UNIVERSIDAD SANTO TOMÁS

PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA

SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732

Acreditación Institucional Internacional

OTORGADA POR EL IAC CINDA ACUERDO 55 DEL 8 DE MAYO–VIGENCIA 5 AÑOS

ACREDITACIÓN INSTITUCIONAL DE ALTA CALIDAD MULTICAMPUS

Vigencia por seis años

QS STARS RATED FOR EXCELLENCE

ISO 9001 icontec SC4289-1

CERTIFIED IQNet MANAGEMENT SYSTEM

UNIVERSIDAD SANTO TOMÁS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
SECCIONAL TUNJA
VIGILADA MINEDUCACIÓN - SNIES 1732

Faculty: Systems engineer
Course: Deep Learning
Topic: sentiment analysis

Professor: Luis Fernando Castellanos Guarin
Email: Luis.castellanosg@usantoto.edu.co
Phone: 3214582098

CONTENIDO

Análisis de sentimientos usando:

- Dataset estático.
- Dataset de Twitter.

¡Siempre hacia lo alto!

Aplicar el aprendizaje automático para el análisis de sentimientos.

- Obtener base de datos de conocimientos.
- Limpiar y preparar datos textuales.
- Crear vectores de características a partir de documentos de texto.
- Entrenar un modelo de aprendizaje automático para clasificar criticas de películas (negativas y positivas).
- Trabajar con grandes conjunto de datos textuales.
- Inferir temas a partir de colecciones de documentos.

Análisis de sentimientos

Análisis de sentimientos = minería de opiniones:

Es una su sub-disciplina del campo de **PLN** (procesamiento del lenguaje natural) y se orienta en emociones u opiniones expresadas de autores respecto a un tema en particular.

Trabajaremos con un dataset (conjunto de datos) de criticas de cine procedentes de IMDb:https://www.imdb.com:

El dataset tiene 50.000 criticas de cine polarizadas como negativas (menos de 5 estrellas) y positivas (más de 5 estrellas), pasos:

- Obtener el conjunto de datos de criticas de cine.
- Preprocesar el conjunto de datos a un formato adecuado para aplicar machine learning.
- Definir que modelo de entrenamiento se usara.
- Transformar palabras a vectores.
- Relevancia de palabras mediante “frecuencia-termino”.
- Limpiar los datos textuales.
- Procesar componentes léxicos.
- Entrenar

Análisis de sentimientos

Para el ejercicio utilizaremos un cuaderno de **jupyter** en Google colaboratory https://colab.research.google.com/

Análisis de sentimientos: P1-obtener dataset

Base de conocimiento

http://ai.stanford.edu/~amaas/data/sentiment/

"conjunto de 25,000 críticas de películas altamente polares para capacitación y 25,000 para pruebas"

URL: http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz , Pesa 84 mb aprox

Paso 1: conociendo el dataset: descargar y descomprir el archivo…que tiene por dentro?

El propósito es que las computadoras hagan el trabajo rutinario por los humanos

¿descargamos y descomprimimos el archivo de forma manual o dejamos que Python por nosotros?

Análisis de sentimientos: P1-obtener dataset

Importando librerías

Librerías para descargar y descomprimir archivos de internet.

```
import os          #  trabajar sobre el sistema operativo
import sys         #  manipular archivos (cortar, copiar, borrar, crear)
import tarfile     #  Manipular archivos comprimidos (comprimir, descomprimir)
import time        #  calcular tiempo (en este caso tiempo de descarga de archivo)
```

Definimos dos variables:

- **Source:** de donde descargaremos el archivo.
- **Target:** como lo guardaremos en el disco

```
source = 'http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz‘

target = 'aclImdb_v1.tar.gz'
```

