notebook2-imdb-bert

November 21, 2024

1 Nombre: Rodolfo Jesús Cruz Rebollar

2 Matrícula: A01368326

3 Grupo: 101

4 Instalación e importación de paquetes y librerías

[1]:
```
# Instalación de keras versión 2.2.4 la primera vez que se ejecuta el notebook
!pip install keras==2.2.4

# Instalación de bert con tensorflow versión 1.0.1 en la primera ejecución del␣
 ↪notebook
!pip install -q bert-tensorflow==1.0.1
```

```
Collecting keras==2.2.4
  Downloading https://files.pythonhosted.org/packages/5e/10/aa32dad071ce52
b5502266b5c659451cfd6ffcbf14e6c8c4f16c0ff5aaab/Keras-2.2.4-py2.py3-none-any.whl
(312kB)
     |                  | 317kB 5.9MB/s eta 0:00:01
Requirement already satisfied: keras-preprocessing>=1.0.5 in
/opt/conda/lib/python3.6/site-packages (from keras==2.2.4) (1.1.0)
Requirement already satisfied: pyyaml in /opt/conda/lib/python3.6/site-packages
(from keras==2.2.4) (5.1.2)
Requirement already satisfied: scipy>=0.14 in /opt/conda/lib/python3.6/site-
packages (from keras==2.2.4) (1.2.1)
Requirement already satisfied: h5py in /opt/conda/lib/python3.6/site-packages
(from keras==2.2.4) (2.9.0)
Requirement already satisfied: six>=1.9.0 in /opt/conda/lib/python3.6/site-
packages (from keras==2.2.4) (1.12.0)
Requirement already satisfied: numpy>=1.9.1 in /opt/conda/lib/python3.6/site-
packages (from keras==2.2.4) (1.16.4)
Requirement already satisfied: keras-applications>=1.0.6 in
/opt/conda/lib/python3.6/site-packages (from keras==2.2.4) (1.0.8)
Installing collected packages: keras
  Found existing installation: Keras 2.3.0
    Uninstalling Keras-2.3.0:
```

```
      Successfully uninstalled Keras-2.3.0
Successfully installed keras-2.2.4
```

[2]:
```
# Archivo txt con los requerimientos de imagénes del entorno de Kaggle

!pip freeze > kaggle_image_requirements.txt
```

[3]:
```
# Importar librerías para trabajar con redes neuronales: tensorflow,␣
 ↪tensorflow_hub, bert.tonenization
# y tensorflow.keras

from bert.tokenization import FullTokenizer
import tensorflow as ten_flow
from tensorflow.keras import backend as tflow_keras
import tensorflow_hub as t_flow_hub

# Comenzar o inicializar sesión de trabajo de Tensorflow

working_sesion = ten_flow.Session()
```

[4]:
```
# Importación de otras librerías clave de Python para trabajar con modelos de
# deep learning y para manipulación y manejo eficiente de conjuntos de datos

import pandas as pd
import os
import numpy as np
from tqdm import tqdm
import re
```

5 Definir método de tokenización y funciones para eliminar signos de puntuación y palabras de paro (stop-words)

Antes de proceder, es de vital relevancia decidir la cantidad de muestras a extraer de cada clase. Adicionalmente, también es necesario elegir la máxima cantidad de tokens por cada correo electrónico, al igual que el tamaño máximo por token, esto se consigue configurando los hiperparámetros a continuación.

[67]:
```
# Valores de los parámetros para el proceso de tokenización y el modelo BERT

tok_max = 230 # cantidad máxima de tokens por cada documento

tam_max_token = 200 # Tamaño máximo por cada token

N_muestras = 1000 # cantidad de muestras a extraer por clase - 'spam', 'no spam'
```

Tokenización

[68]:
```
# Definir función tokenize para realizar el proceso de tokenización de las␣
 ↪frases y palabras que recibe como
# argumento de entrada un renglón o fila de información 'renglon'

def tokenize(renglon):
    # Si el renglón no tiene información, no generar ningún token
    if renglon is None or renglon == "":
        tokens = ""
    # De lo contrario, hacer lo siguiente
    else:
        # Generar los tokens correspondientes al renglón, dividiendo el renglón␣
 ↪unidades
        # semánticas de menor tamaño delimitadas por espacios en blanco y
        # posterior a ello, tomar como máximo la cantidad de elementos␣
 ↪especificada en tok_max (máximo de tokens)
        try:
            tokens = renglon.split(" ")[:tok_max]
        # En caso opuesto, no generar tokens
        except:
            tokens=""
    # Devolver los tokens generados
    return tokens
```

Utilización de expresiones regulares para la eliminación de caractéres innecesarios

Como próximo paso, se define una función para eliminar signos de puntuación junto con otros caractéres que no sean palabras o que no tengan significado semántico como tal (empleando expresiones regulares) de los emails con el apoyo de la librería `regex` de Python. Además en el msimo paso, se trunca la totalidad de los tokens a la longitud máximo definida en los hiperparámetros previos.

