Clasificacion_3

September 11, 2022

1 Regresión Logística: Detección de SPAM

En este ejercicio se muestran los fundamentos de la Regresión Logística planteando uno de los primeros problemas que fueron solucionados mediante el uso de técnicas de Machine Learning: la detección de SPAM.

1.1 Enunciado del ejercicio

Se propone la construcción de un sistema de aprendizaje automático capaz de predecir si un correo determinado se corresponde con un correo de SPAM o no, para ello, se utilizará el siguiente conjunto de datos:

2007 TREC Public Spam Corpus The corpus trec07p contains 75,419 messages:

```
25220 ham
50199 spam
```

These messages constitute all the messages delivered to a particular server between these dates:

```
Sun, 8 Apr 2007 13:07:21 -0400
Fri, 6 Jul 2007 07:04:53 -0400
```

1.1.1 Funciones complementarias

En este caso práctico relacionado con la detección de correos electrónicos de SPAM, el conjunto de datos que disponemos esta formado por correos electrónicos, con sus correspondientes cabeceras y campos adicionales. Por lo tanto, requieren un preprocesamiento previo a que sean ingeridos por el algoritmo de Machine Learning.

[ ]:
```
# Esta clase facilita el preprocesamiento de correos electrónicos que poseen
↪código HTML
from html.parser import HTMLParser

class MLStripper(HTMLParser):
    def __init__(self):
        self.reset()
        self.strict = False
```

```
        self.convert_charrefs = True
        self.fed = []

    def handle_data(self, d):
        self.fed.append(d)

    def get_data(self):
        return ''.join(self.fed)
```

[ ]:
```
# Esta función se encarga de elimar los tags HTML que se encuentren en el texto del correo electrónico
def strip_tags(html):
    s = MLStripper()
    s.feed(html)
    return s.get_data()
```

[ ]:
```
# Ejemplo de eliminación de los tags HTML de un texto
t = '<tr><td align="left"><a href="../../issues/51/16.html#article">Phrack World News</a></td>'
strip_tags(t)
```

[ ]:
```
'Phrack World News'
```

Además de eliminar los posibles tags HTML que se encuentren en el correo electrónico, deben realizarse otras acciones de preprocesamiento para evitar que los mensajes contengan ruido innecesario. Entre ellas se encuentra la eliminación de los signos de puntuación, eliminación de posibles campos del correo electrónico que no son relevantes o eliminación de los afijos de una palabra manteniendo únicamente la raiz de la misma (Stemming). La clase que se muestra a continuación realiza estas transformaciones.

[ ]:
```
import email
import string
import nltk

class Parser:

    def __init__(self):
        self.stemmer = nltk.PorterStemmer()
        self.stopwords = set(nltk.corpus.stopwords.words('english'))
        self.punctuation = list(string.punctuation)

    def parse(self, email_path):
        """Parse an email."""
        with open(email_path, errors='ignore') as e:
            msg = email.message_from_file(e)
        return None if not msg else self.get_email_content(msg)
```

```
    def get_email_content(self, msg):
        """Extract the email content."""
        subject = self.tokenize(msg['Subject']) if msg['Subject']
        else [] body = self.get_email_body(msg.get_payload(),
        msg.get_content_type())
        content_type =
        msg.get_content_type() #
        Returning the content of the
        email return {"subject":
        subject,
               "body": body,
               "content_type": content_type}

    def get_email_body(self, payload, content_type):
        """Extract the body of the email."""
        body = [] if type(payload) is str and
        content_type == 'text/plain': return
        self.tokenize(payload)
        elif type(payload) is str and content_type == 'text/html':
            return self.tokenize(strip_tags(payload))
        elif type(payload) is list:
            for p in payload:
                body += self.get_email_body(p.get_payload(),
                                            p.get_content_type())
        return body

    def tokenize(self, text):
        """Transform a text string in tokens. Perform two main
        actions, clean the punctuation symbols and do stemming
        of the text.""" for c in self.punctuation:
            text = text.replace(c, "")
        text = text.replace("\t", "
        ") text = text.replace("\n",
        " ")
        tokens = list(filter(None, text.split(" ")))
        # Stemming of the tokens return [self.stemmer.stem(w) for w in
        tokens if w not in self.stopwords]
```

