Documentación

Se llaman las librerías que serán utilizadas. Entre ellas dos librerías propias las cuales son:

- url_config: Esta libreria contiene una función a través de la cual se realiza el webscrapping de varios periodicos en línea.
- utilitools: Libreria de varios usos, se usa para configurar el arbol de rutas particularmente.

In [1]:
```
import requests
import re
import nltk
import random
import pandas as pd
from nltk import FreqDist
from nltk import word_tokenize
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.util import ngrams
from bs4 import BeautifulSoup
from urllib import request
from url_config import *
from utilitools import *
```

Creamos las siguientes funciones para el proceso de entrenamiento y el proceso de tokenización de las noticias.

In [61]:
```
# Esta función regresa los tokens del cuerpo de la noticia
def tokenizar_url(url):
    #Se envia solicitud a la página
    resultado=requests.get(url)

    #Se solicita el texo
    content=resultado.text
    soup=BeautifulSoup(content, 'lxml')

    if 'espectador' in url.lower(): box = bs_espectador(soup)
    elif 'semana' in url.lower(): box = bs_semana(soup)
    elif 'larepublica' in url.lower(): box = bs_larepublica(soup)
    elif 'portafolio' in url.lower(): box = bs_portafolio(soup)

    else: return 'no config'

    tokenizer=RegexpTokenizer('\w+')
    tokens=tokenizer.tokenize(box)
    tokens=[token.lower() for token in tokens]

    return tokens

# Esta función regresa un top de las palabras mas utilizadas dentro de las noticias, por
def top_freqdst(tokens,n=100):
    stops = stopwords.words('spanish')

    filtered_tokens = [token for token in tokens if token not in stops]

    f_dst = FreqDist(filtered_tokens)
    top_n = f_dst.most_common(n)

    return (top_n)
```

```
# Calcula la riqueza léxica del cuerpo de la noticia. Este calculo se realiza con el tot
def riq_lex(tokens):
    vocabulario = sorted(set(tokens))
    return len(vocabulario)/len(tokens)

# Retorna el top de n-gramas, es decir, las n agrupaciones de palabras que componen el c
def top_ngrams_freqdist(tokens,n=2,m=100):
    n_grams = ngrams(tokens, n)

    token_grams = [ ' '.join(elms) for elms in n_grams]

    f_dst = FreqDist(token_grams)
    top_m = f_dst.most_common(m)
    return (top_m)

# Esta función se utiliza para agrupar los parametros de entrenamiento del modelo de cla
def atributos(tokens):
    atrib = {}

    top_freqdist = top_freqdst(tokens,n=100)
    for x in top_freqdist:
        atrib['top_freqdist({})'.format(x[0])] = x[1]

    n_gram_freqdist = top_ngrams_freqdist(tokens,n=2,m=100)
    for x in n_gram_freqdist:
        atrib['top_ngram({})'.format(x[0])] = x[1]

    #atrib['riq_lex'] = riq_lex(tokens)

    return atrib
```

Se realiza la lectura del arbol de archivos del proyecto.

In [3]:
```
data = leer_paths('data')
recomendador = leer_paths('recomendador')
scripts = leer_paths('scripts')
```

Se procede a cargar como dataframe un excel que contiene las URL clasificadas previamente, las cuales serán utilizadas como entrenamiento. Adicionalmente, se realiza el proceso de tokenización del cuerpo de las noticias

In [6]:
```
df = pd.read_excel(os.path.join(data,'archivos_auxiliares\\url_noticias_clasificadas.xls
df = df[['url','categoria']]
df['tokens'] = df['url'].apply(tokenizar_url)
```

Se realiza el entrenamiento del modelo de clasificación Bayesiana. Inicialmente se cuenta con un total de 18 noticias, las cuales corresponden a 3 noticias por cada categoría. Para el proceso de entrenamiento, se tendrán en cuenta 10 de ellas y las 8 restantes se usan el testeo. Por otra parte, se realiza la medida del accuracy

In [57]:
```
fset = [(atributos(texto), clase) for texto, clase in zip(df['tokens'].values, df['categ
random.shuffle(fset)
train, test = fset[:10], fset[10:]
classifier = nltk.NaiveBayesClassifier.train(train)
print(nltk.classify.accuracy(classifier, test)) #se mide el accuracy
```

```
0.625
```

El accuracy mas alto obtenido con las condiciones dadas, es de 0.625, sin embargo, debemos tener en cuenta que actualmente el valor fluctua debido a que la muestra de entrenamiento es muy pequeña y esto

genera discrepancias en cada re entrenamiento. Esto se puede solucionar, teniendo una base de noticias clasificadas mas grande

Una vez entrenado el modelo, procedemos a cargar el archivo que contiene las noticias

In [58]:
```
# Esta función carga el archivo .csv que se encuentra en la ruta path con el nombre csv_
def leer_csv(path,csv_file):
    '''
    Inserta ruta de carpeta donde se encuentra el archivo y nombre de archivo para hacer
    '''
    for file in os.listdir(path):
        file_path = os.path.join(path,file)

        if csv_file in file_path:
            df = pd.read_csv(file_path)
    return df

df_noticias = leer_csv(data,'noticias.csv')
```

Se agrega la columna que contiene los tokens del cuerpo de la noticia

In [62]:
```
df_noticias['tokens'] = df_noticias['news_text_content'].apply(word_tokenize)
```

Se agrega una columna donde se clasifica la noticia, utilizando el clasificador entrenado

In [63]:
```
df_noticias['clase'] = df_noticias['tokens'].apply(lambda x: classifier.classify(atribut
```

Se exportan los datos

In [66]:
```
df_noticias.drop(['tokens','news_title','news_text_content'], axis=1,inplace=True)
df_noticias.to_csv(os.path.join(data,'output\\categorizacion.csv'),index=False)
```