[69]:
```
# Función reg_expressions para eliminar caractéres carentes de significado␣
 ↪semático que recibe como
# argumento de entrada un renglón de información 'par'

def reg_expressions(par):
    # Crear listado vacío de tokens para el reglón correspondiente
    tk_list = []
    # Si el token no es vacío, convertirlo todo a minúsculas, remover␣
 ↪caractéres sin significado y
    # truncarlo a la longitud máxima establecida por token y finalmente␣
 ↪agregarlo a la lista de tokens
    # definida con anterioridad
    try:
        for tk in par:
            tk = tk.lower()
            tk = re.sub(r'[\W\d]', "", tk)
            tk = tk[:tam_max_token] # truncar el token en cuestión
            tk_list.append(tk)
```

```
    # Si el token no tiene texto, agregar un string vacío a la lista de tokens
    except:
        tk = ""
        tk_list.append(tk)
    # Retornar los tokens creados
    return tk_list
```

Eliminación de Stop-words (palabras de paro)

Ahora definiremos una función para quitar las palabras de paro (stopwords) - palabras cuya ocurrencia es muy frecuente en el languaje de forma que no proporcionan información de utilidad para las tareas clasificatorias, esto engloba palabras como "the" y "are", junto con la librería popular NLTK que porporciona una lista ampliamente empleada de stop words que a continuación utilizaremos.

[70]:
```
# Importar la librería nltk para trabajar con la lista amplia de stop words

import nltk

# Descargar listado de stop words de la librería nltk

nltk.download('stopwords')

# Importar función stopwords del módulo corpus perteneciente a la librería nltk

from nltk.corpus import stopwords

# Elegir el lenguaje Inglés para que los stop words estén en dicho lenguaje

stopwords = stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[71]:
```
# Crear una función para remover los stop words de un cierto renglón de␣
↪información

def stop_word_removal(fila):
    # Si un token no se encuentra en la lista de stopwords, no removerlo del␣
↪reglón de
    # información

    tkn = [tkn for tkn in fila if tkn not in stopwords]

    # Verificar si los tokens son nulos, por lo que en caso de serlo, se␣
↪devolverá un string vacío
    # como resultado de la función posteriormente

    tkn = filter(None, tkn)
```

```
    # Devolver los tokens generados ya sin los stop words

    return tkn
```

6 Descarga y Generación del IMDB Review Dataset

Descargar las reseñas IMDB a continuación.

[72]:
```
# Descargar el archivo correspondiente a las reseñas de IMDB del sitio web␣
 ↪especificado

!wget -q "http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"
!tar xzf aclImdb_v1.tar.gz
```

Preprocesamiento y barajeo (shuffle) de los datos.

[73]:
```
# Función para barajear o aleatorizar los datos

def unison_shuffle(datos, enc):
    p = np.random.permutation(len(enc))
    datos = datos[p]
    enc = np.asarray(enc)[p]
    return datos, enc

# Definir otra función para importar o cargar los datos de sentimientos en una
# determinada carpeta

def carga_datos(ruta):
    datos, sentimientos = [], []
    for carpeta, sentimiento in (('neg', 0), ('pos', 1)):
        carpeta = os.path.join(ruta, carpeta)
        for nombre in os.listdir(carpeta):
            with open(os.path.join(carpeta, nombre), 'r') as lector:
                  texto = lector.read()
            texto = tokenize(texto)
            texto = stop_word_removal(texto)
            texto = reg_expressions(texto)
            datos.append(texto)
            sentimientos.append(sentimiento)
    datos_np = np.array(datos)
    datos, sentimientos = unison_shuffle(datos_np, sentimientos)

    return datos, sentimientos

# Ruta donde están almacenados los datos de entrenamiento
```

```
ruta_train = os.path.join('aclImdb', 'train')

# Ruta donde se almacenan datos de testing o prueba

ruta_test = os.path.join('aclImdb', 'test')

# Cargar los datos crudos y el encabezado también en estado crudo

d_crudos, enc_crudo = carga_datos(ruta_train)

# Mostrar dimensiones de los datos crudos

print(d_crudos.shape)

# Desplegar longitud del encabezado crudo

print(len(enc_crudo))
```

```
(25000,)
25000
```

[74]:
```
# Submuestra requerida en base a la cantidad de muestras que se tengan

ind_aleatorios = np.random.choice(range(len(enc_crudo)), size=(N_muestras *␣
 ↪2,),replace=False)

tr_datos = d_crudos[ind_aleatorios]

enc = enc_crudo[ind_aleatorios]

print("DEBUG::data_train::")