Lectura de un correo en formato raw

[ ]:
```
inmail = open("datasets/trec07p/data/inmail.1").read()
print(inmail)
```

```
From RickyAmes@aol.comSun Apr8 13:07:32 2007
 Return-Path: <RickyAmes@aol.com>
```

```
Received: from 129.97.78.23 ([211.202.101.74]) by
       speedy.uwaterloo.ca (8.12.8/8.12.5) with SMTP id
       l38H7G0I003017; Sun, 8 Apr 2007 13:07:21 -0400
Received: from 0.144.152.6 by 211.202.101.74; Sun, 08 Apr 2007
19:04:48 +0100
Message-ID: <WYADCKPDFWWTWTXNFVUE@yahoo.com>
From: "Tomas Jacobs" <RickyAmes@aol.com>
Reply-To: "Tomas Jacobs" <RickyAmes@aol.com>
To: the00@speedy.uwaterloo.ca
Subject: Generic Cialis, branded quality@
Date: Sun, 08 Apr 2007 21:00:48 +0300
X-Mailer: Microsoft Outlook Express 6.00.2600.0000
MIME-Version: 1.0
Content-Type: multipart/alternative; boundary="--
       8896484051606557286"
X-Priority: 3
X-MSMail-Priority: Normal
Status: RO
Content-Length: 988
Lines: 24

----8896484051606557286
Content-Type: text/html;
Content-Transfer-Encoding: 7Bit

<html>
<body bgcolor="#ffffff">
<div style="border-color: #00FFFF; border-right-width: 0px; border-
bottom-width:
0px; margin-bottom: 0px;" align="center">
<table style="border: 1px; border-style: solid; border-
color:#000000;" cellpadding="5" cellspacing="0" bgcolor="#CCFFAA">
<tr>
<td style="border: 0px; border-bottom: 1px; border-style: solid;
bordercolor:#000000;">
<center>
Do you feel the pressure to perform and not rising to the
occasion??<br> </center>
</td></tr><tr>
<td bgcolor=#FFFF33 style="border: 0px; border-bottom: 1px; border-
style: solid; border-color:#000000;"> <center>

<b><a href='http://excoriationtuh.com/?lzmfnrdkleks'>Try
<span>V</span><span>ia<span></span>gr<span>a</span>…</a></b></center>
</td></tr><td><center>your anxiety will be a thing of the past and
you will<br> be back to your old self.
```

```
</center></td></tr></table></div></body></html>


----8896484051606557286--
```

Parsing del correo electrónico

```
[ ]: p = Parser()
     p.parse("datasets/trec07p/data/inmail.1")
```

```
[ ]: {'subject': ['gener', 'ciali', 'brand', 'qualiti'],
      'body': ['Do',
       'feel',
       'pressur',
       'perform',
       'rise',
       'occas',
       'tri',
       'viagra',
       'anxieti',
       'thing',
       'past',
       'back',
       'old',
       'self'],
      'content_type': 'multipart/alternative'}
```

Lectura del índice Estas funciones complementarias se encargan cargar en memoria la ruta de cada correo electrónico y su etiqueta correspondiente {spam, ham}

```
[ ]: index = open("datasets/trec07p/full/index").readlines()
     index
```

```
[ ]: ['spam ../data/inmail.1\n',
      'ham ../data/inmail.2\n',
      'spam ../data/inmail.3\n',
      'spam ../data/inmail.4\n',
      'spam ../data/inmail.5\n',
      'spam ../data/inmail.6\n',
      'spam ../data/inmail.7\n',
      'spam ../data/inmail.8\n',
      'spam ../data/inmail.9\n',
      'ham ../data/inmail.10\n',
      'spam ../data/inmail.11\n',
      'spam ../data/inmail.12\n',
      'spam ../data/inmail.13\n',
      'spam ../data/inmail.14\n',
```

```
'spam ../data/inmail.15\n',
'spam ../data/inmail.16\n',
'spam ../data/inmail.17\n',
'spam ../data/inmail.18\n',
'spam ../data/inmail.19\n',
```

```
def parse_index(path_to_index, n_elements):
    ret_indexes = []
    index = open(path_to_index).readlines()
    for i in range(n_elements):
        mail = index[i].split(" ../")
        label = mail[0]
        path = mail[1][:-1]
        ret_indexes.append({"label":label, "email_path":os.path.
↪join(DATASET_PATH, path)})
    return ret_indexes
```

