

# U03 PRÁCTICA 1

## ALGORITMOS CLASIFICADORES QUE TRABAJAN CON DATOS NO ESTRUCTURADOS: TEXTO

**SISTEMAS  
DE APRENDIZAJE  
AUTOMÁTICO**

**IES SERRA PERENXISA  
TORRENT (VALENCIA)**



# ALGORITMOS DE APRENDIZAJE QUE TRABAJAN CON TEXTO

## ACTIVIDAD 1: UN CLASIFICADOR NAÏVE BAYES DESDE CERO.

Implementar un filtro de spam es uno de los problemas típicos que se realizan cuando aprendes machine learning. Las librerías de ML como scikit-learn son perfectas para implementar modelos sin apenas esfuerzo, como es el caso de esta práctica. Sin embargo, para comprender mejor los algoritmos y el propio modelo vamos a comenzar realizando una implementación sin ayuda de ninguna librería, solo código Python. Crea una carpeta llamada **actividad1** y crea el fichero **a1.py** en ella.

### DATOS DE TRAIN Y TEST

Vamos a definir unos pocos datos para train y test para cada clase de correo (ham y spam) que por claridad van a ser escasos y muy cortos. Supondremos que cada mail es uno de los textos del array.

```
1  # -*- coding: utf-8 -*-
2  train_spam = ['send us your password', 'review our website',
3               'send your password', 'send us your account']
4  train_ham = ['Your activity report', 'benefits physical activity',
5               'the importance vows']
6  test_emails = {'spam': ['renew your password', 'renew your vows'],
7                 'ham': ['benefits of our account', 'the importance of physical activity']}
```

Vamos a iterar por cada mail etiquetado como spam y para cada palabra *w* de todo el conjunto de train contamos cuantos emails de spam la contienen:

```
9  # Hacer un vocabulario de palabras únicas que aparecen en los mails spam
10 vocab_palabras_spam = []
11 for frase in train_spam:
12     frase_como_lista = frase.split()
13     for w in frase_como_lista:
14         vocab_palabras_spam.append(w)
15 print(vocab_palabras_spam)
```

Convertimos cada lista en un diccionario siendo las claves cada palabra de la lista para eliminar duplicados (un diccionario no puede tener claves duplicadas):

```
17 vocab_palabras_spam_unicas = list(dict.fromkeys(vocab_palabras_spam))
18 print(vocab_palabras_spam_unicas)
```

Ahora hay que calcular las probabilidades de cada palabra de estar en spam (su spamicidad) y para ello hay que contar sus apariciones (su frecuencia de aparición nos permite estimar estas probabilidades). Esto nos permite aproximar la probabilidad condicionada a que sean spam. Podemos contar cuantos mail de tipo spam tienen la palabra "send" y dividir el contador entre el número total de mail que son spam.

```
19 # Probabilidades de cada palabra condicionado a spam
20 dict_spamicidad = {}
21 for w in vocab_palabras_spam_unicas:
22     mails_con_w = 0 # contador
23     for frase in train_spam:
24         if w in frase:
25             mails_con_w += 1
26     print(f"Número de mails spam con la palabra {w}: {mails_con_w}")
27     spamicidad = (mails_con_w + 1) / (len(train_spam) + 2) # suavizado
28     print(f"Spamicidad de la palabra '{w}': {spamicidad}\n")
29     dict_spamicidad[w.lower()] = spamicidad
```

Para evitar el escenario en que una palabra aparezca cero veces en los datos train de spam pero aparezca en los datos train de ham o al contrario hay que aplicar un suavizado (*smoothing*) utilizando por ejemplo contadores que comiencen en 1 en vez de en cero. Así que añadimos 1 a cada contador de

cada palabra y añadimos 2 al denominador (el número de clases que tenemos) y así conseguimos dejar las probabilidades entre 0 y 1. En este ejemplo, la spamicidad de la palabra "send" es del 75% (pero al aplicar el suavizado se queda en un 66%). Con un límite del 0.5 para considerarla spam, ayudaría a considerar como spam un mail.

Ahora debemos hacer algo parecido y calcular la hamicidad de las palabras.