print(tr_datos)
```

```
DEBUG::data_train::
[list(['tashan', '', 'title', 'explains', 'nature', 'moviebr', 'br', 'this',
'type', 'movies', 'actually', 'made', 'flop', 'what', 'shame', 'yash', 'raj',
'films', 'produces', 'movies', 'worthless', 'cgrade', 'movies', 'or', 'even',
'cgrade', 'movies', 'better', 'pleasing', 'story', 'tashan', 'the', 'much',
'hyped', 'overconfidently', 'promoted', 'tashan', 'poorly', 'bombed',
'boxoffice', 'certainly', 'deservedbr', 'br', 'in', 'view', 'worst', 'movie',
'ever', 'made', 'honourable', 'yash', 'raj', 'films', 'banner', 'how', 'come',
'handled', 'heavy', 'project', 'new', 'vijay', 'krishna', 'acharya', 'actual',
'sense', 'making', 'action', 'flick', 'he', 'tried', 'imitate', 'sanjay',
'gadhvis', 'ways', 'making', 'like', 'dhoom', 'suffered', 'last', 'the',
'action', 'scenes', 'like', 'comics', 'cartoon', 'movies', 'made', 'exhausting',
'audiencesbr', 'br', 'the', 'story', 'also', 'loses', 'meaning', 'substances',
'tenderly', 'win', 'audiences', 'hearts', 'in', 'scenes', 'anil', 'kapoor',
```

'reminds', 'southern', 'tamil', 'star', 'rajnikant', 'body', 'languages',
'wordly', 'expressions', 'i', 'fan', 'neither', 'saif', 'akshay', 'award',
'kareena', 'finally', 'gone', 'saifs', 'hand', 'instead', 'akshay', 'just',
'starting', 'point', 'i', 'expected', 'it', 'end', 'displeased', 'climax',
'truth', 'saif', 'main', 'behind', 'whole', 'adventure', 'akshay', 'joins',
'midst'])
 list(['what', 'mess', 'why', 'movie', 'made', 'this', 'movies', 'caliber',
'teaching', 'tools', 'make', 'movie', 'children', 'may', 'like', 'it', 'anyone',
'', 'may', 'disapprove', 'to', 'make', 'matters', 'worse', 'fact', 'great',
'talent', 'like', 'whoopi', 'goldberg', 'armin', 'mueller', 'stahl', 'entirely',
'wasted', 'film', 'unworthy', 'notice'])
 list(['flat', 'funniest', 'spoof', 'pretentious', 'art', 'house', 'films',
'ever', 'madebr', 'br', 'this', 'flick', 'exposes', 'clichés', 'some',
'excruciatingly', 'bad', 'downssyndrome', 'actors', 'terribly', 'heavy', 'self',
'important', 'dialog', 'scenes', 'supposed', 'shock', 'fall', 'flat', 'jarring',
'editing', 'pointless', 'plot', 'points', 'all', 'wrapped', 'kind', 'smirky',
'miasma', 'disrespect', 'audience', 'vague', 'psychdrivelbr', 'br', 'it',
'achieves', 'exactly', 'designed', 'to', 'a', 'hilarious', 'satire', 'tedious',
'movies', 'made', 'spoiled', 'teenage', 'trustfunders', 'show', 'parents',
'ask', 'theyve', 'last', 'two', 'years', 'after', 'what', 'is', 'it',
'received', 'cannes', 'award', 'presenter', 'werner', 'herzog', 'rumored',
'told', 'film', 'fact', 'spoof', 'part', 'films', 'he', 'supposedly', 'blew',
'info', 'to', 'day', 'refuses', 'discuss', 'incidentbr', 'br', 'anyway', 'see',
'laugh', 'classic', 'humor', 'many', 'years', 'come'])
 …
 list(['this', 'may', 'made', 'hell', 'it', 'probably', 'worst', 'film', 'ive',
'seen', 'years', 'the', 'best', 'thing', 'entire', 'dvd', 'would', 'case', 'im',
'surprised', 'people', 'took', 'time', 'make', 'something', 'rubbish', 'yet',
'spend', 'money', 'too', 'im', 'glad', 'rented', 'i', 'suppose', 'real', 'fans',
'film', 'would', 'probably', 'sadistic', 'gothic', 'care', 'without', 'taking',
'cgi', 'effects', 'matter', 'i', 'hope', 'alex', 'chandon', 'learnt', 'lesson',
'lighting', 'sfx', 'make', 'better', 'film', 'future', 'is', 'still', 'workbr',
'br', 'notes', 'buyers', 'extremely', 'disappointing', 'dont', 'buy', 'it'])
 list(['did', 'use', 'entire', 'budget', 'paying', 'porno', 'stars', 'whatbr',
'br', 'sound', 'effects', 'background', 'music', 'editing', 'general', 'bad',
'think', 'yearold', 'wannabe', 'made', 'filmbr', 'br', 'most', 'acting', 'good',
'considering', 'script', 'innocent', 'virgin', 'played', 'part', 'really',
'wellbr', 'br', 'the', 'mutants', 'look', 'really', 'cool', 'actually', 'could',
'really', 'cool', 'flick', 'right', 'brain', 'behind', 'wheel', 'but',
'unfortunately', 'involved', 'thats', 'casebr', 'br', 'turn', 'left', 'made',
'better', 'movie', 'guys', 'even', 'money', 'good', 'thing', 'i', 'rent',
'movie', 'myself'])
 list(['rich', 'ditzy', 'joan', 'winfield', 'a', 'woefully', 'miscast', 'bette',
'davis', 'engaged', 'married', 'stupid', 'egotistical', 'allen', 'brice',
'jack', 'carson', 'looking', 'lost', 'her', 'father', 'eugene', 'palette',
'determined', 'stop', 'marriage', 'kidnapped', 'pilot', 'steve', 'collins',
'james', 'cagney', 'seriously', 'they', 'crash', 'land', 'desert', 'hate',
'sigh', 'start', 'falling', 'lovebr', 'br', 'this', 'seems', 'getting', 'high',