[ ]:
```
def parse_email(index):
    p = Parser()
    pmail = p.parse(index["email_path"])
    return pmail, index["label"]
```

[ ]:
```
indexes = parse_index("datasets/trec07p/full/index", 10)
indexes
```

[ ]:
```
[{'label': 'spam', 'email_path': 'datasets/trec07p/data/inmail.1'},
 {'label': 'ham', 'email_path': 'datasets/trec07p/data/inmail.2'},
 {'label': 'spam', 'email_path': 'datasets/trec07p/data/inmail.3'},
 {'label': 'spam', 'email_path': 'datasets/trec07p/data/inmail.4'},
 {'label': 'spam', 'email_path': 'datasets/trec07p/data/inmail.5'},
 {'label': 'spam', 'email_path': 'datasets/trec07p/data/inmail.6'},
 {'label': 'spam', 'email_path': 'datasets/trec07p/data/inmail.7'},
 {'label': 'spam', 'email_path': 'datasets/trec07p/data/inmail.8'},
 {'label': 'spam', 'email_path': 'datasets/trec07p/data/inmail.9'},
 {'label': 'ham', 'email_path': 'datasets/trec07p/data/inmail.10'}]
```

1.1.2 Preprocesamiento de los datos del conjunto de datos

Con las funciones presentadas anteriormente se permite la lectura de los correos electrónicos de manera programática y el procesamiento de los mismos para eliminar aquellos componentes que no resultan de utilidad para la detección de correos de SPAM. Sin embargo, cada uno de los correos sigue estando representado por un diccionario de Python con una serie de palabras.

[ ]:
```
# Cargamos el índice y las etiquetas en memoria
index = parse_index("datasets/trec07p/full/index", 1)
```

[ ]:
```
# Leemos el primer correo
import os

open(index[0]["email_path"]).read()
```

[ ]:
```
'From RickyAmes@aol.comSun Apr8 13:07:32 2007\nReturn-Path:
```

```
<RickyAmes@aol.com>\nReceived: from 129.97.78.23
([211.202.101.74])\n\tby speedy.uwaterloo.ca (8.12.8/8.12.5) with
SMTP id l38H7G0I003017;\n\tSun, 8 Apr 2007 13:07:21 -0400\nReceived:
from 0.144.152.6 by 211.202.101.74; Sun, 08 Apr
2007 19:04:48 +0100\nMessage-ID:
<WYADCKPDFWWTWTXNFVUE@yahoo.com>\nFrom: "Tomas Jacobs"
<RickyAmes@aol.com>\nReply-To: "Tomas Jacobs"
<RickyAmes@aol.com>\nTo: the00@speedy.uwaterloo.ca\nSubject: Generic
Cialis, branded quality@ \nDate: Sun, 08 Apr 2007 21:00:48 +0300\nX-
Mailer: Microsoft Outlook Express 6.00.2600.0000\nMIME-Version:
1.0\nContent-Type:
multipart/alternative;\n\tboundary="--8896484051606557286"\nX-
Priority: 3\nXMSMail-Priority: Normal\nStatus: RO\nContent-Length:
988\nLines: 24\n\n---8896484051606557286\nContent-Type:
text/html;\nContent-Transfer-Encoding: 7Bit\n\n<html>\n<body
bgcolor="#ffffff">\n<div style="border-color: #00FFFF; border-right-
width: 0px; border-bottom-width: 0px; margin-bottom: 0px;"
align="center">\n<table style="border: 1px; border-style: solid;
bordercolor:#000000;" cellpadding="5" cellspacing="0"
bgcolor="#CCFFAA">\n<tr>\n<td style="border: 0px; border-bottom: 1px;
border-style: solid; bordercolor:#000000;">\n<center>\nDo you feel
the pressure to perform and not rising to the
occasion??<br>\n</center>\n</td></tr><tr>\n<td bgcolor=#FFFF33
style="border: 0px; border-bottom: 1px; border-style: solid;
bordercolor:#000000;">\n<center>\n\n<b><a
href=\'http://excoriationtuh.com/?lzmfnrdkleks\'>Try
<span>V</span><span>ia<span
></span>gr<span>a</span>…</a></b></center>\n</td></tr><td><center>
your anxiety will be a thing of the past and you will<br>\nbe back
to your old
self.\n</center></td></tr></table></div></body></html>\n\n\n---
8896484051606557286--\n\n'
```