**ENTREGA 1:** completa el código y calcula la *hamicidad* de las palabras creando un diccionario Python donde cada clave sea una de las palabras que aparecen en los datos de mail ham. Si lo imprimes debe darte este resultado:

```
Hamicidad: {'your': 0.4, 'activity': 0.6, 'report': 0.4, 'benefits': 0.4, 'physical': 0.4,
'the': 0.4, 'importance': 0.4, 'vows': 0.4}
```

Ahora calculamos las probabilidades (a priori) de que un email sea spam  $P(S)$  (la cantidad de emails que son spam entre la cantidad de todos los emails) y la probabilidad de que un email sea ham  $P(H)$  (la cantidad de emails que son ham dividida entre el número total de emails):

```
51 # Probabilidades de spam y ham
52 prob_spam = len(train_spam) / (len(train_spam)+(len(train_ham)))
53 print("P(S)=", prob_spam)
54 prob_ham = len(train_ham) / (len(train_spam) + (len(train_ham)))
55 print("P(H)=", prob_ham)
```

Ahora vamos a preparar la lista de las distintas palabras de los datos de test (tokenizar los mails):

```
63 # Dividir los mail en palabras únicas
64 distintas_palabras_como_frase_test = []
65 for frase in tests:
66     frase_como_lista = frase.split()
67     sentencia = []
68     for w in frase_como_lista:
69         sentencia.append(w)
70     distintas_palabras_como_frase_test.append(sentencia)
71 print(distintas_palabras_como_frase_test)
72 test_spam_tokenizado = [distintas_palabras_como_frase_test[0],
73                         distintas_palabras_como_frase_test[1]]
74 test_ham_tokenizado = [distintas_palabras_como_frase_test[2],
75                        distintas_palabras_como_frase_test[3]]
76 print("Test spam tokenizado", test_spam_tokenizado)
77 print("Test ham tokenizado", test_ham_tokenizado)
```

**Ignorar las palabras que no aparecen etiquetas en los datos de train.**

El motivo de hacer esto es que de esas palabras no tenemos información sobre su spamicidad o su hamicidad, así que no tenerlas en cuenta no afectará al resultado que obtenemos ni al entrenar ni al predecir.

```
78 # Eliminar palabras de test sin datos en los datos de train
79 spam_test_reducido = []
80 for frase in test_spam_tokenizado:
81     palabras_ = []
82     for w in frase:
83         if w in vocab_palabras_spam_unicas:
84             print(f"{'w}', ok")
85             palabras_.append(w)
86         elif w in vocab_palabras_ham_unicas:
87             print(f"{'w}', ok")
88             palabras_.append(w)
89         else:
90             print(f"{'w}', sin información en train como spam")
91     spam_test_reducido.append(palabras_)
92 print("Test de spam reducido:", spam_test_reducido)
```

**ENTREGA 2:** completa el código y haz lo mismo para los datos de test etiquetados como ham, eliminando las palabras que no estén en los datos de train como ham. Si lo imprimes, debe darte este resultado:

```
Test de ham reducido: [['benefits', 'our', 'account'], ['the', 'importance', 'physical',
'activity']]
```

## STEMMING (ELIMINAR ELEMENTOS POCO IMPORTANTES)

Si eliminamos las palabras que tienen poca influencia el clasificador se podrá centrar en aquellas más importantes para el trabajo que debe realizar. Para decidir que palabras tener en cuenta y cuales no, se suelen utilizar conjuntos de datos preparados con antelación de en este caso palabras que por ser muy comunes en el lenguaje, no deberían aportar excesiva información (artículos, pronombres, preposiciones, etc.).

```
107 # stemmed
108 test_spam_stemmed = []
109 poco_importantes = ['us', 'the', 'of', 'your'] # palabras no clave
110 for email in spam_test_reducido:
111     email_limpiado = []
112     for w in email:
113         if w in poco_importantes:
114             print(f'Eliminar "{w}"')
115         else:
116             email_limpiado.append(w)
117     test_spam_stemmed.append(email_limpiado)
118 print("Test spam stemmed:", test_spam_stemmed)
```

**ENTREGA 3:** completa el código y haz lo mismo para los datos de test y consigue `test_ham_stemmed`. Si lo imprimes, debe darte este resultado:

Eliminar "the"

Test ham stemmed: [['benefits', 'our', 'account'], ['importance', 'physical', 'activity']]

