:

      data = file.read().replace('\n', '')

      data\_limpia = limpiar(data)

      contar\_terminos

      terminos\_docuementos[doc]= contar\_terminos(data\_limpia)

      terminos = terminos + data\_limpia

  terminos = list(set(terminos))

  terminos = dict.fromkeys(terminos,[])

  lista=[]

  for item in terminos:

    lista=[]

    for termino\_doc in terminos\_docuementos:

      if item in terminos\_docuementos[termino\_doc]:

        lista = lista + [(documentos[termino\_doc],terminos\_docuementos[termino\_doc][item])]

    terminos[item]=lista

  return terminos

## Modelo

### Función encontrar\_ci

Datos de entrada: lista de documentos recuperados y numero de documentos relevantes.

Datos de salida: ci de todos los términos encontrados y lista de términos limpios.

Descripción: se calcula cada uno de los pesos CI de cada término encontrado dentro de cada documento.

def encontrar\_ci(documentos, documentosRelevantes):

  terminos=[]

  terminos\_documentos = dict.fromkeys(documentos)

  for key , value in documentos.items():

    data\_limpia = limpiar(value)

    print(data\_limpia)

    terminos\_documentos[key]= contar\_terminos(data\_limpia)

    terminos = terminos + data\_limpia

  terminos = sorted(list(set(terminos)))

  dict\_terminos\_ri = dict.fromkeys(terminos, 0)

  dict\_terminos\_ni\_ri = dict.fromkeys(terminos, 0)

  dict\_terminos\_ci = dict.fromkeys(terminos, 0)

  for key , value in dict\_terminos\_ri.items():

    for i in range(len(documentos.keys())):

      if (key in terminos\_documentos[i+1].keys()) and (i<documentosRelevantes):

        dict\_terminos\_ri[key] = dict\_terminos\_ri[key] + 1

      elif (key in terminos\_documentos[i+1].keys()) and (i>=documentosRelevantes):

        dict\_terminos\_ni\_ri[key]=dict\_terminos\_ni\_ri[key]+1

  for key  in dict\_terminos\_ri.keys():

    dict\_terminos\_ci[key] = math.log10(((dict\_terminos\_ri[key]+0.5)/(documentosRelevantes-dict\_terminos\_ri[key]+0.5))/((dict\_terminos\_ni\_ri[key]+0.5)/(len(documentos.keys())-documentosRelevantes-dict\_terminos\_ni\_ri[key]+0.5)))

  return dict\_terminos\_ci,terminos\_documentos

### Función modelo\_probabilistico

Datos de entrada: lista de valores CI, documentos procesados y la consulta.

Datos de salida: arreglo con la similitud entre la consulta y cada documento.

Descripción: se ejecuta el producto escalar entre la consulta y cada uno de los documentos, en este caso, directamente con los valores de CI calculados para cada uno.

def modelo\_probabilistico(ci, documentosLimpios, consulta):

  consulta\_arreglo=limpiar(consulta)

  similitud = dict.fromkeys(documentosLimpios,0)

  for key, value in documentosLimpios.items():

    contador = 0

    for item in value.keys():

      if item in consulta\_arreglo:

        similitud[key] = similitud[key] + ci[item]

        contador = contador+1

    if contador==0:

      similitud[key]="n/a"

  return similitud

### Función ponerId2

Datos de entrada: lista de documentos.

Datos de salida: diccionario con los documentos procesados.

Descripción: se recuperan los documentos como objetos, en lugar de como cadenas con los nombres.

def ponerId2(listaDocuemntos):

  ids = range(1,len(listaDocuemntos)+1)

  dict\_doc = dict.fromkeys(ids)

  for doc in dict\_doc.keys():

    dict\_doc[doc] = listaDocuemntos[doc-1]

  return dict\_doc

### Función recuperarDocumentos

Datos de entrada: Ninguno.

Datos de salida: lista con los objetos que apuntan a los documentos.

Descripción: se recupera cada uno de los documentos, en este caso, buscando en un directorio.

def recuperarDocumentos():

  with open("doc1.txt", 'r') as file:

      d1 = file.read().replace('\n', '')

  with open("doc2.txt", 'r') as file:

      d2 = file.read().replace('\n', '')

  with open("doc3.txt", 'r') as file:

      d3 = file.read().replace('\n', '')

  return d1,d2,d3

### Función consulta\_usuario

Datos de entrada:

Datos de salida:

Descripción:

def consulta\_usuario(q1):

  q1 = limpiar(q1)

  q1 = " ".join(q1)

  similitud = modelo\_probabilistico(ci,documentosLimpios,q1)

  print("Respuesta de la consulta: ",q1," =")

  for key, value in similitud.items():

    print("Sim(q, d%s)= %s"%(key,value))

  lista = list(similitud.values())

  maximo = max(lista)

  for key, value in similitud.items():

    if value == maximo:

      print(f"El mejor documento es {key} con un valor de {value}")

## Uso del Modelo

Para usar el modelo, lo primero que debemos hacer es recuperar todos los documentos que son parte de la colección sobre la que el modelo funciona.

d1,d2,d3 = recuperarDocumentos()

docs = [d1,d2,d3]

documentos = ponerId2(docs)

Posteriormente, procesamos estos documentos, para lo cual, utilizamos la función **encontrar\_ci,** pues con ella se van a procesar los documentos: extraer los términos, tokenizarlos, lematizarlos y todo lo que sea necesario. Así mismo, una vez limpios los documentos y su información, calcula los valores de CI para cada documento, por lo que, con ello, ya obtenemos los valores necesarios para la posterior consulta.

ci, documentosLimpios = encontrar\_ci(documentos, 1)

print("ci:")

Con ello, el siguiente paso es pedir la consulta al usuario, así como procesarla, para lo cual nos serviremos de la función **consulta\_usuario,** la cuál a su vez, ya se encarga de dos cosas: primero, preprocesar la consulta, es decir, procesar cada uno de sus términos incluyendo tokenización y lematización, entre otros; y, segundo, encontrar la similitud de esta consulta con cada uno de los documentos.

print("Ingrese la consulta que desea")

q1 = input()

consulta\_usuario(q1)

# Resultados

Para observar los resultados, se van a realizar ejecuciones por partes de cada una de las actividades anteriores al procesamiento de la consulta y, posteriormente, se ejecutarán 3 consultas distintas para observar los resultados.

