KeyWord Research and Searh Engine Optimization

Wilson Andrés Pinzón

May 28, 2020

1 Introduccion

1.1 ¿Qué es el SEO?

El posicionamiento en buscadores u optimización de motores de búsqueda es el proceso de mejorar la visibilidad de un sitio web en los resultados orgánicos (los que no son pagados), de los diferentes buscadores. También es frecuente nombrarlo por su título inglés, SEO (Search Engine Optimization).

De la definición debemos quedarnos con dos ideas:

- Aumento de la visibilidad: el objetivo del SEO es hacer que lleguen más visitas a una página.
- Posicionamiento en buscadores: aunque Google es el rey de los motores de búsqueda (y siempre que hablamos de SEO nos referimos a Google), el significado de SEO se aplica a todos los buscadores (Bing, Yahoo, etc).

1.2 ¿Por qué es el SEO importante?

La razón más importante por la que es necesario el SEO es porque hace más útil tu página web tanto para los usuarios como para los motores de búsqueda. Aunque estos aún no pueden ver una página web como lo hace un humano. El SEO es necesario para ayudar a los motores de búsqueda a entender sobre qué trata cada página y si es o no útil para los usuarios.

Ademas presenta varias ventajas al implementar una estrategia SEO * Aumenta la visibilidad de una marca * Atrae tráfico cualificado * Genera oportunidades de ventas * Educar el mercado * Mejor rendimiento sobre la inversión

1.3 Keyword Research

Las keywords son términos utilizados en los buscadores para expresar la información que los usuarios quieren encontrar en Internet.

Lo que separa el contenido de los usuarios se llama palabras clave o Keywords. Una palabra clave es el término que digitan los usuarios en los buscadores para encontrar contenidos que solucionen sus inquietudes.

Por ejemplo, si una persona está planeando un viaje a Japón seguramente digitará “mejores destinos en Japón” o “los destinos más económicos de Japón” en la barra de búsqueda. Debemos realizar una investigación de palabras clave

El análisis y selección de keywords va a determinar casi todos los pasos que a seguir en la web.

Desde la elección del dominio (aunque no siempre), pasando por los contenidos y continuando con la estrategia de enlaces.

Por eso merece la pena realizar este esfuerzo al principio, porque no es de gusto descubrir que las palabras clave que se han escogido no aportan visitas, o atraen a los visitantes equivocados.

1.3.1 ¿Qué Keyword debería elegir?

Deberían ser esas keywords que mejor definen el servicio web , los objetivos o el modelo de negocio.

- Comenzar con las Keywords más genéricas.
- Depurar la lista de Keywords enfocado a el servicio web.
- Realizar una busqueda con palabras más especificas.
- Depurar la nueva lista enfocado a la web.

2 Codigo

2.1 Definimos las funciones Iniciales

Vamos a definir las funciones iniciales para el Keyword research de nuestro proyecto.

2.1.1 Funcion para definir el idioma y la palabra clave a utilizar.

Aqui vamos a elegir nuestra Keyword y además, el idioma y el país donde vamos a realizar la busqueda.

[1]:
```
def set_values():
    print("Language:")
    print("(1) Español")
    print("(2) English")
    language = input()
    if int(language) == 1:
        print ("Cuál es su Keyword?")
        Keyword= input()
        print(" \n Tu palabra es:", Keyword, "\n")
        LANG = "es"
        CONT="CO"
        return Keyword, LANG, CONT
    if int(language) == 2:
        print("What is your keyword?")
        Keyword= input()
        print("Your word is:", Keyword)
        LANG = "en"
        CONT="US"
        return Keyword, LANG, CONT
```

2.1.2 Funciones para escoger una nueva palabra y realizar todo el filtro.

En el caso que queramos escoger una nueva keyword, por medio de la función ya podremos aplicar todas las funciones

[2]:
```
def news():
    if CONT == 'US':
        print('Do you want to add another Keyword?')
        print('(1) Yes')
        print('(2) No')
        choice = input()
    if CONT == 'CO':
        print('¿Desea añadir otra Keyword?')
        print('(1) Si')
        print('(2) No')
        choice = input()
    return choice
```

[3]:
```
def new_keyword():
    a,b,c = set_values()
    erasecsvdata()
    downdata(a, b, c)
    data = reading_data(a)
    data = clean_zeros(data)
    keywording = Keymarcas(data.Keyword)
    print('{} Keywords'.format(len(keywording)))
    getalldata(keywording,a)
    dates= rec_data(a)
    data = pd.concat([ keywords_data, dates])
    Key = keys(data)
    Key = textkey(Key)
    return Key,data
```

2.1.3 Función para eliminar datos con Keywords anteriores o innecesarios.

En el caso de tener algun csv repetido, o con alguna palabra que no vayamos a utilizar podemos eliminarlo

[4]:
```
from pathlib import Path
path = Path.cwd()
import shutil, os, sys

def erasecsvdata():
    csv_files=list(filter(lambda x: '.csv' in x, os.listdir(path)))
    if CONT == 'CO':
        print ('Estos son los CSV en tu carpeta')
        for i in range(len(csv_files)):
            print ('({}) -> {}'.format(i+1, csv_files[i]))
```

```
        print ('Escoge los numeros de los CSV que quieres eliminar o 0 para␣
↪pasar (ejemplo 1,3)')
        numero = input()
    if CONT == 'US':
        print('This are the CSV on your folder')
        for i in range(len(csv_files)):
            print('({} -> {})'.format(i+1, csv_files[i]))
        print('Choose the numbers of the CSV that you want to delete or choose 0␣
↪to pass (example 1,3)')
        numero = input()
    A = numero.split(',')
    A = [int(integer) for integer in A]
    for j in range(len(csv_files)):
        #print(csv_files[j])
        if j+1 in A:
            os.remove(csv_files[j])
    print('Los archivos restantes son:')
    print (list(filter(lambda x: '.csv' in x, os.listdir(path))))
```

2.2 Funciones para descarga de datos

Luego de definir nuestra Keyword objetivo, creamos las funciones para realizar la busqueda.