## CALCULAR BAYES

Ahora vamos a aplicar la regla de Bayes para calcular la probabilidad de que un mail sea spam usando las palabras que contiene. Ya hemos calculado las probabilidades y las condicionales. Obtener la probabilidad consistirá en multiplicar todas esas probabilidades entre si. Definimos 2 métodos:

```
130 def multiplica(lista) : # multiplica las probs de las palabras de la lista
131     total_prob = 1
132     for i in lista:
133         total_prob = total_prob * i
134     return total_prob
135
136 def Bayes(email):
137     probs = []
138     for w in email: # para cada palabra w del mail
139         PS = prob_spam
140         print('P(S)=', PS)
141         try:
142             p_ws = dict_spamicidad[w]
143             print(f'P({w}|spam)= {p_ws}')
144         except KeyError:
145             p_ws = 1 / (len(train_spam) + 2) # Aplicar suavizado a palabras
146             print(f'P({w}|spam)= {p_ws}')
147         PH = prob_ham
148         print('P(H)=', PH)
149         try:
150             p_wh = dict_hamicidad[w]
151             print(f'P({w}|ham)=', p_wh)
152         except KeyError:
153             p_wh = 1 / (len(train_ham) + 2) # Aplicar smoothing
154             print(f'P({w}|ham)= {p_wh}')
155         p_spam_BAYES = (p_ws * PS) / ((p_ws * PS) + (p_wh * PH))
156         print(f'Usando Bayes P({w}|spam)= {p_spam_BAYES}')
157         probs.append(p_spam_BAYES)
158     print(f"Probabilidades de todas las palabras del mail: {probs}")
159     clasificacion = multiplica(probs)
160     if clasificacion >= 0.5:
161         print(f'email es SPAM: P(spam)={clasificacion * 100:.4f}%')
162     else:
163         print(f'email es HAM: P(spam)={clasificacion * 100:.4f}%')
164     return clasificacion
```

Esta aproximación explica porqué estos clasificadores tienen la palabra '**Naïve**' porque no tienen en cuenta las relaciones de cualquier palabra con la siguiente o la anterior, no tienen en cuenta el



contexto, las usan como si fuesen elementos independientes en los mensajes cuando en realidad el orden en que aparecen si es importante en un mensaje de cualquier idioma. Y ahora utilizamos estas funciones con los datos de test:

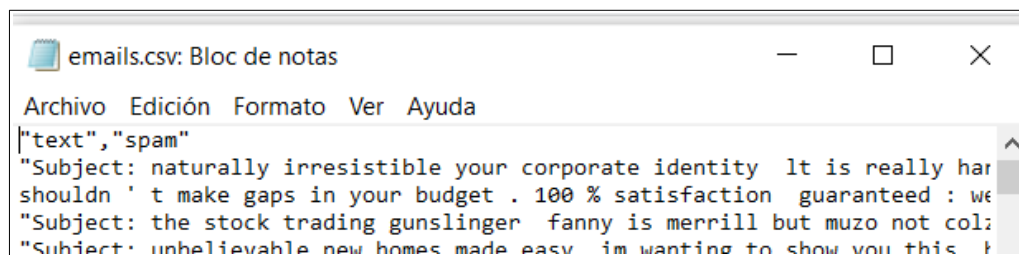
```
166 for email in test_spam_stemmed:
167     print(f"==== Test mail SPAM {email} ====")
168     Bayes(email)
169 for email in test_ham_stemmed:
170     print(f"==== Test mail HAM {email} ====")
171     Bayes(email)
```

**ENTREGA 4:** Ejecuta el código y comprueba qué % de emails clasifica bien.

## ACTIVIDAD 2: VECTORIZAR TEXTO DE MANERA AUTOMÁTICA.

En la actividad 1 hemos implementado un clasificador usando el teorema de Bayes sin utilizar ninguna librería. Una de las tareas que hemos tenido que hacer para trabajar con texto es vectorizar las palabras para contarlas. En realidad, las librerías de ML ya nos dan muchas herramientas para estos procesamientos. Crea una carpeta llamada *actividad2* y crea el fichero *a2.py* en ella.