```
'rating', 'reviewers', 'cagney', 'davis', 'it', 'they', 'brilliant', 'actors',
'known', 'dramas', 'not', 'comedy', 'movie', 'shows', 'why', 'the', 'script',
'horribletheres', 'one', 'genuine', 'laugh', 'entire', 'movie', 'the',
'running', 'joke', 'cagney', 'davis', 'falling', 'rump', 'first', 'cactus',
'this', 'done', 'three', 'times', 'only', 'considerable', 'talents', 'save',
'completely', 'humiliated', 'as', 'best', 'lousy', 'material', 'cagney',
'tries', 'best', 'lines', 'davis', 'screeches', 'every', 'line', 'full',
'force', 'work', 'carson', 'what', 'hell', 'look', 'face', 'throughout',
'entire', 'movie', 'probably', 'characters', 'emotions', 'change', 'seconds',
'only', 'palette', 'distinctive', 'voice', 'top', 'readings', 'manges',
'elicit', 'smiles', 'but', 'all', 'dull', 'laughlessa', 'real', 'chore', 'sit',
'through', 'this', 'gets', 'two', 'stars'])]
```

Mostrar sentimientos and sus respectivas frecuencias en el dataset, para garantizar que se encuentra balanceado entre las distintas clases.

[75]:
```
sents_unicos, frec_sents = np.unique(enc, return_counts=True)

print("Sentimentos y sus frecuencias:")

print(sents_unicos)

print(frec_sents)
```

```
Sentimentos y sus frecuencias:
[0 1]
[ 996 1004]
```

[76]:
```
# Función para transformar datos al formato adecuado, debido a la diferencia en␣
 ↪el formato requerido de los modelos de sklearn
# Se espera a un único string por cada email contra un listado de tokens para␣
 ↪los modelos de Scilit Learn antes explorados

def transformar_datos(d_crudos, enc):
    datos_conv, etiquetas = [], []
    for i in range(d_crudos.shape[0]):
        # combinar listado de tokens que representan a cada email en un único␣
 ↪string
        salida = ' '.join(d_crudos[i])
        datos_conv.append(salida)
        etiquetas.append(enc[i])
    datos_conv = np.array(datos_conv, dtype=object)[:, np.newaxis]

    return datos_conv, np.array(etiquetas)

tr_datos, enc = unison_shuffle(tr_datos, enc)

# split into independent 70% training and 30% testing sets
```

```
index = int(0.7 * tr_datos.shape[0])

# 70% of data for training

x_tr, y_tr = transformar_datos(tr_datos[:index], enc[:index])

# remaining 30% for testing

x_test, y_test = transformar_datos(tr_datos[index:], enc[index:])

print("Detalles de las listas x_train/y_train, para asegurar que estén en el␣
 ↪formato correcto:")

print(len(x_tr))

print(x_tr)

print(y_tr[:5])

print(y_tr.shape)
```

```
Detalles de las listas x_train/y_train, para asegurar que estén en el formato
correcto:
1400
[['spoilers extreme bashing lay aheadbr br when show first started i found
tolerable fun fairly oddparents kind cartoon kids adults liked it also high
ratings along spongebob but started fall following crap butch hartman team
shoved showbr br first off toilet humor funny you easily pull fast laugh little
kiddie burp thats pretty much audience would laugh cliché joke next kiddie jokes
lol see people underwear see people crossdressing lololol i cant stop laughing
gay bliss somebody help me but course show suck bad stereotypes did see team
portrayed australians they saw nothing kangarooloving boomerangthrowing
simpletons live hot desert but now is coup de grace why show truly sucks loudest
all overused jokes the show constantly pulls jokes the majority unfunny thinking
like greatest thing ever cosmo mostly one blame i hated kept mentioning super
toilet which also blend']
 ['woody allen lost ability write dialogue characters clearly distinguishable
other this case melinda melinda characters speak allens generic
pseudosophistication problems points view relatable anyone outside four block
radius allen lives they also share curious condition able afford multimillion
dollar manhattan apartments appear designed professional decorators regardless
financial situation livingbr br the character exists outside dull mindset will
ferrel obligatory woody allen surrogate although simply come merely woody allen
impression like kenneth branagh godawful celebrity ferrel lacks charm charisma
real woody playing part best moviesbr br the end result another string self
indulgent bores oncegreat filmmaker trading former reputation years']
 ['im surprised movie rated highly although i go typical grade scale  c perhaps
```

```
thats right movie typical thriller except boringly slow unrealistic not typical
thriller realistic one seemed trying to yet woman got rapped press charges want
cross examined court even though would putting man broke arm beat crap raped
away life also protecting lawyer feelings family random people even know there
similar problems movie would right kind moral take away movie moral questions
like whether right try killbeat kady anything illegal presented little one sided
since kady ended crazed bastard bent revenge sure lawyer justified protecting
family since waiting kady actually rape daughter could something legally would
bit absurd so ive waisted life']
 …
 ['the good earth great movie hear much anymore there lot big disasters events
also nonpassionate love story all happens little two hours short todays
standards the special effects costumes good time periodbr br i surprised luise
rainer received oscar limiting role she basically three emotions submissive
hungry heartbrokenbr br the performances asian asianamerican actors terrificbr
br ']
 ['never heard moviesaw dvdgreat movieperfect example movie took every cast
member make workno overhyped typical hollywood movie old overhyped actorsno
current quote a list actor could pulled performance moviebrought back memories
post vietnam war military experiencesit concentrated people sent fightas
portrayed characters fears emotions even volunteered servicethey regular people
toosome cut military lifei remember experienceto put mildly adapt military life
eitherbut ill never forget themshould stayed touchi highly recommend think
serving present day afganistanbasic training trip notice drill sergeants morning
people maybe need sensitivity training hahaha']
 ['i know sitcom not i agree one greatest television shows ever its great show
still airs and i love michelle its cute episodes baby talked sometimes said
something funny awbr br this show relate children teens and well families
struggle rough times try work family i know would ever turn opportunity watch
show someone br br i love episode i think name dd older girl accidentally stole
sweatshirt learned lesson stealing that great episode an example tv show shows
family working things familybr br i recommend show everyone']]
[0 0 0 0 0]
(1400,)
```

7 Construcción, entrenamiento y evaluación de modelo BERT

Primero se definen aquellas funciones de importancia crítica que definen diversos componentes del modelo BERT.

[77]:
```
class Ejemplo_Input(object):