Análisis de sentimientos: P1-obtener dataset

Función de reporthook

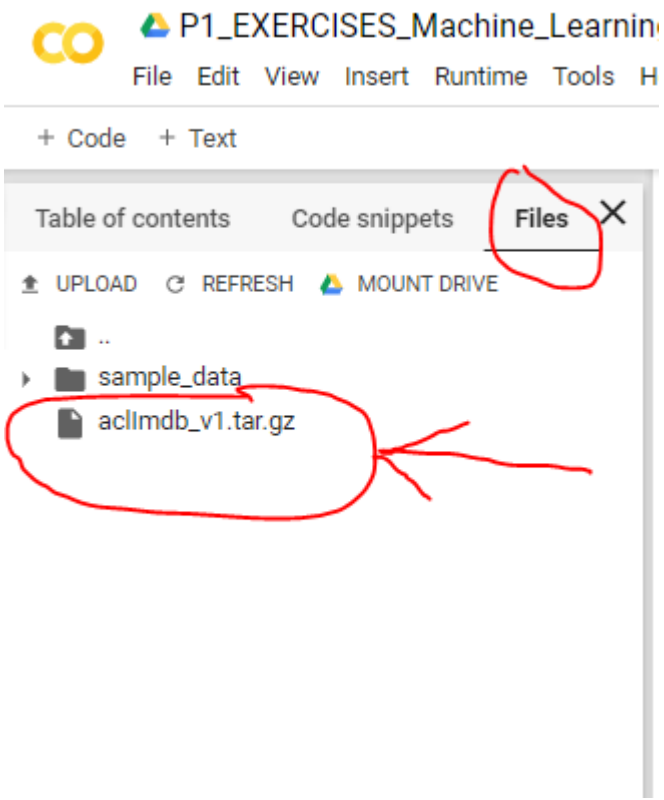
Descargar el archivo y mostrar proceso de descarga (usaremos mucho este código)

```
def reporthook(count, block_size, total_size):
    global start_time
    if count == 0:
        start_time = time.time()
        return
    duration = time.time() - start_time
    progress_size = int(count * block_size)
    speed = progress_size / (1024.**2 * duration)
    percent = count * block_size * 100. / total_size
    sys.stdout.write("\r%d%% | %d MB | %.2f MB/s | %d segundos transcurrido" %
                     (percent, progress_size / (1024.**2), speed, duration))
    sys.stdout.flush()


if not os.path.isdir('aclImdb') and not os.path.isfile('aclImdb_v1.tar.gz'):

    if (sys.version_info < (3, 0)):
        import urllib
        urllib.urlretrieve(source, target, reporthook)

    else:
        import urllib.request
        urllib.request.urlretrieve(source, target, reporthook)
```

Análisis de sentimientos: P1-obtener dataset

Dataset en .zip

Descargar el archivo y mostrar proceso de descarga (usaremos mucho este código)




```
100% | 80 MB | 7.15 MB/s | 11 segundos transcurridos
```

Nota:

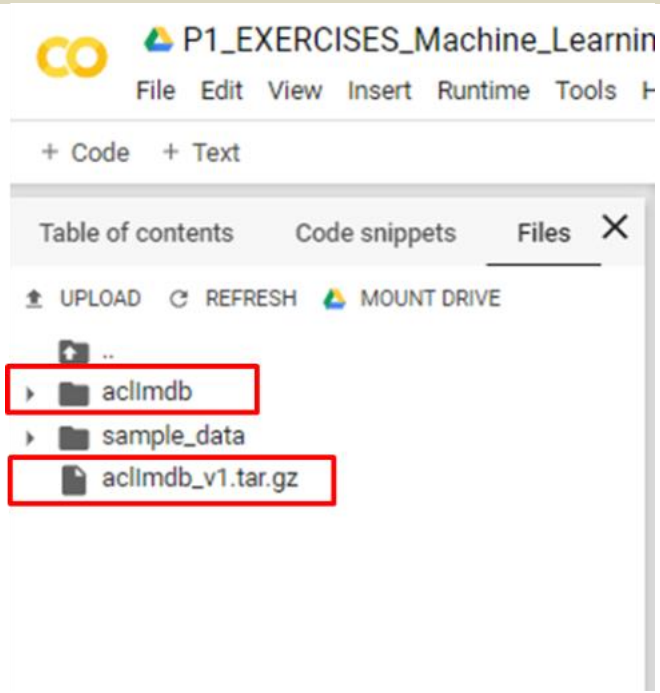
El listado de archivos es temporal (en el momento que se cierre la sesión en google colaboratory se borraran)…si son archivos de vital importancia y difícil de conseguir se recomienda guardar en drive (google).

Análisis de sentimientos: P1-obtener dataset

Descomprimir .zip

Una vez descargado el archivo procedemos a descomprimirlo

```
if not os.path.isdir('aclImdb'):

    with tarfile.open(target, 'r:gz') as tar:
        tar.extractall()
```



Análisis de sentimientos: P2-Preprocesar el DATASET

El DATASET “'***aclImdb***” está separado en archivos de texto independientes (un archivo por cada crítica, positiva o negativa) y necesitamos a convertir todos esos archivos en un único archivo CSV (un formato muy usado en procesos de machine learning), para este trabajo usaremos una librería llamada PANDA y buscaremos crear un objeto llamados **DATAFRAME** que denominaremos **DF**.

Análisis de sentimientos: P2-Preprocesar el DATASET

Importando librerías

```
pip install pyprind  # instalar librería para visualizar el progreso de ejecución una tarea en background
```

Importamos las librerías necesarias

```
import pyprind
import pandas as pd
import os
```

Cambiamos el "**basepath**" al directorio donde esta el conjunto de datos.

```
basepath = 'aclImdb'
```

Lo anterior se hace para que Python pueda tomar un **path** temporal por defecto sin afectar las variables de entorno del sistema operativo.