En esta actividad vas a implementar un modelo clasificador naive Bayes usando *scikit-learn*. Ahora vamos a trabajar con datos más reales. Vas a usar el dataset del fichero *emails.csv* que tiene dos características. La primera es el texto de los emails y la segunda es si es spam (1) o ham (0). Hay más emails ham que spam. Copia el fichero a la carpeta, su contenido es similar a este:



```
emails.csv: Bloc de notas
Archivo Edición Formato Ver Ayuda
"text", "spam"
"Subject: naturally irresistible your corporate identity It is really hard
shouldn't make gaps in your budget . 100 % satisfaction guaranteed : we
"Subject: the stock trading gunslinger fanny is merrill but muzzo not col
"Subject: unbelievable new homes made easy im wanting to show you this t
```

**¿Qué hace *CountVectorizer*?** Convierte una colección de documentos de texto en una matriz de contadores de tokens. Esta implementación genera una representación eficiente y dispersa de los contadores usando *scipy.sparse.csr\_matrix*. Si no indicas un diccionario a-priori y no usas un analizador que haga algún tipo de selección de características, entonces el número de características será igual al tamaño del vocabulario encontrado al analizar los datos. **Copia el código, lo ejecutas y responde a las preguntas.**

```
1 # -*- coding: utf-8 -*-
2 import pandas as pd
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5
6 # Cargar datos
7 spam_df = pd.read_csv('./emails.csv')
8 print( spam_df.head(10) )
9 print( spam_df.info() )
10
11 #Visualizar datos
12 ham = spam_df[spam_df['spam'] == 0]
13 spam = spam_df[spam_df['spam'] == 1]
14 print(f'Porcentaje de Spam = {len(spam)/len(spam_df) * 100:.4f}%')
15 print(f'Porcentaje de Ham = {len(ham)/len(spam_df)*100:.4f}%')
16 sns.countplot(x='spam', data= spam_df, label='Spam vs Ham')
17 plt.show()
```

**ENTREGA 5:** ¿Cuántos emails contiene el dataset? ¿Qué porcentaje de ellos son spam?

```

19 # Aplicar la vectorización automática
20 from sklearn.feature_extraction.text import CountVectorizer
21 c_v = CountVectorizer()
22 spamham_c_v = c_v.fit_transform( spam_df['text'] )
23 print("CountVectorizer: Nombres de características:\n", c_v.get_feature_names_out() )
24 print("CountVectorizer: diseño", spamham_c_v.shape)
25 label = spam_df['spam']
26 X = spamham_c_v
27 y = label
28 print("Dimensiones de X", X.shape)

```

**ENTREGA 6:** El problema de no eliminar palabras comunes es que la cantidad de palabras o tokens que debes procesar se dispara mucho y cada una de ellas se considera una característica de los datos de entrenamiento ¿Cuántas distintas palabras ha encontrado el vectorizador?

```

30 # Dividir en train y test
31 from sklearn.model_selection import train_test_split
32 X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.2)
33 # Crear el modelo naive Bayes de tipo Multinomial
34 from sklearn.naive_bayes import MultinomialNB
35 nbc = MultinomialNB()
36 nbc.fit(X_train, y_train)
37
38 # Imprimir resultados de validación: mapa de calor y matriz de confusion.
39 from sklearn.metrics import classification_report, confusion_matrix
40 y_pred_train = nbc.predict(X_train)
41 cm = confusion_matrix(y_train, y_pred_train)
42 sns.heatmap(cm, annot=True)
43 plt.show()
44 y_pred_test = nbc.predict(X_test)
45 cm = confusion_matrix(y_test, y_pred_test)
46 sns.heatmap(cm, annot=True)
47 print(classification_report(y_test, y_pred_test))
48 plt.show()

```

**ENTREGA 7:** En el problema de clasificar spam que es peor, ¿Tener un falso positivo (decir que un mail es spam cuando no lo es) o un falso negativo (decir que un mail es ham cuando en realidad es spam)?

- Muestra la matriz de confusión del entrenamiento y del test.
- ¿Parece que generaliza bien este modelo? (argumenta la respuesta)
- Para la del test indica donde comete más fallos cuantitativamente (5 errores, 4 errores, ...) y porcentualmente (fallo 5 veces en decir que es spam de 100 emails spam que tengo, es el 5% de error).
- Pasa captura del informe de la clasificación, e indica si los valores de las métricas darían por bueno el modelo según lo que hayas respondido en la primera cuestión.