    """Un único ejemplo de entrenamiento/prueba para clasificación de␣
 ↪secuencias simples."""

    def __init__(self, id_ejem, texto_A, texto_B = None, etiqueta = None):
```

```
        """Construir un objeto de clase Ejemplo_Input.
    Args:
      id_ejem: Id único para el ejemplo.
      texto_A: string. Texto sin tokenizar de la primer secuencia. Para tareas␣
 ↪de 1 secuencia,
      solamente se especifica esta secuencia.
      texto_B: string opcional. Texto sin tokenizar de la segunda secuencia.
        Solamente se especifica para tareas de doble secuencia.
      etiqueta: string opcional. La etiqueta del ejemplo. Esto debe ser␣
 ↪especificado
      para ejemplos de entrenamiento, pero no para ejemplos de prueba.
    """
        self.guid = id_ejem
        self.text_a = texto_A
        self.text_b = texto_B
        self.label = etiqueta


def generar_tokenizador_de_modulo_hub(ruta_bert):
    """Extraer archivo con vocabulario e información de casing del módulo Hub.
 ↪"""
    modulo_bert = t_flow_hub.Module(ruta_bert)
    inf_tokenizacion = modulo_bert(signature = "tokenization_info", as_dict =␣
 ↪True)
    archivo_vocab, hacer_minusculas = working_sesion.run(
        [inf_tokenizacion["vocab_file"], inf_tokenizacion["do_lower_case"]]
    )

    return FullTokenizer(vocab_file = archivo_vocab, do_lower_case =␣
 ↪hacer_minusculas)


def convert_single_example(tokenizador, ejemplo, max_tam_secuencia = 256):

    """Transforma un único `EjemploInput` en un único `InputFeatures`."""

    tks_A = tokenizador.tokenize(ejemplo.text_a)

    if len(tks_A) > max_tam_secuencia - 2:
        tks_A = tks_A[0 : (max_tam_secuencia - 2)]

    tks = []

    ids_de_segmentos = []

    tks.append("[CLS]")
```

```
    ids_de_segmentos.append(0)

    for token in tks_A:
        tks.append(token)
        ids_de_segmentos.append(0)

    tks.append("[SEP]")
    ids_de_segmentos.append(0)

    ids_entradas = tokenizador.convert_tokens_to_ids(tks)

    # La máscara tiene 1 para tokens verdaderos y 0 para los padding tokens. Solamente
    # los tokens verdaderos son tomados en cuenta

    mascara_entrada = [1] * len(ids_entradas)

    # Zero-pad hasta la longitud de la secuencia

    while len(ids_entradas) < max_tam_secuencia:
        ids_entradas.append(0)
        mascara_entrada.append(0)
        ids_de_segmentos.append(0)

    assert len(ids_entradas) == max_tam_secuencia
    assert len(mascara_entrada) == max_tam_secuencia
    assert len(ids_de_segmentos) == max_tam_secuencia

    return ids_entradas, mascara_entrada, ids_de_segmentos, ejemplo.label


def transforma_ejemplos_a_features(tokenizer, ejems, long_max_seq = 256):

    """Transforma un set of `InputExample`s a una lista de `InputFeatures`."""

    ids_inp, inp_masks, segment_ids, etiquetas = [], [], [], []

    for ej in tqdm(ejems, desc = "Converting examples to features"):
        id_inp, inp_mask, segment_id, etiqueta = convert_single_example(
            tokenizer, ej, long_max_seq
        )

        ids_inp.append(id_inp)

        inp_masks.append(inp_mask)

        segment_ids.append(segment_id)
```

```
        etiquetas.append(etiqueta)

    return (
        np.array(ids_inp),
        np.array(inp_masks),
        np.array(segment_ids),
        np.array(etiquetas).reshape(-1, 1),
    )

def convert_text_to_examples(textos, ets):

    """Crear InputEntradas"""

    InputEntradas = []

    for texto, et in zip(textos, ets):
        InputEntradas.append(
            Ejemplo_Input(id_ejem = None, texto_A = " ".join(texto), texto_B =␣
 ↪None, etiqueta = et)
        )
    return InputEntradas
```

Next, we define a custom tf hub BERT layer

[78]:
```
class BertLayer(ten_flow.keras.layers.Layer):
    def __init__(
        self,
        n_fine_tune_layers=10,
        pooling="mean",
        bert_path="https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1",
        **kwargs,
    ):
        self.n_fine_tune_layers = n_fine_tune_layers
        self.trainable = True
        self.output_size = 768
        self.pooling = pooling
        self.bert_path = bert_path
        if self.pooling not in ["first", "mean"]:
            raise NameError(
                f"Tipo de agrupación (pooling) indefinido (debe ser ya sea␣
 ↪first o mean, pero es {self.pooling}"
            )

        super(BertLayer, self).__init__(**kwargs)

    def build(self, shape_entrada):
```

```
        self.bert = t_flow_hub.Module(
            self.bert_path, trainable = self.trainable, name = f"{self.
↪name}_module"
        )