Análisis de sentimientos: P2-Preprocesar el DATASET

Convertir archivos en un dataset panda

Ejecutar proceso para convertir todo en un único archivo (recuerde los archivos son temporales). Este proceso en un computador sencillo (4 gigas de ram, procesador I5, SDD) puede durar de 10 minutos a 30 minutos…en google colaboratory cuando dura?

```
labels = {'pos': 1, 'neg': 0}
pbar = pyprind.ProgBar(50000)

df = pd.DataFrame()

for s in ('test', 'train'):
    for l in ('pos', 'neg'):
        path = os.path.join(basepath, s, l)
        for file in os.listdir(path):
            with open(os.path.join(path, file),
                      'r', encoding='utf-8') as infile:
                txt = infile.read()
            df = df.append([[txt, labels[l]]],
                           ignore_index=True)
            pbar.update()

df.columns = ['review', 'sentiment']
```

Se realizan 50000 iteraciones (documentos), mediante bucles **FOR** interactuamos con los subdirectorios **TRAIN** y **TEST** y leemos los archivos de texto de forma individual de los subdirectorios **POS** y **NEG** y añadimos al **DATAFRAME DF** de pandas una etiqueta de clase (1= positivo y 0= negativo).

Análisis de sentimientos: P2-Preprocesar el DATASET

Generar permutación

Como las etiquetas de clase en el DATASET están ordenadas (primeras 2500 positivas y las siguientes 25000 negativas), por lo tanto vamos a barajar el **DATAFRAME** mediante la función **PERMUTATION** del submodulo **np.random**.

```
import numpy as np

np.random.seed(0)
df = df.reindex(np.random.permutation(df.index))
```

Opcional: Guardamos el DATAFRAME en un único archivo CSV.

```
df.to_csv('movie_data.csv', index=False, encoding='utf-8')
```

¿Quedaría bien guardado el CSV?...imprimamos las tres primeras filas

```
import pandas as pd

df = pd.read_csv('movie_data.csv', encoding='utf-8')
df.head(3)
```

	review	sentiment
0	When you think of brilliant Australian comedy ...	1
1	Proud as i am of being a Dutchman, i'm truly s...	0
2	How sad there is no option to post a mark lowe...	0

Descargue el archivo ***movie_data.csv*** *y guárdelo en una carpeta en Google drive*

Análisis de sentimientos: Definir que modelo de entrenamiento se usaremos

Usando el bag-of-words model:

- Crear un vocabulario de componentes léxicos únicos (palabras únicas a partir de un conjunto de textos)

- Construir un vector de características a partir de cada documento que contiene el recuento de la frecuencia en que cada palabra aparece en un documento.

Análisis de sentimientos: Definir que modelo de entrenamiento se usaremos

Se deben convertir los datos categóricos (texto o palabras) en un formato numérico antes de pasarlo por un algoritmo de machine learning (ni texto, ni imágenes, ni audios….solo números)

Vectorizando texto plano (transformar palabras en vectores)

```
from sklearn.feature_extraction.text  import CountVectorizer

new_text  = ['probando  un texto  para pruebas  de texto]
vector= CountVectorizer(stop_words=None)
vector.fit(new_text)

print(vector.vocabulary_)
```

Análisis de sentimientos: P3-Vectorizando texto

fit_transform

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer()
docs = np.array([
        'The sun is shining',
        'The weather is sweet',
        'The sun is shining, the weather is sweet, and one and one is two'])
bag = count.fit_transform(docs)

print(count.vocabulary_)
```

```
{
  'and': 0,
  'is': 1,
  'one': 2,
  'shining': 3,
  'sun': 4,
  'sweet': 5,
  'the': 6 ,
  'two': 7,
  'weather': 8
}
```

El método ***fit_transform*** se construye el vocabulario del modelo y transformando en vectores los tres vectores:

- 'The sun is shining',
- 'The weather is sweet',
- 'The sun is shining, the weather is sweet, and one and one is two'

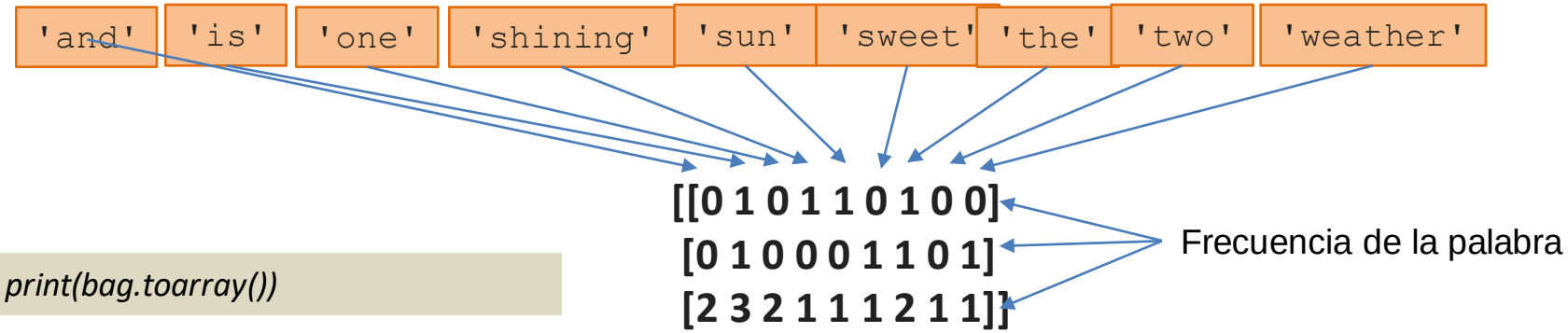
Análisis de sentimientos: P3-Vectorizando texto

fit_transform

Una vez creado el vocabulario esta almacenado en un diccionario de Python que mapea las palabras únicas en índices enteros.

print(count.vocabulary_)