## ACTIVIDAD 3: DETECTANDO SPAM EN MENSAJES SMS.

### CARGAR Y EXPLORAR LOS DATOS

Crea una carpeta llamada **actividad3** y crea el fichero **a3.py** en ella. Abrimos el fichero **SMSSpamCollection.csv** (que habrás copiado a la carpeta) con la función de pandas **read\_csv()**:

- **sep='\\t'** porque los datos están separados con el carácter tabulador.
- **header=None** porque el dataset no tiene una primera fila explicativa o de cabecera.
- **names=['Label', 'SMS']** para dar nombre a las características del dataset.

```

1 # -*- coding: utf-8 -*-
2 import pandas as pd
3
4 sms_spam = pd.read_csv('./SMSSpamCollection.csv', sep='\\t', header=None,
5                        names=['Label', 'SMS'])
6 print("== Dimensiones: ", sms_spam.shape)
7 print("== Primeros 5 ejemplos:\\n", sms_spam.head() )
8 print("== Información de las columnas:")
9 print(sms_spam.info())

```

Ahora vamos a averiguar qué porcentaje de ejemplos son spam y qué porcentaje son ham. Si tenemos más ham que spam se parecerá a una situación de la vida real porque la mayoría de los mensajes afortunadamente no son spam.

```
10
11 print("== Porcentajes de spam y ham:")
12 print( sms_spam['Label'].value_counts(normalize=True) )
13
```

### PREPARAR LOS DATOS DE ENTRENAMIENTO Y TEST

Usaremos el 80% de los ejemplos para entrenar y el 20% restante para comprobar el modelo. Antes de realizar la división redistribuimos los ejemplos de manera aleatoria para asegurarnos que los mensajes spam quedan repartidos por todo el dataset de manera aleatoria.

Ahora mostramos si los porcentajes de spam quedan parecidos a los valores que había antes de hacer la división. Si en los datos de entrenamiento se cumple, también se cumplirá en los de test.

```
14 # Dividir en train + test
15 datos = sms_spam.sample(frac=1, random_state=1) # Aleatorizar dataset
16 indices = round( len(datos) * 0.8 ) # Calcula índices división
17 train = datos[:indices].reset_index(drop=True)
18 test = datos[indices:].reset_index(drop=True)
19 print("== Dimensiones de train:", train.shape)
20 print("== Dimensiones de test:", test.shape)
21 print("== Porcentajes de spam en datos train:")
22 print( train['Label'].value_counts(normalize=True) )
23 print("== Porcentajes de spam en datos test:")
24 print( test['Label'].value_counts(normalize=True) )
```

### LIMPIEZA DE DATOS

Cuando al algoritmo Naive Bayes Multinomial se le presente un nuevo mensaje lo clasificará basándose en estas dos ecuaciones, donde " $w_1$ " es su primera palabra y  $w_1, w_2, \dots, w_n$  son todas las palabras que forman el mensaje. Si  $P(\text{Spam} | w_1, w_2, \dots, w_n)$  es mayor que  $P(\text{Ham} | w_1, w_2, \dots, w_n)$ , entonces el mensaje será spam.

$$P(\text{Spam} | w_1, w_2, \dots, w_n) \propto P(\text{Spam}) \cdot \prod_{i=1}^n P(w_i | \text{Spam})$$

$$P(\text{Ham} | w_1, w_2, \dots, w_n) \propto P(\text{Ham}) \cdot \prod_{i=1}^n P(w_i | \text{Ham})$$

Para calcular  $P(w_i | \text{Spam})$  y  $P(w_i | \text{Ham})$ , necesitamos usar ecuaciones separadas:

$$P(w_i | \text{Spam}) = \frac{N_{w_i | \text{Spam}} + \alpha}{N_{\text{Spam}} + \alpha \cdot N_{\text{Vocabulary}}}$$

$$P(w_i | \text{Ham}) = \frac{N_{w_i | \text{Ham}} + \alpha}{N_{\text{Ham}} + \alpha \cdot N_{\text{Vocabulary}}}$$

Repasamos algunos términos de estas ecuaciones:

- $N_{w_i | \text{Spam}}$  = el número de veces que la palabra  $w_i$  aparece en mensajes etiquetados como Spam.
- $N_{w_i | \text{Ham}}$  = el número de veces que la palabra  $w_i$  aparece en mensajes etiquetados como Ham.
- $N_{\text{Spam}}$  = el número total de palabras diferentes en los mensajes etiquetados como Spam.
- $N_{\text{Ham}}$  = el número total de palabras diferentes en los mensajes etiquetados como Ham.
- $N_{\text{Vocabulary}}$  = el número total de palabras diferentes en todos los mensajes.
- $\alpha = 1$  es un parámetro de suavizado.