        # Eliminar capas no utilizadas

        vars_entrenables = self.bert.variables

        if self.pooling == "first":
            vars_entrenables = [variable for variable in vatrs_entrenables if␣
↪not "/cls/" in variable.name]
            capas_entrenables = ["pooler/dense"]

        elif self.pooling == "mean":
            vars_entrenables = [
                variable
                for variable in vars_entrenables
                if not "/cls/" in variable.name and not "/pooler/" in variable.
↪name
            ]
            capas_entrenables = []
        else:
            raise NameError(
                f"Tipo de agrupación (pooling) no definido (debe ser de tipo␣
↪first o mean, pero es {self.pooling}"
            )

        # Elegir la cantidad de capas a tunear

        for i in range(self.n_fine_tune_layers):
            capas_entrenables.append(f"encoder/layer_{str(11 - i)}")

        # Actualizar las variables entrenables para que sólo contegan las capas␣
↪especificadas

        vars_entrenables = [
            variable
            for variable in vars_entrenables
            if any([l in variable.name for l in capas_entrenables])
        ]

        # Agregar a los pesos entrenables

        for variable in vars_entrenables:
            self._trainable_weights.append(variable)
```

```
        for variable in self.bert.variables:
            if variable not in self._trainable_weights:
                self._non_trainable_weights.append(variable)

        super(BertLayer, self).build(shape_entrada)

    def call(self, entradas):
        entradas = [tflow_keras.cast(entrada, dtype="int32") for entrada in␣
↪entradas]
        input_ids, input_mask, segment_ids = entradas
        bert_inputs = dict(
            input_ids=input_ids, input_mask=input_mask, segment_ids=segment_ids
        )
        if self.pooling == "first":
            pooled = self.bert(inputs=bert_inputs, signature="tokens",␣
↪as_dict=True)[
                "pooled_output"
            ]
        elif self.pooling == "mean":
            result = self.bert(inputs=bert_inputs, signature="tokens",␣
↪as_dict=True)[
                "sequence_output"
            ]

            m_mask = lambda x, m: x * ten_flow.expand_dims(m, axis=-1)
            masked_reduce_mean = lambda x, m: ten_flow.reduce_sum(m_mask(x, m),␣
↪axis = 1) / (
                    ten_flow.reduce_sum(m, axis = 1, keepdims = True) + 1e-10)
            input_mask = ten_flow.cast(input_mask, ten_flow.float32)
            pooled = masked_reduce_mean(result, input_mask)
        else:
            raise NameError(f"Tipo de pooling o agrupación no definido (debe␣
↪ser de tipo mean o first, pero es {self.pooling}")

        return pooled

    def compute_output_shape(self, shape_entrada):
        return (shape_entrada[0], self.output_size)
```

Ahora se procede a utilizar las capas incrustradas (embedding) y personalizadas de TF hub BERT que incluye una función de alto nivel para definir el modelo completo o general. De forma más específica, se coloca una capa densa entrenable con una dimensión de salida de 256 encima del BERT embedding.

[79]:
```
# Función para la construcción del modelo completo

def build_model(sec_longitud_max):
```

```
    ent_id = ten_flow.keras.layers.Input(shape = (sec_longitud_max,), name =␣
 ↪"input_ids")

    ent_mask = ten_flow.keras.layers.Input(shape = (sec_longitud_max,), name =␣
 ↪"input_masks")

    ent_segment = ten_flow.keras.layers.Input(shape = (sec_longitud_max,), name␣
 ↪= "segment_ids")

    bert_ents = [ent_id, ent_mask, ent_segment]

    # Extraer solamente las features para bert sin tunearlas

    salida_bert = BertLayer(n_fine_tune_layers = 0)(bert_ents)

    # Entrenamiento de la capa densa clasificatoria encima de las features␣
 ↪extraídas

    dense = ten_flow.keras.layers.Dense(256, activation="relu")(salida_bert)

    pred = ten_flow.keras.layers.Dense(1, activation="sigmoid")(dense)

    modelo = ten_flow.keras.models.Model(inputs = bert_ents, outputs = pred)

    modelo.compile(loss = "binary_crossentropy", optimizer = "adam", metrics =␣
 ↪["accuracy"])

    modelo.summary()

    return modelo

# Definir función para llevar a cabo la correcta inicialización de las␣
 ↪variables implicadas

def initialize_vars(sesion):

    sesion.run(ten_flow.local_variables_initializer())

    sesion.run(ten_flow.global_variables_initializer())

    sesion.run(ten_flow.tables_initializer())

    tflow_keras.set_session(sesion)
```

[80]:
```
# Ruta al modelo bert de Tensorflow Hub

Ruta_modelo_BERT = "https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1"

# Crear una instancia del tokenizador

tokenizador = generar_tokenizador_de_modulo_hub(Ruta_modelo_BERT)

# Convertir datos al formato de InputExample

ejemplos_train = convert_text_to_examples(x_tr, y_tr)
ejemplos_test = convert_text_to_examples(x_test, y_test)