```
{'and':0,'is':1,'one':2, 'shining':3, 'sun':4, 'sweet':5, 'the':6, 'two':7 ,'weather': 8}
```

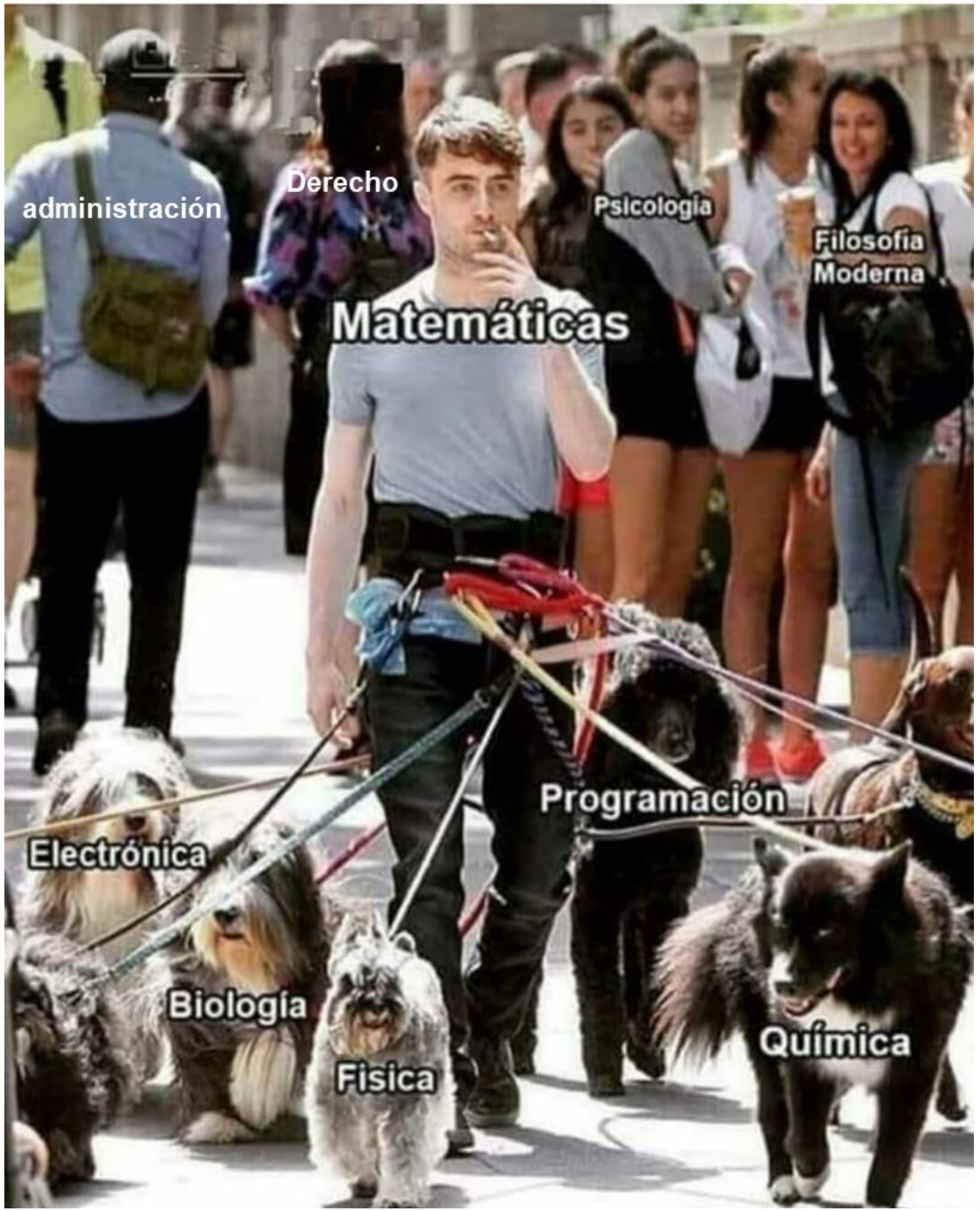


print(bag.toarray())

Cada posición de índice en los vectores de características que se muestran aquí corresponde a los valores enteros que se almacenan como elementos de diccionario en el vocabulario **CountVectorizer**.

Esos valores en los vectores de características también se **denominan frecuencias de términos sin procesar: tf (t, d): el número de veces que aparece un término t en cada posición de índice en los vectores de características que se muestran aquí corresponde a los valores enteros que se almacenan como diccionario elementos en el vocabulario CountVectorizer**

“Todo es matemáticas”



Análisis de sentimientos: P3-Vectorizando texto

relevancia de palabras

Cuando analizamos datos textuales, a menudo encontramos palabras que aparecen en múltiples documentos. Normalmente, estas palabras no contienen información útil o discriminatoria.

Usaremos la técnica frecuencia de termino-frecuencia inversa de documento (TF-IDF *frequency-inverse document frequency*) que se usa para bajar de peso a esas palabras

$$tf - idf\ (t,d) = tf\ (t,d) * idf\ (t,d)$$

Donde, tf(t,d) es la frecuencia de término y idf (t,d) es la frecuencia inversa de documento y se calcula así:

$$idf\ (t,d) = \log \frac{n_d}{1 + df(d,t)}$$

Donde n_d es el número total de documentos y df(d,t) es el número de documentos ***d*** que contiene el termino ***t***

Análisis de sentimientos: P3-Vectorizando texto

relevancia de palabras

Scikit-learn implementa otro transformador, el **TfidfTransformer**, que toma las frecuencias del término bruto de **CountVectorizer** como entrada y las transforma en **tf-idfs**:

```
from sklearn.feature_extraction.text import TfidfTransformer

tfidf = TfidfTransformer(use_idf=True,
                         norm='l2',
                         smooth_idf=True)
np.set_printoptions(precision=2)

print(tfidf.fit_transform(count.fit_transform(docs)).toarray())
```

'and'	'is'	'one'	'shining'	'sun'	'sweet'	'the'	'two'	'weather'
[0.	**0.43**	0.	0.56	0.56	0.	0.43	0.	0.]
[0.	**0.43**	0.	0.	0.	0.56	0.43	0.	0.56]
[0.5	**0.45**	0.5	0.19	0.19	0.19	0.3	0.25	0.19]

]

La palabra ***'is'*** tiene la frecuencia de término más grande en el tercer documento, siendo la palabra más frecuente. Sin embargo, después de transformar el mismo vector de características en tf-idfs, vemos que la palabra ***'is'*** ahora está asociada con un tf-idf relativamente pequeño (***0.45***) en el documento 3, ya que también está contenido en los documentos 1 y 2 y, por lo tanto**, es poco probable** que contener información útil y discriminatoria.

Análisis de sentimientos: P3-Vectorizando texto

Cargando el dataframe

Recordando lo de diapositivas anteriores que …"*el listado de archivos es temporal (en el momento que se cierre la sesión en google colaboratory se borraran)…si son archivos de vital importancia y difícil de conseguir se recomienda guardar en drive (google)"*.

Entonces vamos a usar el archivo "**movie_data.cvs**" que guardamos en drive/google

```
from google.colab import drive
drive.mount('/content/gdrive')

import pandas as pd
df = pd.DataFrame()
df = pd.read_csv('/content/gdrive/My Drive/USTA-201902/USTA-201902_7°_DEEP_LEARNING/Documentos/libros_jupyter/PLN_movie_data.csv', encoding='utf-8')
```

Una vez ingrese el código de verificación, Revisamos si quedo bien cargado, visualizando los tres primeros registros:

```
df.head(3)
```

Análisis de sentimientos: P4-limpiar datos textuales

Antes de aplicar cualquier modelo siempre debemos limpiar lo datos, en este caso eliminaremos los caracteres no deseados

```
df.loc[0, 'review'][-50:]
```

```
"HANK YOU ABC!! Finally I don't have to tape it! :)"
```

El texto contiene marcadores HTML, signos de puntuación y otros marcadores no alfabéticos.
Eliminaremos los marcadores HTML pero los signos **NO** porque pueden representar información semántica útil.

```
import re

# creamos una funcion llamada preprocessor
def preprocessor(text):
    text = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)',text)
    text = (re.sub('[\W]+', ' ', text.lower()) +' '.join(emoticons).replace('-', ''))
    return text
```

Análisis de sentimientos: P4-limpiar datos textuales

Antes de ejecutar la limpieza en todo el DATASET hagamos una prueba en un texto.

```
print(df.loc[0, 'review'][-50:])
print("---mismo  texto limpiado---")
preprocessor(df.loc[0, 'review'][-50:])
```

Aplicamos la función “preprocessor” a todas las criticas del DATAFRAME.

```
df['review'] = df['review'].apply(preprocessor)
```

Visualicemos un texto para ver si quedo bien.

```
print(df.loc[0, 'review'])
```

Análisis de sentimientos: P4-limpiar datos textuales

Una vez se tenga la seguridad que los datos están limpios, se procede a convertir el DATASET en palabras individuales, tomando como método los espacios entre las palabras.