2.2.1 Función para descargar los datos de la Keyword Principal.

[5]:
```
from selenium import webdriver
from selenium.webdriver import Chrome
from selenium.webdriver.support.ui import Select
from selenium.webdriver.common.keys import Keys

import time


def downdata(Keyword,LANG, CONT):
    if os.path.exists( Keyword +'.csv') == False:
        options = webdriver.ChromeOptions()
        prefs = {"download.default_directory": str(path)}
        options.add_experimental_option("prefs", prefs)
        browser = webdriver.Chrome(executable_path='chromedriver', options =␣
↪options)
        browser.get('https://cocolyze.com/en/google-keyword-planner-tool#null')

        username = browser.find_element_by_id('keyword')
        username.send_keys(Keyword)
```

```
        select_Country = Select(browser.find_element_by_id('country'))
        select_Country.select_by_value(CONT)

        select_lang = Select(browser.find_element_by_id("lang"))
        select_lang.select_by_value(LANG)

        button = browser.find_element_by_id('submitBtn')
        button.click()


        time.sleep(15)

        ids = browser.find_elements_by_xpath("//*[@href]")
        for ii in ids:
            #print (ii.text)
            if ii.text == 'CSV':
                download = ii
        #print (download.text)
        download.click()


        time.sleep(5)
        browser.close()
        os.rename('suggestion.csv', Keyword+'.csv')
```

2.2.2 Función para descargar los datos para todos los keywords de una lista.

[6]:
```
def getalldata(Keywording,Keyword):
    dire = "Data_"+ Keyword
    try:
    # Create target Directory
        os.mkdir(dire)
        print("Directory Created")
    except FileExistsError:
        print("Directory already exists")
    options = webdriver.ChromeOptions()
    prefs = {"download.default_directory": str(path)}
    options.add_experimental_option("prefs", prefs)
    browser = webdriver.Chrome(executable_path='chromedriver', options = options)
    browser.get('https://cocolyze.com/en/google-keyword-planner-tool#null')
    for i in range(len(Keywording)):
        if os.path.exists('Data_'+ Keyword +'/'+ Keywording[i]+str(i+1) +'.csv'):
            continue
        elif os.path.exists('Data_'+ Keyword+'/'+ Keywording[i] +'.csv')== False␣
↪:
            time.sleep(1)
```

```
            username = browser.find_element_by_id('keyword')
            username.clear()
            username.send_keys(Keywording[i])

            select_Country = Select(browser.find_element_by_id('country'))
            select_Country.select_by_value(CONT)
            select_lang = Select(browser.find_element_by_id("lang"))
            select_lang.select_by_value(LANG)
            button = browser.find_element_by_id('submitBtn')
            button.click()
            time.sleep(15)
            ids = browser.find_elements_by_xpath("//*[@href]")
            for ii in ids:
                #print (ii.text)
                if ii.text == 'CSV':
                    download = ii
            #print (download.text)
            download.click()

            time.sleep(3)

            os.rename('suggestion.csv',  Keywording[i]+str(i+1)+'.csv')

            shutil.move(Keywording[i]+str(i+1)+'.csv', str(path)+"\Data_"+
↪Keyword)
            time.sleep(2)
    print("All data downloaded")
    browser.close()
```

2.3 Entramos los valores y Keyword para nuestro SEO

[7]:
```
Keyword, LANG, CONT = set_values()
```

```
Language:
(1) Español
(2) English
2
What is your keyword?
Pants
Your word is: Pants
```

2.3.1 Borramos archivos innecesarios de alguna keyword erronea.

[8]:
```
erasecsvdata()
```

```
This are the CSV on your folder
(1 -> all_data_copy.csv)
```

```
(2 -> all_data_Pants.csv)
(3 -> all_data_sweatshirt.csv)
(4 -> nikeid sweatshirt1.csv)
(5 -> Pants.csv)
(6 -> sweatshirt.csv)
Choose the numbers of the CSV that you want to delete or choose 0 to pass
(example 1,3)
1,2,3,4
Los archivos restantes son:
['Pants.csv', 'sweatshirt.csv']
```

2.3.2 Descargamos los datos.

[9]:
```
downdata(Keyword,LANG, CONT)
```

2.4 Lectura y limpieza de datos

Ahora vamos a leer y limpiar el dataset descargado.