[ ]:
```
# Parseamos el primer correo
mail, label = parse_email(index[0])
print("El correo es:", label)
print(mail)
```

```
El correo es: spam
{'subject': ['gener', 'ciali', 'brand', 'qualiti'], 'body': ['Do',
'feel',
'pressur', 'perform', 'rise', 'occas', 'tri', 'viagra', 'anxieti',
'thing', 'past', 'back', 'old', 'self'], 'content_type':
'multipart/alternative'}
```

El algoritmo de Regresión Logística no es capaz de ingerir texto como parte del conjunto de datos. Por lo tanto, deben aplicarse una serie de funciones adicionales que transformen el texto de los correos electrónicos parseados en una representación numérica.

Aplicación de CountVectorizer

[ ]:
```
from sklearn.feature_extraction.text import CountVectorizer

# Preapración del email en una cadena de texto
prep_email = [" ".join(mail['subject']) + " ".join(mail['body'])]
```

```
vectorizer = CountVectorizer()
X = vectorizer.fit(prep_email)

print("Email:", prep_email, "\n")
print("Características de entrada:", vectorizer.get_feature_names())
```

```
Email: ['gener ciali brand qualitiDo feel pressur perform rise occas
tri viagra anxieti thing past back old self']

Características de entrada: ['anxieti', 'back', 'brand', 'ciali',
'feel',
'gener', 'occas', 'old', 'past', 'perform', 'pressur', 'qualitido',
'rise',
'self', 'thing', 'tri', 'viagra']
```

[ ]:
```
X = vectorizer.transform(prep_email)
print("\nValues:\n",  .toarray())
```

```
Values:
 [[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]]
```

Aplicación de OneHotEncoding

[ ]:
```
from sklearn.preprocessing import OneHotEncoder

prep_email = [[w] for w in mail['subject'] + mail['body']]

enc = OneHotEncoder(handle_unknown='ignore')
X = enc.fit_transform(prep_email)

print("Features:\n", enc.get_feature_names())
print("\nValues:\n",  .toarray())
```

```
Features:
```

```
 ['x0_Do' 'x0_anxieti' 'x0_back' 'x0_brand' 'x0_ciali' 'x0_feel'
 'x0_gener'
 'x0_occas' 'x0_old' 'x0_past' 'x0_perform' 'x0_pressur' 'x0_qualiti'
 'x0_rise' 'x0_self' 'x0_thing' 'x0_tri' 'x0_viagra']

Values:
 [[0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
```

Funciones auxiliares para preprocesamiento del conjunto de datos

[ ]:
```
def create_prep_dataset(index_path, n_elements):
    X = []
    y = []
    indexes = parse_index(index_path, n_elements)
    for i in range(n_elements):
        print("\rParsing email: {0}".format(i+1), end='')
        mail, label = parse_email(indexes[i])
        X.append(" ".join(mail['subject']) + " ".join(mail['body']))
        y.append(label)
    return X, y
```

1.1.3 Entrenamiento del algoritmo

[ ]:
```
# Leemos únicamente un subconjunto de 100 correos electrónicos
X_train, y_train = create_prep_dataset("datasets/trec07p/full/index",
100) X_train
```

```
Parsing email: 100
```

```
[ ]: ['gener ciali brand qualitiDo feel pressur perform rise occas tri
     viagra anxieti thing past back old self',
      'typo debianreadmHi ive updat gulu I check mirror It seem littl typo
     debianreadm file exampl httpgulususherbrookecadebianreadm
     ftpftpfrdebianorgdebianreadm test lenni access releas diststest the
     current test develop snapshot name etch packag test unstabl pass
     autom test propog releas etch replac lenni like readmehtml yan morin
     consult en logiciel libr yanmorinsavoirfairelinuxcom 5149941556 To
     unsubscrib email debianmirrorsrequestlistsdebianorg subject
     unsubscrib troubl contact listmasterlistsdebianorg',
      'authent viagramega authenticv I A G R A discount pricec I A L I S
     discount pricedo miss IT click httpwwwmoujsjkhchumcom authent viagra
     mega authenticv I A
     G R A discount pricec I A L I S discount pricedo miss IT click',
      'nice talk yahey billi realli fun go night talk said felt insecur
     manhood I notic toilet quit small area worri websit I tell secret
     weapon extra 3 inch trust girl love bigger one ive 5 time mani chick
     sinc I use pill year ago the
```

```
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090909
090909090909090909090909090909090909090909090920 2020 I hope other
easy2',
```

```
'lose weight quicklihoodialif start lose weight now hoodialif natur
substanc liter take appetit away learn buy hoodialif \xa0hoodialif
start lose weight now
```