Para calcular todas estas probabilidades, primero necesitamos realizar un poco de limpieza de datos a los datos de entrenamiento y dejarlos en un formato que nos permita extraer fácilmente toda la información que necesitamos para realizar esos cálculos. Ahora mismo los datos de entrenamiento y test tienen un formato parecido a este:

	Label	SMS
0	spam	SECRET PRIZE! CLAIM SECRET PRIZE NOW!!
1	ham	Coming to my secret party?
2	spam	Winner! Claim secret prize now!

Y para facilitar los cálculos los vamos a transformar a este otro formato:

	Label	secret	prize	claim	now	coming	to	my	party	winner
0	spam	2	2	1	1	0	0	0	0	0
1	ham	1	0	0	0	1	1	1	1	0
2	spam	1	1	1	1	0	0	0	0	1

Observa que:

- La columna SMS es reemplazada por una serie de nuevas columnas que representan palabras no repetidas del vocabulario (el conjunto de todas las palabras que contienen los mensajes).
- Cada fila describe un único mensaje. La primera fila contiene los valores: **spam, 2, 2, 1, 1, 0, 0, 0, 0, 0** que nos informa de que:
  - El mensaje es spam.
  - La palabra "secret" aparece dos veces dentro del mensaje.
  - La palabra "prize" aparece dos veces dentro del mensaje.
  - La palabra "claim" aparece una vez dentro del mensaje.
  - La palabra "now" aparece una vez dentro del mensaje.
  - Las palabras "coming", "to", "my", "party" y "winner" no aparecen en el mensaje (0 veces).
- Todas las palabras del vocabulario están en minúsculas, así que "SECRET" y "secret" se consideran la misma palabra.
- El orden de las palabras en las frases originales se pierde.
- Los signos de puntuación no se tienen en cuenta (desaparecen puntos, comas, puntos y coma, dos puntos, exclamaciones, interrogaciones, paréntesis, etc.).

Lo primero que vamos a realizar es eliminar los signos de puntuación de los mensajes y transformar todas las letras a minúsculas.

```

26 # Limpieza de datos
27 train['SMS'] = train['SMS'].str.replace('\W', ' ') # Elimina signos puntuación
28 train['SMS'] = train['SMS'].str.lower()           # Convierte a minúsculas

```

**ENTREGA 8:** Modifica este trozo de código y añade dos sentencias para que muestre las dos primeras filas de `train` antes y después de aplicar la limpieza de datos como se ve en la figura de abajo para comprobar que efectivamente eliminas los signos de puntuación y conviertes a minúsculas. Si no consigues hacerlo intenta algunas de estas modificaciones:

- Importa `re` (expresiones regulares) y `string` y sustituye la línea 27 por esta: `train['SMS'] = re.sub('[%s]' % re.escape(string.punctuation), ' ', train['SMS'].str)`
- Importa `string` y sustituye la línea 27 por esta: `train['SMS'] = train['SMS'].str.replace('{}'.format(string.punctuation), ' ', regex=True)`
- A la función `replace()` de la línea 27 le añades el parámetro: `regex=True`



```

== 2 Primeras filas antes de la limpieza:
Label SMS
0 ham Yep, by the pretty sculpture
1 ham Yes, princess. Are you going to make me moan?
== 2 Primeras filas después de la limpieza:
Label SMS
0 ham yep by the pretty sculpture
1 ham yes princess are you going to make me moan

```

## CREAR EL VOCABULARIO Y TRANSFORMAR LOS DATOS

El vocabulario en este contexto significa la lista de palabras únicas que aparecen en los mensajes de los emails de train.

- Transformamos cada mensaje de la columna SMS en una lista de palabras dividiendo el texto con el método `Series.str.split()`.
- Inicializamos una lista vacía llamada **vocabulario**.
- Iteramos o recorremos las palabras de cada columna **SMS** transformada.
  - Usando un bucle anidado, iteramos por cada mensaje de la columna SMS y añadimos cada palabra a la lista del vocabulario.
- Transformamos la lista vocabulario en un conjunto con la función `set()`. Esto hace que desaparezcan las palabras repetidas porque un conjunto no permite tener elementos repetidos.
- Transformamos el set vocabulario en una lista usando la función `list()`.

```

36 # Crear el vocabulario
37 train['SMS'] = train['SMS'].str.split()
38 vocabulario = []
39 for sms in train['SMS']:
40     for palabra in sms:
41         vocabulario.append(palabra)
42 vocabulario = list( set(vocabulario) )
43 print(f"== Hay {len(vocabulario)} palabras distintas en los mensajes de train.")