2.4.1 Leemos el dataset bajo las columnas que necesitamos.

[10]:
```
import pandas as pd
def reading_data(Keyword):
    col_list = ["Keyword","Search Volume","CPC"]
    keyword_Data=pd.read_csv(Keyword+'.csv', sep =";",usecols=col_list)
    print(keyword_Data.head())
    return keyword_Data
keyword_Data = reading_data(Keyword)
```

```
                  Keyword  Search Volume   CPC
0           carolina pants         823000  1.11
1  sponge bob square pants         246000  0.94
2              sweat pants         165000  1.06
3               pants yoga         165000  1.27
4               yoga pants         165000  1.83
```

2.4.2 Revisamos si existen datos vacios y eliminamos las filas con datos iguales a cero.

Quitamos todos los datos con algún valor a cero ya que esto significa que no hay informacion acerca del volumen de busqueda o del costo por clic para la palabra correspondiente. Además, en caso que eliminemos más del 40% de la informacion, lanzamos una advertencia ya que es una perdida muy grande de información.

[11]:
```
def clean_zeros(keyword_Data):
    col_names= keyword_Data.columns.tolist()
    for column in col_names:
```

```
        print("Null Data in {} = {}".format(column,keyword_Data[column].isnull().
↪sum()))
        print("\n")
    Total = len(keyword_Data)
    for column in col_names[1:-1]:
         fixed = keyword_Data.loc[keyword_Data[column] > 0]

    Ft= len(fixed)
    rang = (Ft/Total)*100

    if CONT == 'CO':
        if rang < 60:
            print ('Se presenta más del 40% de datos sucios, se recomienda usar␣
↪otra palabra')
        else :
            print('Tienes un buen dataset, continua')
    if CONT == 'US':
        if rang < 60:
            print('The Data has 40% of Null data, you should use another␣
↪Keyword')
        else:
            print('Great Dataset! Continue')

    return fixed



keyword_Data = clean_zeros(keyword_Data)
```

```
Null Data in Keyword = 0


Null Data in Search Volume = 0


Null Data in CPC = 0


Great Dataset! Continue
```

[12]:
```
keyword_Data.head(30)
```

[12]:

	Keyword	Search Volume	CPC
0	carolina pants	823000	1.11
1	sponge bob square pants	246000	0.94
2	sweat pants	165000	1.06
3	pants yoga	165000	1.27

4	yoga pants	165000	1.83
5	pants	135000	1.04
6	nike sweat pants	135000	0.49
7	men's jeans	135000	2.12
8	shirt	135000	0.86
9	tops	135000	0.97
10	pants cargo	110000	0.99
11	cargo pants	110000	0.97
12	khaki pants	90500	1.54
13	the sisterhood of the traveling pants	90500	0.22
14	pants dickies	74000	0.87
15	dickie pants	74000	0.77
16	khaki	74000	1.26
17	women's suit pants	60500	0.80
18	pants suit womens	60500	0.93
19	sweater pants womens	60500	1.12
20	cargo pants womens	60500	0.92
21	plaid pants	60500	0.74
22	women's cargo pants	60500	0.85
23	women's pants suits	60500	0.96
24	men's sweat pants	60500	2.04
25	palazzo pants	60500	0.87
26	women cargo pants	60500	0.97
27	sisterhood of the travelling pants	60500	0.12
28	suit pants women's	60500	0.96
29	cargo pants for womens	60500	0.95

2.5 Hacemos un filtro sobre las marcas que queremos analizar.

Queremos filtrar nuestro dataset respecto a las palabras que estan relacionadas con los productos o marcas que queremos "ofrecer".

[13]:
```
Marcas=['Nike', 'Adidas','Puma','Nautica','Levis', 'Under Armour', 'Zara',␣
↪'Diadora', 'Carolina']
```

2.6 FUZZY WUZZY

FuzzyWuzzy es una biblioteca de Python que se utiliza para la coincidencia de cadenas de texto. La coincidencia de cadenas difusa es el proceso de encontrar cadenas que coinciden con un patrón dado. Básicamente utiliza la distancia de **Levenshtein** para calcular las diferencias entre secuencias.

2.6.1 Distancia de Levenshtein

La distancia de Levenshtein es una métrica para medir la diferencia entre dos secuencias. Informalmente, la distancia de Levenshtein entre dos palabras es el número mínimo de ediciones de

un solo carácter (es decir, inserciones, eliminaciones o sustituciones) requeridas para cambiar una palabra en la otra.

[14]:
```
from fuzzywuzzy import process, fuzz
```

```
C:\Users\Pinnzon\anaconda3\lib\site-packages\fuzzywuzzy\fuzz.py:11: UserWarning:
Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove
this warning
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-
Levenshtein to remove this warning')
```

[15]:
```
def get_matches(query, choices, limit=30):
    results = process.extract(query, choices, limit = limit, scorer = fuzz.
 ↪partial_ratio)
    return results
```

2.6.2 Hacemos un filtro sobre nuestro conjunto de palabras con aquellas que tengan mas de un 90% de coincidencia con nuestras marcas.