Aplicamos la vectorización a los datos

[ ]:
```
vectorizer = CountVectorizer()
X_train = vectorizer.fit_transform(X_train)
```

[ ]:
```
print(X_train.toarray())
print("\nFeatures:", len(vectorizer.get_feature_names()))
```

```
[ [ 0 0 0 … 0 0 0 ]
 [0 0 0 … 0 0 0]
 [0 0 0 … 0 0 0]
 …
 [0 0 0 … 0 0 0]
 [0 0 0 … 0 0 0]
 [0 0 0 … 0 0 0]]

Features: 4911
```

[ ]:
```
import pandas as pd

pd.DataFrame(X_train.toarray(), columns=[vectorizer.get_feature_names()])
```

[ ]:

	0000	000000	00085	002	003	00450	009	01	01000u	0107…	anz	š	\
0	00	0	0	0	0	0	0	0	0	0	…	0	0
	0	0											
1	00	0	0	0	0	0	0	0	0	0	…	0	0
	0	0											
2	00	0	0	0	0	0	0	0	0	0	…	0	0
	0	0											
3	00	0	0	0	0	0	0	0	0	0	…	0	0
	0	0											
4	00	0	0	0	0	0	0	0	0	0	…	0	0
	0	0											
..	…	…	…	…	..	..	…	..	..	…	…	…	…	..	…	..
95	00	0	0	0	0	0	0	0	0	0	…	0	0
	0	0											
96	00	0	0	0	0	0	0	0	0	0	…	0	0
	0	0											
97	00	0	0	0	0	0	0	0	0	0	…	0	0
	0	0											
98	00	0	0	0	0	0	0	0	0	0	…	0	0
	0	0											

```
    99   00     0     0     0     0     0     0     0     0     …     0     0
           0     0

        ŵþ ţ ï26 jwk ã
    0    0 0     0     0     0     0
    1    0 0     0     0     0     0
    2    0 0     0     0     0     0
    3    0 0     0     0     0     0
    4    0 0     0     0     0     0
    ..  ..     …  ..   …  …  ..
    95   0 0     0     0     0     0
    96   0 0     0     0     0     0
    97   0 0     0     0     0     0
    98   0 0     0     0     0     0
    99   0 0     0     0     0     0
 [100 rows x 4911 columns]
```

[ ]:
```
y_train
```

[ ]:
```
['spam',
 'ham',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'ham',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'ham',
 'ham',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'ham',
```

```
'spam',
'ham',
'spam',
'spam',
'ham',
'spam',
'spam',
'spam',
'spam',
'spam',
'spam',
'spam',
'spam',
'spam',
'spam',
'spam',
'spam',
'spam',
'spam',
'ham',
'ham',
'spam',
'spam',
'spam',
'spam',
'spam',
'spam',
'ham',
'spam',
'ham',
'spam',
'spam',
'ham',
'spam',
'spam',
'spam',
'ham',
'ham',
'spam',
'spam',
'spam',
'ham',
'ham',
'ham',
'spam',
'ham',
'spam',
```

```
 'ham',
 'spam',
 'spam',
 'spam',
 'ham',
 'spam',
 'spam',
 'ham',
 'spam',
 'ham',
 'spam',
 'spam',
 'ham',
 'spam', 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam',
 'spam']
```

Entrenamiento del algoritmo de regresión logística con el conjunto de datos preprocesado

```
[ ]: from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
clf.fit(X_train, y_train)
```

```
[ ]: LogisticRegression(C=1.0, class_weight=None, dual=False,
                        fit_intercept=True, intercept_scaling=1,
                        l1_ratio=None, max_iter=100, multi_class='auto',
                        n_jobs=None, penalty='l2', random_state=None,
                        solver='lbfgs', tol=0.0001, verbose=0,
                        warm_start=False)
```