```
def tokenizer(text):
    return text.split()

tokenizer('runners like running and thus they run')
```

```
['runners', 'like', 'running', 'and', 'thus', 'they', 'run']
```

Existe una técnica mejor llamada "**declinación de palabras**" o "**algoritmo de Porter"** que permite mapear las palabras pero tomando la raíz de la palabra

```
from nltk.stem.porter import PorterStemmer
porter = PorterStemmer()

def tokenizer_porter(text):
    return [porter.stem(word) for word in text.split()]
```

```
['runner', 'like', 'run', 'and', 'thu', 'they', 'run']
```

Análisis de sentimientos: P4-limpiar datos textuales

Cargando librerías

En cuanto a algoritmos de declinación el "**algoritmo porter**" es uno de los más antiguos y más sencillos, pero no el más eficiente.

Existen unos mejore como el snowball (porter2 o en ingles stemmer) y el Lancaster (paice/ husk stemmer) .

Para usarlos es necesario instalar el paquete **nltk:**

```
pip install nltk
```

Una vez instalado se debe descargar los componentes necesarios para trabajar con NLTK

```
import nltk
#nltk.download("all")
#nltk.download("popular")
nltk.download("stopwords")
```

Análisis de sentimientos: P4-limpiar datos textuales

Palabras en ingles.

Una vez descargado los paquetes necesarios podremos eliminar palabras vacías para el **idioma inglés**.

```
from nltk.corpus import stopwords

stop = stopwords.words('english')

[w for w in tokenizer_porter('a runner likes running and runs a lot')
if w not in stop]
```

```
['runner', 'like', 'run', 'run', 'lot']
```

Como vemos se ejecutan dos procesos:

- Tomar la raíz de las palabras.
- Eliminar palabras que no son relevantes para el idioma ingles (**“a”** y **“and”**)

Análisis de sentimientos: P5-entrenar un modelo de regresión logística

Definimos las librerías y paquetes que necesitaremos, en este caso usaremos una librería nueva llamada **GridSearchCV** para encontrar el conjunto de parámetros óptimo para nuestro modelo de **regresión logística** mediante una validación cruzada de 5 iteraciones

```
from sklearn.pipeline import Pipeline                    # permite implementar métodos de ajuste y transformación
from sklearn.linear_model import LogisticRegression      # modelo de regresión logistica
from sklearn.feature_extraction.text import TfidfVectorizer   #conversor de texto a vector
from sklearn.model_selection import GridSearchCV          #búsqueda de cuadrícula con validación cruzada (para usar con regresión logistica)
```

Dividiremos el DATAFRAME en 25000 documentos para entrenar y 25000 para probar.

```
# separamos los datos de entrenamiento y de pruebas
X_train = df.loc[:25000, 'review'].values
y_train = df.loc[:25000, 'sentiment'].values
X_test = df.loc[25000:, 'review'].values
y_test = df.loc[25000:, 'sentiment'].values
```

Análisis de sentimientos: P5-entrenar un modelo de regresión logística

Le definimos los parámetros globales para hacer la regresión logística, primero que convierta una colección de documentos en bruto en una matriz de características TF-IDF.

TfidfVectorizer *hace lo mismo que* ***CountVectorizer*** *y* ***Tfidftransformer***

"Recuerda la idea de Python es optimizar código"

```
from sklearn.feature_extraction.text import TfidfVectorizer
#propiedades de la conversor de texto a vectores
tfidf = TfidfVectorizer(strip_accents=None,
                        lowercase=False,
                        preprocessor=None)
```

Lo anterior nos permite tener una base de como queremos que realice la vectorización del texto:

- No Definir un preprocesador → **preprocessor= none** (no pues ya usamos uno atras)
- No Poner todas las letras en minúscula → **lowercase= fase**
- No Eliminar los acentos → **strip_accents= fase**

La Eliminación de los acentos se usa en idiomas como el español donde los acentos son muy frecuentes

Para La eliminación y normalización de caracteres durante el paso de preprocesamiento. Existen varias opciones:

- **'ascii'** es un método rápido que solo funciona en caracteres que tienen un mapeo ASCII directo.
- **'unicode'** es un método un poco más lento que funciona en cualquier personaje.
- **'None'**, Ninguno (predeterminado) no hace nada

Análisis de sentimientos: P5-entrenar un modelo de regresión logística

Diccionario

Creamos dos diccionarios**:**

- El primero: *use_idf=True, smoth_idf=true y norm='12'* , que se usarán para calcular los tf-idf

- El segundo: *use_idf=false, smoth_idf=false y norm=none,* para entrenar un modelo basado en frecuencias de términos sin procesar. Para usar la regresión logística debemos usar ***L1*** y ***L2*** mediante parámetros de penalización y adicionalmente comparamos diferentes fuerzas de regulación definiendo un rango de valores para el parámetro de regulación inversa ***C***.