[16]:
```
def Keymarcas(Keyword):
    keywords=[]
    for marca in Marcas:
        data = get_matches(marca,Keyword)
        for i in range(len(data)):
            if data[i][1] > 90:
                keywords.append(data[i][0])
    for i in range(len(keywords)):
        keywords[i]=keywords[i].replace('.','')
    return keywords
```

[17]:
```
keywords = Keymarcas(keyword_Data.Keyword)
```

[18]:
```
keywords
```

[18]:
```
['nike sweat pants',
 'nike womens sweat pants',
 'nike sweat pants for men',
 "nike men's sweat pants",
 "men's nike pants",
 'nike pants mens',
 'nike pants for men',
 'nike mens pants',
 'track pants nike',
 'nike track pants',
 'nike fleece tech pants',
 "nike women's pants",
 "women's nike pants",
```

```
"nike pants women's",
'nike gold pants',
'nikelab womens pants',
'nike tech fleece pants',
'nike pants tech fleece',
'adidas pants',
'adidas tracksuit pants',
'tracksuit pants adidas',
'track pants adidas',
'adidas pants womens',
'adidas women pants',
'adidas pants for men',
"women's adidas pants",
"adidas men's pants",
'men adidas pants',
'adidas tiro 17 pants',
'adidas pants red',
'adidas pants soccer',
'adidas pants tiro 17',
'adidas soccer pants',
'adidas red pants',
'under armor pants',
'underarmour sweat pants',
'carolina pants']
```

2.6.3 Para cada palabra encontrada, volvemos a realizar un Keyword Research

[19]:
```
getalldata(keywords ,Keyword)
```

```
Directory already exists
All data downloaded
```

Leemos y concatenamos todas las palabras encontradas por cada Keyword de nuestra lista filtrada

[20]:
```
def rec_data(Keyword):
    path_2 = 'Data_'+Keyword
    files = [file for file in os.listdir(path_2) if not file.startswith('.')] #␣
 ↪Ignore hidden files
    col_list = ["Keyword","Search Volume","CPC"]
    keywords_data = pd.DataFrame()

    for file in files:
        current_data = pd.read_csv(path_2+"/"+file, sep =";", usecols = col_list)
        keywords_data = pd.concat([keywords_data, current_data])
    for  i in range(len(keywords)):
```

```
        keywords_data= pd.concat([keywords_data,keyword_Data.
 ↪loc[keyword_Data['Keyword']==keywords[2]]])

    keywords_data.to_csv("all_data_"+Keyword+".csv", index=False)
    return keywords_data
keywords_data = rec_data(Keyword)
```

[21]:
```
keywords_data.head()
```

[21]:
```
                  Keyword  Search Volume   CPC
0      adidas pants men's          18100  1.46
1      adidas men's pants          18100  1.46
2      men's adidas pants          18100  1.46
3      adidas joggers men          18100  1.84
4  adidas joggers women's          18100  1.34
```

Eliminamos todas las palabras que tengan un volumen de busqueda menor al promedio y aquellas que esten repetidas

[22]:
```
def keys(keywords_data):
    keywords_data = keywords_data.sort_values(by='Search Volume',␣
 ↪ascending=False)
    keywords_data= keywords_data.reset_index(drop=True)

    keywords_data= keywords_data.loc[keywords_data['Search Volume']>␣
 ↪keywords_data['Search Volume'].mean()]

    Key= keywords_data['Keyword'].values.tolist()
    Key = list(dict.fromkeys(Key))
    return Key
Key = keys(keywords_data)
```

Extraemos las palabras como texto y eliminamos cualquier mayuscula para un menor procesamiento de datos

[23]:
```
def textkey(Key):
    Key = [text.lower() for text in Key]
    return Key
Key = textkey(Key)
Key[:30]
```

[23]:
```
['nike',
 'nfl schedule',
 'adidas',
 'nfl news',
 'under armour',
 'nike shoes',
```

```
 'carolina pants',
 'patriots schedule',
 'panthers',
 'nike outlet',
 'adidas shoes',
 'nfl shop',
 'saints schedule',
 'nike shoes for men',
 'under armour outlet',
 'nike backpacks',
 'sweatpants',
 'nike air max womens',
 'adidas womens shoes',
 'nike sweatpants',
 'under armour shoes',
 'nike sweat pants',
 'nike shorts',
 'adidas shoes for men',
 'adidas outlet',
 'joggers for men',
 'nike windbreaker',
 "men's joggers",
 'nike women',
 'nike headbands']
```

2.7 Insertar otra Keyword

En el caso que queramos combinar nuestra palabra inicial y queremos añadir más relacionadas a nuestro producto aqui lo podremos realizar

[24]:
```
while True:
    choice = news()
    if int(choice) == 1:
        Key,data = new_keyword()
    else:
        break
```

```
Do you want to add another Keyword?
(1) Yes
(2) No
1
Language:
(1) Español
(2) English
2
What is your keyword?
sweatshirt
```

```
Your word is: sweatshirt
This are the CSV on your folder
(1 -> all_data_Pants.csv)
(2 -> Pants.csv)
(3 -> sweatshirt.csv)
Choose the numbers of the CSV that you want to delete or choose 0 to pass
(example 1,3)
0
Los archivos restantes son:
['all_data_Pants.csv', 'Pants.csv', 'sweatshirt.csv']
               Keyword  Search Volume   CPC
0           sweatshirt         165000  1.05
1      hoodies for men         165000  1.18
2  champion sweatshirt         110000  0.68
3  sweatshirt champion         110000  0.66
4    nikeid sweatshirt          74000  0.50
Null Data in Keyword = 0


Null Data in Search Volume = 0


Null Data in CPC = 0


Great Dataset! Continue
62 Keywords
Directory already exists
All data downloaded
Do you want to add another Keyword?
(1) Yes
(2) No
2
```