1.1.4 Predicción

Lectura de un conjunto de correos nuevos

```
[ ]: # Leemos 150 correos de nuestro conjunto de datos y nos quedamos
     únicamente con␣ ↪los 50 últimos
     # Estos 50 correos electrónicos no se han utilizado para entrenar el
     algoritmo
```

```
X, y =
create_prep_dataset("datasets/trec07p/full/index",
150) X_test = X[100:] y_test = y[100:]
```

```
Parsing email: 150
```

Preprocesamiento de los correos con el vectorizador creado anteriormente

[ ]:
```
X_test = vectorizer.transform(X_test)
```

Predicción del tipo de correo

[ ]:

```
y_pred
```

```
y_pred = clf.predict(X_test) [ ]: array(['spam', 'spam', 'spam', 'spam',
'spam', 'spam', 'spam', 'spam',
           'spam', 'spam', 'ham', 'spam', 'spam', 'spam', 'spam', 'spam',
           'ham', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam',
           'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam',
           'spam',
           'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam',
           'spam',
           'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam',
           'spam', 'spam', 'spam'], dtype='<U4')
```

[ ]:
```
print("Predicción:\n", y_pred)
print("\nEtiquetas reales:\n", y_test)
```

```
Predicción:
 ['spam' 'spam' 'spam' 'spam' 'spam' 'spam' 'spam' 'spam' 'spam'
 'spam'
 'ham' 'spam' 'spam' 'spam' 'spam' 'spam' 'ham' 'spam' 'spam' 'spam'
 'spam' 'spam' 'spam' 'spam' 'spam' 'spam' 'spam' 'spam' 'spam'
 'spam'
 'spam' 'spam' 'spam' 'spam' 'spam' 'spam' 'spam' 'spam' 'spam'
 'spam'
 'spam' 'spam' 'spam' 'spam' 'spam' 'spam' 'spam' 'spam' 'spam'
 'spam']

Etiquetas reales:
 ['spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam',
 'spam',
'spam', 'ham', 'spam', 'spam', 'spam', 'spam', 'spam', 'ham', 'spam',
'spam',
'spam', 'spam', 'spam', 'spam', 'ham', 'spam', 'spam', 'ham', 'spam',
'spam',
```

```
'ham', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam',
'spam', 'spam',
'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam',
'spam', 'spam', 'spam']
```

Evaluación de los resultados

```
[ ]: from sklearn.metrics import accuracy_score

     print('Accuracy: {:.3f}'.format(accuracy_score(y_test, y_pred)))
```

```
Accuracy: 0.940
```

1.1.5 Aumentando el conjunto de datos

```
[ ]: # Leemos 12000 correos electrónicos
     X, y = create_prep_dataset("datasets/trec07p/full/index", 12000)
```

```
Parsing email: 12000
```

```
[ ]: # Utilizamos 10000 correos electrónicos para entrenar el algoritmo y
      2000 para␣ ↪realizar pruebas
     X_train, y_train = X[:10000], y[:10000]
     X_test, y_test = X[10000:], y[10000:]
```

```
[ ]: vectorizer = CountVectorizer()
     X_train = vectorizer.fit_transform(X_train)
```

```
[ ]: clf = LogisticRegression()
     clf.fit(X_train, y_train)
```

```
/opt/anaconda3/lib/python3.7/sitepackages/sklearn/linear_model/_logis
tic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
    shown in: https://scikit-
    learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver
options:
    https://scikit-
learn.org/stable/modules/linear_model.html#logisticregression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
[ ]: LogisticRegression(C=1.0, class_weight=None, dual=False,
                        fit_intercept=True, intercept_scaling=1,
                        l1_ratio=None, max_iter=100, multi_class='auto',
                        n_jobs=None, penalty='l2', random_state=None,
```

```
                   solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[ ]:
```
X_test = vectorizer.transform(X_test)
```

[ ]:
```
y_pred = clf.predict(X_test)
```

[ ]:
```
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, y_pred)))
```

```
Accuracy: 0.987
```