```
#creación de dos diccionarios (1° para cálculos de TF-IDF, 2° para entrenar modelo con regresión logistica
param_grid = [{'vect__ngram_range': [(1, 1)],
               'vect__stop_words': [stop, None],
               'vect__tokenizer': [tokenizer, tokenizer_porter],
               'clf__penalty': ['l1', 'l2'],
               'clf__C': [1.0, 10.0, 100.0]},
              {'vect__ngram_range': [(1, 1)],
               'vect__stop_words': [stop, None],
               'vect__tokenizer': [tokenizer, tokenizer_porter],
               'vect__use_idf':[False],
               'vect__norm':[None],
               'clf__penalty': ['l1', 'l2'],
               'clf__C': [1.0, 10.0, 100.0]}, ]
```

Análisis de sentimientos: P5-entrenar un modelo de regresión logística

Pipeline

Crearemos una ***Pipeline,*** que es lo que encadena varios pasos juntos, una vez que se realiza la exploración inicial. Por ejemplo:

- Puede transformar características: normalizar números, ***convertir texto en vectores o completar datos faltantes (nuestro caso)***, son transformadores.
- predecir variables ajustando un algoritmo, como "*bosque aleatorio*" o "*máquina de vectores de soporte*", son estimadores

```
#mezclamos regresión lineal y vectores de textos en un solo proceso
lr_tfidf = Pipeline([('vect', tfidf),
                     ('clf', LogisticRegression(random_state=0))])
```

Creamos La búsqueda de cuadrícula (GridSearchCV), cuyo propósito es un enfoque para el ajuste de parámetros que construirá y evaluará metódicamente un modelo para cada combinación de parámetros de algoritmo especificados en una cuadrícula

```
gs_lr_tfidf = GridSearchCV(lr_tfidf, param_grid,
                           scoring='accuracy',
                           cv=5,
                           verbose=1,
                           n_jobs=-1)
```

El parámetro **n_Jobs=-1** con ello el interprete de Python utilizar todos los procesadores disponibles en el sistema

(no es recomendable usar sobre sistema operativo Windows, pues genera errores en la declinación de palabras)

Análisis de sentimientos: P5-entrenar un modelo de regresión logística

train

Una vez terminamos de definir todos los parámetros necesarios, procedemos a entrenar nuestros modelo

*(**tenga mucha paciencia, el siguiente proceso puede durar vaaaaaaaaaaarios minutos**)*

```
gs_lr_tfidf.fit(X_train, y_train)
```

Mínimo 30 minutos máximo hora y media (que en el mundo de la IA no es nada).

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
/usr/local/lib/python3.6/dist-packages/joblib/externals/loky/process_executor.py:706: UserWarning: A worker stopped while some jobs were given to the executor. Th
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 28.9min
```

Análisis de sentimientos: P5-entrenar un modelo de regresión logística

train

Si no tiene mucha paciencia, puedes desmejorar el proceso de entrenamiento

Si ha pasado más de 40 minutos y no ha finalizado el proceso de entrenamiento, lo mejor es detenerlo y cambiar el parámetro del diccionario (solo dejar el primero)

```
param_grid = [{'vect__ngram_range': [(1, 1)],
               'vect__stop_words': [stop, None],
               'vect__tokenizer': [tokenizer],
               'clf__penalty': ['l1', 'l2'],
               'clf__C': [1.0, 10.0]},
              ]
```

Lo anterior afectara en la precisión del modelo (estará por debajo del 90%) pero demorara menos tiempo en su proceso de creación.

Análisis de sentimientos: P5-entrenar un modelo de regresión logística

Ver resultados

Una vez terminada la búsqueda de cuadrícula, imprimimos el mejor conjunto de parámetros:

```
print('Mejores set de parametros: %s ' % gs_lr_tfidf.best_params_)
print('CV exactitud: %.3f' % gs_lr_tfidf.best_score_)
```

Una vez terminada la búsqueda de cuadrícula, imprimimos el mejor conjunto de parámetros:

```
clf = gs_lr_tfidf.best_estimator_
print('Test exactitud: %.3f' % clf.score(X_test, y_test))
```

Análisis de sentimientos: archivos independientes

P1. Instalando librerías

```
pip install pyprind
```

P2. Instalando librerías y descargando paquetes.

```
pip install nltk
```

P3. Descargando paquetes.

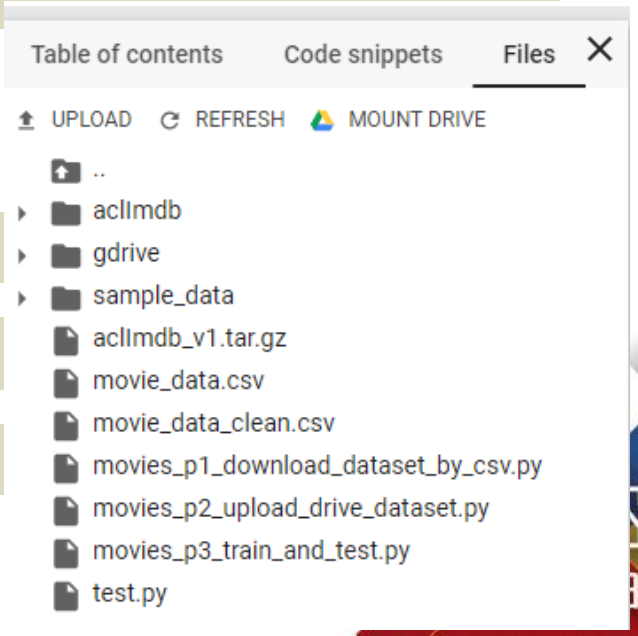
```
import nltk
nltk.download("stopwords")
```