# Convertir a features

(ids_input_train,masks_input_train,ids_segment_train,etiquetas_train) = \
transforma_ejemplos_a_features(tokenizador, ejemplos_train, long_max_seq =␣
 ↪tok_max)
(ids_input_test,masks_input_test,ids_segment_test,etiquetas_test) = \
transforma_ejemplos_a_features(tokenizador, ejemplos_test, long_max_seq =␣
 ↪tok_max)

# Construcción del modelo BERT

BERT = build_model(tok_max)

# Instanciar variables involucradas

initialize_vars(working_sesion)

# Entrenamiento del modelo BERT

historia_BERT = BERT.fit([ids_input_train, masks_input_train,␣
 ↪ids_segment_train],etiquetas_train,
                    validation_data = ([ids_input_test, masks_input_test,␣
 ↪ids_segment_test], etiquetas_test),
                    epochs = 5, batch_size = 32)
```

```
Converting examples to features: 100%|      | 1400/1400 [00:02<00:00,
513.83it/s]
Converting examples to features: 100%|      | 600/600 [00:01<00:00,
504.50it/s]
```

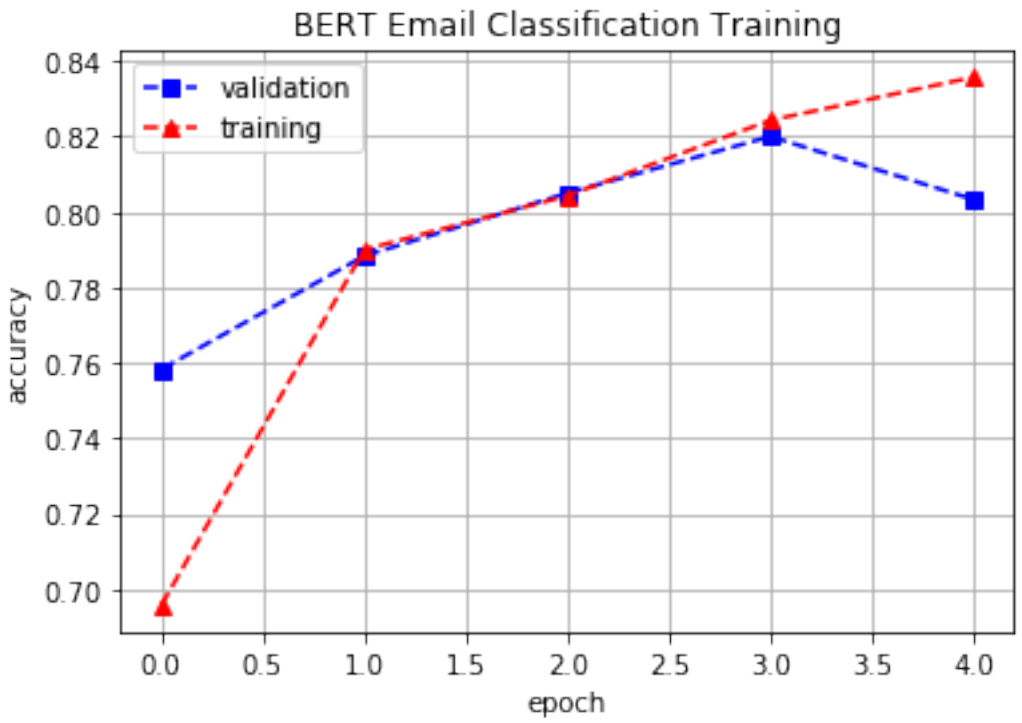
```
WARNING: Entity <bound method BertLayer.call of <__main__.BertLayer object at
0x7eacf6aa53c8>> could not be transformed and will be executed as-is. Please
report this to the AutgoGraph team. When filing the bug, set the verbosity to 10
(on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause:
converting <bound method BertLayer.call of <__main__.BertLayer object at
```

```
0x7eacf6aa53c8>>: AttributeError: module 'gast' has no attribute 'Num'
Model: "model_4"
__________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
==================
input_ids (InputLayer)          [(None, 230)]        0
__________________________________________________________________________________
__________________
input_masks (InputLayer)        [(None, 230)]        0
__________________________________________________________________________________
__________________
segment_ids (InputLayer)        [(None, 230)]        0
__________________________________________________________________________________
__________________
bert_layer_4 (BertLayer)        (None, 768)          110104890   input_ids[0][0]
input_masks[0][0]
segment_ids[0][0]
__________________________________________________________________________________
__________________
dense_8 (Dense)                 (None, 256)          196864
bert_layer_4[0][0]
__________________________________________________________________________________
__________________
dense_9 (Dense)                 (None, 1)            257         dense_8[0][0]
==================================================================================
==================
Total params: 110,302,011
Trainable params: 197,121
Non-trainable params: 110,104,890
__________________________________________________________________________________
__________________
Train on 1400 samples, validate on 600 samples
Epoch 1/5
1400/1400 [==============================] - 25s 18ms/sample - loss: 0.5780 -
acc: 0.6957 - val_loss: 0.5121 - val_acc: 0.7583
Epoch 2/5
1400/1400 [==============================] - 18s 13ms/sample - loss: 0.4520 -
acc: 0.7900 - val_loss: 0.4385 - val_acc: 0.7883
Epoch 3/5
1400/1400 [==============================] - 18s 13ms/sample - loss: 0.4349 -
acc: 0.8043 - val_loss: 0.4215 - val_acc: 0.8050
Epoch 4/5
1400/1400 [==============================] - 18s 13ms/sample - loss: 0.3899 -
acc: 0.8243 - val_loss: 0.4193 - val_acc: 0.8200
Epoch 5/5
1400/1400 [==============================] - 18s 13ms/sample - loss: 0.3727 -
```