2.8 Ultimo Filtro

Realizamos un ultimo filtro con las marcas seleccionadas en caso que hayan keywords poco relacionadas de las nuevas busquedas

[25]:
```
Key = Keymarcas(Key)
```

[26]:
```
Key[:30]
```

[26]:
```
['nike',
 'nike shoes',
 'nike outlet',
 'nike hoodie',
 'nike hoodies',
```

```
 'hoodies nike',
 'nike shoes for men',
 'nike backpacks',
 'nike sweatpants',
 'nike sweat pants',
 'nike id',
 'nike air max womens',
 'nike shorts',
 'nike windbreaker',
 'nike sweatsuit',
 'nike joggers',
 'nike sweatshirts',
 'nike sweatshirt',
 'nike t shirt',
 'nike white shoes',
 'nike jacket',
 'nike women',
 'nikeid sweatshirt',
 'nike headbands',
 'sweatshirts nike',
 'nike hoodie men',
 'nike leggings',
 "men's nike hoodie",
 'nike hoodies men',
 'sweatshirt nike']
```

3 Generando textos por medio LSTM

3.1 Caso 1

Vamos a realizar una RNN regular vectorizando nuestras palabras y entrenando el modelo para realizar una prediccion Primero vamos a dejar todas nuestras palabras como texto, y eliminamos cualquier signo de puntuación

Cargamos todos los paquetes necesarios para realizar nuestra RNN por LSTM

[27]:
```
import re
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils
```

```
Using TensorFlow backend.
```

Hacemos de nuestras Keywords un solo texto

[28]:
```
Key_Text= ', '.join(Key)
processed_text = re.sub('[^a-zA-Z]',r' ', Key_Text)
```

Le damos asignamos un valor a todos los signos del vocabulario que se encuentren en el texto

[29]:
```
chars = sorted(list(set(Key_Text)))
char_to_int = dict((c, i) for i, c in enumerate(chars))
char_to_int
```

[29]:
```
{' ': 0,
 "'": 1,
 ',': 2,
 '0': 3,
 '1': 4,
 '2': 5,
 '3': 6,
 '4': 7,
 '5': 8,
 '6': 9,
 '7': 10,
 '9': 11,
 'a': 12,
 'b': 13,
 'c': 14,
 'd': 15,
 'e': 16,
 'f': 17,
 'g': 18,
 'h': 19,
 'i': 20,
 'j': 21,
 'k': 22,
 'l': 23,
 'm': 24,
 'n': 25,
 'o': 26,
 'p': 27,
 'r': 28,
 's': 29,
 't': 30,
 'u': 31,
 'v': 32,
 'w': 33,
 'x': 34,
 'y': 35,
 'z': 36}
```

[30]:
```
n_chars = len(Key_Text)
n_vocab = len(chars)
print ("Total Characters: ", n_chars)
print ("Total Vocab: ", n_vocab)
```

```
Total Characters:  2798
Total Vocab:  37
```

Ahora convertimmos vectorizammos nuestras palabras por medio de la tokenización anterior

[31]:
```
seq_length = 100
dataX = []
dataY = []
for i in range(0, n_chars - seq_length, 1):
    seq_in =  Key_Text[i:i + seq_length]
    seq_out = Key_Text[i + seq_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
n_patterns = len(dataX)
print ("Total Patterns: ", n_patterns)
```

```
Total Patterns:  2698
```

[32]:
```
X = np.reshape(dataX, (n_patterns, seq_length, 1))
# normalize
X = X / float(n_vocab)
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
```

3.1.1 Realizamos nuestra LSTM

[33]:
```
model = Sequential()
model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2]), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(256))
model.add(Dropout(0.2))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

Guardamos un checkpoint de los mejores modelos por epoca

[38]:
```
try:
    os.mkdir("checkpoint")
    print("Directory Created")
except FileExistsError:
    print("Directory already exists")

filepath="checkpoint\weights-improvement-{loss:.3f}.hdf5"
```

```
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1,␣
 ↪save_best_only=True, mode='min')
callbacks_list = [checkpoint]
```

```
Directory already exists
```

Entrenamos nuestra RNN

[35]:
```
model.fit(X, y, epochs=25, batch_size=128, callbacks=callbacks_list)
```

```
Epoch 1/25
2698/2698 [==============================] - 27s 10ms/step - loss: 3.1325

Epoch 00001: loss improved from inf to 3.13248, saving model to
checkpoint\weights-improvement-3.1325.hdf5
Epoch 2/25
2698/2698 [==============================] - 30s 11ms/step - loss: 2.9784

Epoch 00002: loss improved from 3.13248 to 2.97843, saving model to
checkpoint\weights-improvement-2.9784.hdf5
Epoch 3/25
2698/2698 [==============================] - 33s 12ms/step - loss: 2.9675

Epoch 00003: loss improved from 2.97843 to 2.96749, saving model to
checkpoint\weights-improvement-2.9675.hdf5
Epoch 4/25
2698/2698 [==============================] - 37s 14ms/step - loss: 2.9579

Epoch 00004: loss improved from 2.96749 to 2.95791, saving model to
checkpoint\weights-improvement-2.9579.hdf5
Epoch 5/25
2698/2698 [==============================] - 39s 14ms/step - loss: 2.9662