## Obtención de documentos

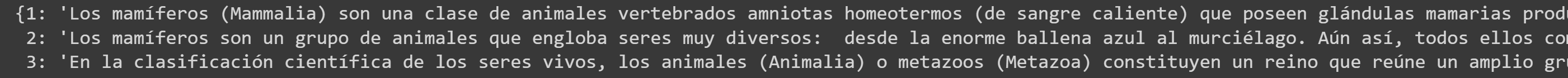
Código ejecutado:

docs = [d1,d2,d3]

documentos = ponerId2(docs)

documentos

Resultado:



*Fig. 1. Descripción de los documentos recuperados.*

## Creación de diccionario

Código ejecutado:

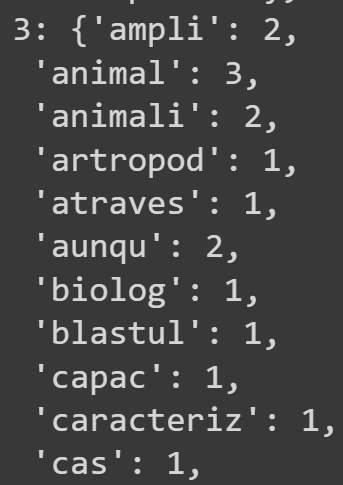
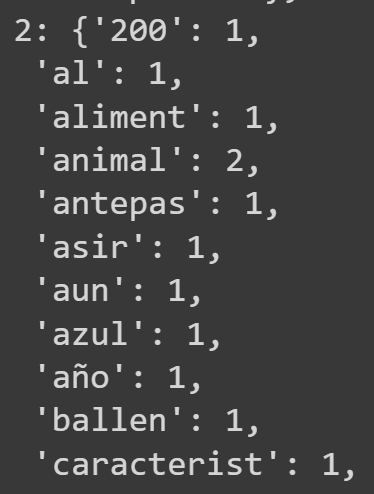
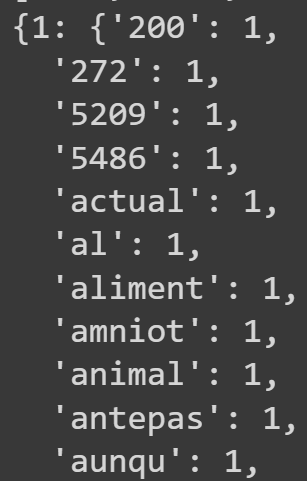
ci, documentosLimpios = encontrar\_ci(documentos, 1)

print("ci:")

for key , value in ci.items():

  print(key,": ",value)

Resultado:



*Fig. 2. Documentos procesados (términos y su frecuencia).*

## Cálculo de CI

Código ejecutado:

ci, documentosLimpios = encontrar\_ci(documentos, 1)

print("ci:")

for key , value in ci.items():

  print(key,": ",value)

Resultado:

Texto

Descripción generada automáticamente

*Fig. 3. Muestra de los términos con su respectivo valor de CI.*

## Consulta de prueba 1

Código ejecutado:

print("Ingrese la consulta que desea")

q1 = input()

consulta\_usuario(q1)

Resultado:

Texto

Descripción generada automáticamente

*Fig. 4. Resultado de la consulta “Clases seres”.*

## Consulta de prueba 2

Código ejecutado:

print("Ingrese la consulta que desea")

q1 = input()

consulta\_usuario(q1)

Resultado:

Texto

Descripción generada automáticamente

*Fig. 5. Resultado de la consulta “Clases seres”.*

## Consulta de prueba 3

Código ejecutado:

print("Ingrese la consulta que desea")

q1 = input()

consulta\_usuario(q1)

Resultado:

Texto

Descripción generada automáticamente

*Fig. 6. Resultado de la consulta “Clases seres”.*

# Bibliografía

|  |  |
| --- | --- |
| [1] | J. A. M. Comeche, «ucm.es,» [En línea]. Available: https://eprints.ucm.es/id/eprint/44030/1/aaaModelos%20derecuperaci%C3%B3ndela%20informaci%C3%B3nGrado10\_version\_final.pdf. [Último acceso: 07 03 2022]. |
| [2] | Spacy, «Spacy,» Spacy, 2022. [En línea]. Available: https://spacy.io/. [Último acceso: 03 03 2022]. |
| [3] | NLTK, «Natural Language Toolkit,» NLTK, 2022. [En línea]. Available: https://www.nltk.org/#. [Último acceso: 03 03 2022]. |
| [4] | python.org, «math,» python.org, 2022. [En línea]. Available: https://docs.python.org/es/3/library/math.html. [Último acceso: 03 03 2022]. |
| [5] | NLTK, «Source code for nltk.stem.snowball,» NLTK, 2022. [En línea]. Available: https://www.nltk.org/\_modules/nltk/stem/snowball.html. [Último acceso: 03 03 2022]. |