```
acc: 0.8357 - val_loss: 0.4148 - val_acc: 0.8033
```

Visualizar la convergencia del modelo BERT

[81]:
```
import matplotlib.pyplot as mpltlib

history_table = pd.DataFrame(historia_BERT.history)

fig, ax = mpltlib.subplots()

mpltlib.plot(range(history_table.shape[0]), history_table['val_acc'],'bs--',␣
 ↪label = 'validation')

mpltlib.plot(range(history_table.shape[0]), history_table['acc'], 'r^--', label␣
 ↪= 'training')

mpltlib.xlabel('epoch')

mpltlib.ylabel('accuracy')

mpltlib.title('BERT Email Classification Training')

mpltlib.legend(loc = 'best')

mpltlib.grid()

mpltlib.show()

fig.savefig('BERTConvergence.eps', format='eps')

fig.savefig('BERTConvergence.pdf', format='pdf')

fig.savefig('BERTConvergence.png', format='png')

fig.savefig('BERTConvergence.svg', format='svg')
```



Make figures downloadable to local system in interactive mode

[82]:
```
from IPython.display import HTML
def create_download_link(title = "Download file", filename = "data.csv"):
    html = '<a href={filename}>{title}</a>'
    html = html.format(title=title,filename=filename)
    return HTML(html)

create_download_link(filename='BERTConvergence.svg')
```

[82]:
```
<IPython.core.display.HTML object>
```

[83]:
```
!ls
!rm -rf aclImdb
!rm aclImdb_v1.tar.gz
```

```
BERTConvergence.eps  aclImdb              aclImdb_v1.tar.gz.3
BERTConvergence.pdf  aclImdb_v1.tar.gz    kaggle_image_requirements.txt
BERTConvergence.png  aclImdb_v1.tar.gz.1
BERTConvergence.svg  aclImdb_v1.tar.gz.2
```

8 Implementación de Pipeline

El pipeline elaborado para la clasificación de los textos por medio del modelo BERT implementado con anterioridad, engloba las fases que se describen a continuación:

1. **Preparación del ambiente de desarrollo o ejecución:**
   - Instalar dependencias de importancia crítica, tales como la versión 1.14 de la librería TensorFlow, en conjunto con la librería `bert-tensorflow`
   - Configurar el entorno en el que se llevó a cabo la sesión de trabajo, además de importar librerías adicionales para la implementación del modelo BERT
2. **Preprocesamiento de los datos analizados:**
   - **Tokenización:** Dividir los textos originales en unidades llamadas **tokens**, mismos que cuentan con un umbral máximo de tamaño o longitud, además de que los textos poseen un límite máximo de tokens para generarse a partir de ellos.
   - **Limpieza de datos:** Consistió en remover aquellos caractéres que carecen de significado semático de los textos originales, además los tokens fueron truncados empleando expresiones regulares.
   - **Suprimir palabras de paro (stopwords):** Se empleó la lista de stopwords de la librería `NLTK` para remover aquellas palabras carentes de información útil para el texto.
   - **Convertir formato:** los textos originales fueron procesados en forma de listas de tokens o cadenas de texto (strings) de acuerdo al tipo de modelo utilizado.
3. **Cargar y preprocesar el conjunto de datos IMDB:**
   - Descargar y extraer el conjunto de datos.
   - Combinar y dividir los datos extraídos en datasets de entrenamiento y prueba (70% para entrenamiento y 30% para prueba).
4. **Definir el modelo:**
   - Programar una capa BERT con personalización mediante la utilización de la librería `tensorflow_hub`.
   - Generación del modelo BERT con una capa densa conformada por 256 unidades para extraer las características, además de que el modelo posee un output final de tipo binario y una función de activación de tipo sigmoide para las tareas de clasificación.
5. **Entrenar y evaluar el modelo BERT:**
   - Transformar los datos en características demandadas por el modelo BERT, tales como: división o segmentación, id de los tokens y máscaras.
   - Entrenar el modelo BERT utilizando el dataset de entrenamiento, incluyendo validación periódicamente con el dataset de prueba.
   - Visualizar de forma gráfica las métricas de convergencia del modelo BERT: pérdida (loss) y exactitud (accuracy).
6. **Obtención de resultados del modelo y guardarlos:**
   - Visualizar y exportar resultados del modelo en diversos formatos para realizar posteriormente reportes junto con mayores análisis.
   - Limpiar datos obtenidos durante el proceso intermedio (todo el proceso para llegar a los resultados finales) con la finalidad de optimizar el ambiente de trabajo.

Adicionalmente, es importante mencionar que el pipeline implementado a lo largo de todo el código previo, fue diseñado con el propósito de poder ejecutar de una manera altamente eficiente todo el proceso de implementación del modelo BERT que va desde el prepocesamiento de los datos iniciales hasta el cálculo de los pronósticos finales y la visualización gráfica de los resultados derivados del modelo.