Epoch 00005: loss did not improve from 2.95791
Epoch 6/25
2698/2698 [==============================] - 40s 15ms/step - loss: 2.9617

Epoch 00006: loss did not improve from 2.95791
Epoch 7/25
2698/2698 [==============================] - 41s 15ms/step - loss: 2.9649

Epoch 00007: loss did not improve from 2.95791
Epoch 8/25
2698/2698 [==============================] - 42s 16ms/step - loss: 2.9456

Epoch 00008: loss improved from 2.95791 to 2.94560, saving model to
checkpoint\weights-improvement-2.9456.hdf5
Epoch 9/25
```

```
2698/2698 [==============================] - 46s 17ms/step - loss: 2.9475

Epoch 00009: loss did not improve from 2.94560
Epoch 10/25
2698/2698 [==============================] - 44s 16ms/step - loss: 2.9137

Epoch 00010: loss improved from 2.94560 to 2.91372, saving model to
checkpoint\weights-improvement-2.9137.hdf5
Epoch 11/25
2698/2698 [==============================] - 46s 17ms/step - loss: 2.8695

Epoch 00011: loss improved from 2.91372 to 2.86946, saving model to
checkpoint\weights-improvement-2.8695.hdf5
Epoch 12/25
2698/2698 [==============================] - 49s 18ms/step - loss: 2.8114

Epoch 00012: loss improved from 2.86946 to 2.81142, saving model to
checkpoint\weights-improvement-2.8114.hdf5
Epoch 13/25
2698/2698 [==============================] - 42s 16ms/step - loss: 2.7603

Epoch 00013: loss improved from 2.81142 to 2.76026, saving model to
checkpoint\weights-improvement-2.7603.hdf5
Epoch 14/25
2698/2698 [==============================] - 43s 16ms/step - loss: 2.7161

Epoch 00014: loss improved from 2.76026 to 2.71614, saving model to
checkpoint\weights-improvement-2.7161.hdf5
Epoch 15/25
2698/2698 [==============================] - 43s 16ms/step - loss: 2.6714

Epoch 00015: loss improved from 2.71614 to 2.67138, saving model to
checkpoint\weights-improvement-2.6714.hdf5
Epoch 16/25
2698/2698 [==============================] - 43s 16ms/step - loss: 2.6172

Epoch 00016: loss improved from 2.67138 to 2.61718, saving model to
checkpoint\weights-improvement-2.6172.hdf5
Epoch 17/25
2698/2698 [==============================] - 44s 16ms/step - loss: 2.5458

Epoch 00017: loss improved from 2.61718 to 2.54583, saving model to
checkpoint\weights-improvement-2.5458.hdf5
Epoch 18/25
2698/2698 [==============================] - 45s 16ms/step - loss: 2.4739

Epoch 00018: loss improved from 2.54583 to 2.47393, saving model to
checkpoint\weights-improvement-2.4739.hdf5
```

```
Epoch 19/25
2698/2698 [==============================] - 45s 17ms/step - loss: 2.3979

Epoch 00019: loss improved from 2.47393 to 2.39788, saving model to
checkpoint\weights-improvement-2.3979.hdf5
Epoch 20/25
2698/2698 [==============================] - 44s 16ms/step - loss: 2.2942

Epoch 00020: loss improved from 2.39788 to 2.29420, saving model to
checkpoint\weights-improvement-2.2942.hdf5
Epoch 21/25
2698/2698 [==============================] - 45s 17ms/step - loss: 2.2132

Epoch 00021: loss improved from 2.29420 to 2.21317, saving model to
checkpoint\weights-improvement-2.2132.hdf5
Epoch 22/25
2698/2698 [==============================] - 45s 17ms/step - loss: 2.1280

Epoch 00022: loss improved from 2.21317 to 2.12800, saving model to
checkpoint\weights-improvement-2.1280.hdf5
Epoch 23/25
2698/2698 [==============================] - 48s 18ms/step - loss: 2.0131

Epoch 00023: loss improved from 2.12800 to 2.01307, saving model to
checkpoint\weights-improvement-2.0131.hdf5
Epoch 24/25
2698/2698 [==============================] - 48s 18ms/step - loss: 1.9265

Epoch 00024: loss improved from 2.01307 to 1.92654, saving model to
checkpoint\weights-improvement-1.9265.hdf5
Epoch 25/25
2698/2698 [==============================] - 48s 18ms/step - loss: 1.8630

Epoch 00025: loss improved from 1.92654 to 1.86303, saving model to
checkpoint\weights-improvement-1.8630.hdf5
```

[35]:
```
<keras.callbacks.callbacks.History at 0x215f0310048>
```

[36]:
```
def get_best_modeled():
    csv_files=list(filter(lambda x: '.hdf5' in x, os.listdir("checkpoint")))
    A = []
    B =[]
    C= np.zeros(len(csv_files))
    for i in range(len(csv_files)):
        A.append(csv_files[i].split('-'))
        B.append(A[i][2].split('.h'))
        C[i] = float(B[i][0])
```

```
    mini = np.min(C)
    return mini
mini = get_best_modeled()
```

[37]:
```
filename = "checkpoint\weights-improvement-"+str(mini)+".hdf5"
model.load_weights(filename)
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