P4. Cargar archivos Python con código completo.
P5. Ejecutar comando para cada uno de los archivos *.py

```
!python movies_p1_download_dataset_by_csv.py
```

```
!python movies_p2_upload_drive_dataset.py
```

```
!python movies_p3_train_and_test.py
```



This is

Precision!

Si muy bueno…pero solo hace eso?

…Creamos una IA que determina si una critica es buena o no y con una aceptabilidad del 89%... con eso creamos SKYNET…QUE SUSTOOOOOOOOO

¡Siempre hacia lo alto!

Sabemos que tenemos **50.000** criticas:

- 50% positivas
- 50% negativas

Pero no sabemos nada más!.

Dejemos que la inteligencia artificial nos categorice los documentos según lo más frecuente encontrado.

Modelado de temas con LDA - Latent Direchlet Allocation

Cargando dataset

CARGAMOS DRIVE donde tenemos el dataset

```
from google.colab import drive
drive.mount('/content/gdrive')

import re
import pandas as pd
import nltk


df = pd.DataFrame()
df = pd.read_csv('/content/gdrive/My Drive/USTA-201902/USTA-
201902_7°_DEEP_LEARNING/Documentos/libros_jupyter/PLN/DataSets/PLN_movie_data.csv', encoding='utf-8')
```

Modelado de temas con LDA - Latent Direchlet Allocation

Bolsa de palabras

usaremos el conocido CountVetorizer para crear la matriz de bolsa de palabras

```
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer(stop_words='english',max_df=.1,max_features=5000)
X = count.fit_transform(df['review'].values)
```

- Colocamos una frecuencia máxima en palabras de 10% (max_df=.1) para excluir palabras que aparecen con demasiada frecuencia en los documentos.
- Limitamos el número de palabras a 5.000 que aparecen con mayor frecuencia (max_features=5000)

Modelado de temas con LDA - Latent Direchlet Allocation

Estimador

Ajustamos el estimador

```
ffrom sklearn.decomposition import LatentDirichletAllocation

lda = LatentDirichletAllocation(n_component=10,
                                random_state=123,
                                learning_method='batch')
X_topics = lda.fit_transform(X)
```

Utilizamos la “bolsa de palabras” y le inferimos que sean 10 temas diferentes (este proceso demorara mínimo 5 minutos).
Al ajustar ***learning_method='batch'*** dejamos que LDA realice su estimación basándose en todos los datos de entrenamiento posible (hará lento el entrenamiento), si se desea hacerlo más rápido pueden usar ajustar ***learning_method='online'*** es un aprendizaje por mini-lotes

Modelado de temas con LDA - Latent Direchlet Allocation

Estimador

Revisemos como quedo la configuración

```
lda.components_.shape
```

```
(10, 5000)
```

Para analizar los resultados, vamos a imprimir las cinco palabras más importantes para cada uno de los 10 temas. Los temas están ordénanos de forma creciente, por lo tanto si queremos imprimir las 5 primeras debemos ordenar de forma inversa.

```
n_top_words = 5
feature_names = count.get_feature_names()

for topic_idx, topic in enumerate(lda.components_):
    print("Topic %d:" % (topic_idx + 1))
    print(" ".join([feature_names[i]
                    for i in topic.argsort()\
                        [:-n_top_words - 1:-1]]))
```

Modelado de temas con LDA - Latent Direchlet Allocation

Estimador

#	5 palabras más importantes	Categorias
1	worst minutes script awful stupid	Generalmente malas (no es una categoria de tema)
2	family mother father children girl	Sobre familias
3	war american dvd music history	Belicas
4	human audience cinema art sense	De autor
5	police guy car dead murder	Policiacas
6	horror house sex blood gore	Terror
7	role performance comedy actor performances	Comedia
8	series episode episodes tv season	Relacionadas con programas de tv
9	book version original effects special	Basadas en libros
10	action fight guy guys cool	Acción

Modelado de temas con LDA - Latent Direchlet Allocation

Estimador

Vamos a confirmar que las categorías tienen sentido y están basadas en las críticas, vamos a representar tres películas de la categoría <<películas de terror>> que son de la categoría 6.

```
terror = X_topics[:, 5].argsort()[::-1]

for iter_idx, movie_idx in enumerate(terror[:3]):
    print('\n Pelicula de terror #%d:' % (iter_idx + 1))
    print(df['review'][movie_idx][:300], '...')
```

Según el método LDA parece que si se han categorizado las críticas según el tema (para el ejemplo películas de terror).