Realizamos una prediccion de texto

[38]:
```
char_to_ints = dict((i, c) for i, c in enumerate(chars))
```

[39]:
```
start = np.random.randint(0, len(dataX)-1)
pattern = dataX[start]
print ("Seed:")
print ("\"", ''.join([char_to_ints[value] for value in pattern]), "\"")
# generate characters
for i in range(10):
    x = np.reshape(pattern, (1, len(pattern), 1))
    x = x / float(n_vocab)
    prediction = model.predict(x, verbose=0)
    index = np.argmax(prediction)
    result = char_to_ints[index]
    seq_in = [char_to_ints[value] for value in pattern]
    sys.stdout.write(result)
    pattern.append(index)
    pattern = pattern[1:len(pattern)]
print ("\n Done.")
```

```
Seed:
" der armour sale, under armour jacket, under armour hoodies men's, under armour
shorts women, under a "
rmour sooe
 Done.
```

3.2 Caso 2

Nuestra LSTM producirá un nuevo texto carácter por carácter. Tendremos que muestrear la letra correcta de las predicciones de la LSTM cada vez. La función de muestra acepta los siguientes dos parámetros:

- preds -> Las neuronas de salida.
- temperatura: 1.0 es el más conservador, 0.0 es el más arriesgado (dispuesto a cometer errores ortográficos y de otro tipo).

La función realiza básicamente básicamenteun softmax en las predicciones de la red neuronal. Esto hace que cada neurona de salida se convierta en una probabilidad de su letra particular.

Cargamos Paquete faltantes.

[40]:
```
from tensorflow.keras.callbacks import LambdaCallback
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.utils import get_file
import random
import io
```

[41]:
```
print('corpus length:', len(processed_text))

chars = sorted(list(set(processed_text)))
print('total chars:', len(chars))
char_indices = dict((c, i) for i, c in enumerate(chars))
indices_char = dict((i, c) for i, c in enumerate(chars))
```

```
corpus length: 2798
total chars: 26
```

Revisamos cuantas subsecuencias de palabras podemos generar de nuestra lista de Keywords.

[42]:
```
maxlen = 20
step = 3
sentences = []
next_chars = []
for i in range(0, len(processed_text) - maxlen, step):
    sentences.append(processed_text[i: i + maxlen])
    next_chars.append(processed_text[i + maxlen])
print('nb sequences:', len(sentences))
```

```
nb sequences: 926
```

Vectorizamos nuestras Keywords.

[43]:
```
print('Vectorization...')
x = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        x[i, t, char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1
```

```
Vectorization...
```

Construimos nuestro modelo LSTM.

[44]:
```
# build the model: a single LSTM
print('Build model...')
model = Sequential()
model.add(LSTM(128, input_shape=(maxlen, len(chars))))
model.add(Dense(len(chars), activation='softmax'))
```

```
optimizer = RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer)
```

```
Build model...
```

Revisamos la arquitectura de nuestra LSTM.

[45]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 128)               79360
_________________________________________________________________
dense_2 (Dense)              (None, 26)                3354
=================================================================
Total params: 82,714
Trainable params: 82,714
Non-trainable params: 0
_________________________________________________________________
```

3.2.1 Añadimos las variables preds y temperature

Esto nos permite cambiar las probabilidades por cada letra desde un modo conservador hasta lo más arriesgado.

[46]:
```
def sample(preds, temperature=1.0):
    # helper function to sample an index from a probability array
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
```

La funcion on_epoch_end nos va a permitir imprimir una generacion de texto por epoca iterando sobre las temperaturas y los preds

[47]:
```
def on_epoch_end(epoch, _):
    # Function invoked at end of each epoch. Prints generated text.
    print()
    print("******************************************************************************")
    print('----- Generating text after Epoch: %d' % epoch)

    start_index = random.randint(0, 30)
    for temperature in [0.2, 0.5, 1.0]:
```

```
        print('----- temperature:', temperature)

        generated = ''
        sentence = processed_text[start_index: start_index + maxlen]
        generated += sentence
        print('----- Generating with seed: "' + sentence + '"')
        sys.stdout.write(generated)

        for i in range(50):
            x_pred = np.zeros((1, maxlen, len(chars)))
            for t, char in enumerate(sentence):
                x_pred[0, t, char_indices[char]] = 1.

            preds = model.predict(x_pred, verbose=0)[0]
            next_index = sample(preds, temperature)
            next_char = indices_char[next_index]

            generated += next_char
            sentence = sentence[1:] + next_char

            sys.stdout.write(next_char)
            sys.stdout.flush()
        print()
```

3.2.2 Lamda Callback

Un Callback puede realizar acciones en varias etapas del entrenamiento (por ejemplo, al comienzo o al final de una época, antes o después de un solo lote, etc.).

Se pueden usar Callbacks para:

- Escribir registros de TensorBoard después de cada lote de entrenamiento para monitorear sus métricas
- Periódicamente guarde su modelo en el disco
- Haz paradas tempranas
- Obtener una vista de los estados internos y las estadísticas de un modelo durante el entrenamiento

De esta forma lo podemos utilizar como:

> tf.keras.callbacks.LambdaCallback(on_epoch_begin=None, on_epoch_end=None, on_batch_begin=None, on_batch_end=None, on_train_begin=None, on_train_end=None, **kwargs)

En este caso vamos a utilizar LambdaCallback para imprimir nuestra las predicciones hechas en la funcion on_epoch_end al final de cada entrenamiento de época

[48]:
```
# See https://github.com/tensorflow/tensorflow/issues/31308
import logging, os
```

```
logging.disable(logging.WARNING)
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

# Fit the model
print_callback = LambdaCallback(on_epoch_end=on_epoch_end)

model.fit(x, y,
          batch_size=128,
          epochs=25,
          callbacks=[print_callback])
```

```
Epoch 1/25
926/926 [==============================] - 1s 797us/step - loss: 3.1668
**************************************************************************
----- Generating text after Epoch: 0
----- temperature: 0.2
----- Generating with seed: "hoes  nike outlet  n"
hoes  nike outlet  naaaaaaaasaasaasaaaaaaaaaaaaa aaa aaaaaaaaaaaaaaaa
----- temperature: 0.5
----- Generating with seed: "hoes  nike outlet  n"
hoes  nike outlet  nessaaaaa assasaaaa aaausaaaand asa a aada aaaaas a
----- temperature: 1.0
----- Generating with seed: "hoes  nike outlet  n"
hoes  nike outlet  n ieshsa snvaoeasera  nsa cdisaa  pdsuso uapdsaa me
Epoch 2/25
926/926 [==============================] - 1s 569us/step - loss: 2.7395
**************************************************************************
----- Generating text after Epoch: 1
----- temperature: 0.2
----- Generating with seed: "e shoes  nike outlet"
e shoes  nike outlet r   s  r  r       s           n         n
----- temperature: 0.5
----- Generating with seed: "e shoes  nike outlet"
e shoes  nike outletuossnusar  s    rc g ms     m ssss ror  r s sr sa
----- temperature: 1.0
----- Generating with seed: "e shoes  nike outlet"
e shoes  nike outletessuarsuri stpusre ain ao  tg  ortn  gnjjlrpm brmp
Epoch 3/25
926/926 [==============================] - 1s 570us/step - loss: 2.6737
**************************************************************************
----- Generating text after Epoch: 2
----- temperature: 0.2
----- Generating with seed: "e shoes  nike outlet"
e shoes  nike outlet
----- temperature: 0.5
----- Generating with seed: "e shoes  nike outlet"
e shoes  nike outlet   a  a    t athsa      aae u     a sai   m ae   a
```

```
----- temperature: 1.0
----- Generating with seed: "e shoes  nike outlet"
e shoes  nike outlet ve   art hmh  et  nad  l tps u ltt  ros  dtot a e
Epoch 4/25
926/926 [==============================] - 1s 577us/step - loss: 2.5974
********************************************************************************
----- Generating text after Epoch: 3
----- temperature: 0.2
----- Generating with seed: "nike outlet  nike ho"
nike outlet  nike ho
----- temperature: 0.5
----- Generating with seed: "nike outlet  nike ho"
nike outlet  nike ho   w ms    la     mhoe a  a o oa uoae s  s ua  pum
----- temperature: 1.0
----- Generating with seed: "nike outlet  nike ho"
nike outlet  nike hoa jteooes n tuurnslemhhosorusma  o r al leimsvgeu
Epoch 5/25
926/926 [==============================] - 1s 564us/step - loss: 2.4578
********************************************************************************
----- Generating text after Epoch: 4
----- temperature: 0.2
----- Generating with seed: "e outlet  nike hoodi"
e outlet  nike hoodida aaamamaaa aaaaaamaaamaaamaaamaaaaauamaamaaaaaaa
----- temperature: 0.5
----- Generating with seed: "e outlet  nike hoodi"
e outlet  nike hoodia  ma aadaa aaaamarr amauammummaaa amadum a amaaaa
----- temperature: 1.0
----- Generating with seed: "e outlet  nike hoodi"
e outlet  nike hoodiouuidr momeadaadama  mtmua umrak  m mar oaaammaoma
```

[48]: <keras.callbacks.callbacks.History at 0x215874b5808>

3.3 Retos:

3.3.1 Añadir Money Keywords

Son aquellas que no solo atraen clientes potenciales directamente al sitio, sino que también atraen a los clientes potenciales correctos a su sitio: los clientes que sacarán su tarjeta de crédito y pagarán dinero.

Por ejemplo

Oferta, Préstamos, Hipoteca, Crédito, Hospedaje y Reclamación

3.3.2 Implementar el Keyword Research en un sitio web

Al momento de implementar las palabras generadas en un sitio web, podremos observar y cuantizar sus resultados en los buscadores

3.4 Conclusión

El proyecto de Keyword Research nos permitió conocer cual es la verdadera labor del SEO y su importancia en el Marketing Digital para sitios web, es un gran ejercicio para aprender sobre scrapping con Selenium, manejar archivos del sistema e incluso aprender sobre el verdadero significado de Marketing y varias de sus estrategias para lograr acercar lo mejor posible servicios y sitios web a clientes potenciales.

Realizar KeyWord Research a traves de Python y con ayuda de Machine Learning, es un campo con muchas ideas por explorar y poco desarrolladas, y que con más trabajo se pueden realizar algoritmos muy eficientes para un posicionamiento Web orgánico.

Todos estos proyectos nos permiten conocer nuevos algoritmos, trabajarlos y entenderlos de la mejor forma, además, estamos aprendiendo sobre el contexto en que podemos aplicar estos, entender las caracteristicas de las diferentes formas de aplicar Machine Learning en varias temáticas de todo tipo.