```

Ahora usamos el vocabulario que acabamos de crear para conseguir la transformación de los datos que muestra la figura de abajo:

	Label	SMS
0	spam	SECRET PRIZE! CLAIM SECRET PRIZE NOW!!
1	ham	Coming to my secret party?
2	spam	Winner! Claim secret prize now!

↓

	Label	secret	prize	claim	now	coming	to	my	party	winner
0	spam	2	2	1	1	0	0	0	0	0
1	ham	1	0	0	0	1	1	1	1	0
2	spam	1	1	1	1	0	0	0	0	1

Se puede decir que tenemos que crear un nuevo **DataFrame**. Por ejemplo, para generar la tabla de abajo de la figura anterior, tendríamos que partir de cada palabra y una lista con las apariciones de la palabra en cada mensaje del dataset. Para ello creamos un diccionario y luego el **DataFrame**:

```

palabra_contadores_por_sms = {'secret': [2,1,1],
                              'prize': [2,0,1],
                              'claim': [1,0,1],
                              'now': [1,0,1],

```

```

        'coming': [0,1,0],
        'to': [0,1,0],
        'my': [0,1,0],
        'party': [0,1,0],
        'winner': [0,0,1]
    }
    palabras = pd.DataFrame(palabras_contadores_por_sms)
    print( palabras.head() )

```

Para crear el diccionario podemos usar los datos train de esta forma:

- Inicializamos el diccionario de nombre **palabra\_contadores\_por\_sms**, donde cada clave es una de las palabras únicas del vocabulario y cada valor es una lista con tantos elementos como ejemplos tiene los datos de train y para cada uno el contador está inicialmente a 0.
- El código `[0] * 5` genera la salida `[0, 0, 0, 0, 0]`. Así que el código `[0] * len(train['SMS'])` genera la lista con la longitud de `train['SMS']`.
- Iteramos sobre `train['SMS']` usando el método `enumerate()` para obtener tanto el índice como el mensaje SMS (índice y sms).
- Usando un bucle anidado, iteramos sobre el sms (que es una lista de palabras).
- Incrementamos en 1 el valor de `palabra_contadores_por_sms[palabra][indice]`.

```

45 palabra_contadores_por_sms = {p: [0] * len(train['SMS']) for p in vocabulario}
46 for idx, sms in enumerate(train['SMS']):
47     for p in sms:
48         palabra_contadores_por_sms[p][idx] += 1
49 palabras = pd.DataFrame(palabra_contadores_por_sms)
50 print(palabras.head())

```

```

   cutter  iraq  whatever  paragon  shrink  ...  sign  500  dogg  slap  acc
0      0     0         0         0         0  ...    0    0     0     0     0
1      0     0         0         0         0  ...    0    0     0     0     0
2      0     0         0         0         0  ...    0    0     0     0     0
3      0     0         0         0         0  ...    0    0     0     0     0
4      0     0         0         0         0  ...    1    0     0     0     0

[5 rows x 7783 columns]

```

La columna **Label** ha desaparecido, así que usaremos la función de Pandas `pd.concat()` para concatenar al **DataFrame** que acabamos de construir con el que contiene los datos de train. De esta forma, tendremos tanto la columna **Label** como la columna **SMS** además de los contadores de cada palabra.

```

51 train = pd.concat([train, palabras], axis=1)
52 print( train.head() )

```

```

   Label  SMS  ...  slap  acc
0  ham  [yep, by, the, pretty, sculpture]  ...    0    0
1  ham  [yes, princess, are, you, going, to, make, me,...  ...    0    0
2  ham  [welp, apparently, he, retired]  ...    0    0
3  ham  [havent]  ...    0    0
4  ham  [i, forgot, 2, ask, ü, all, smth, there, s, a,...  ...    0    0

[5 rows x 7785 columns]

```

### CALCULAR VALORES DE LA FÓRMULA

Ahora vamos a configurar el clasificador. El algoritmo Naive Bayes multinomial necesita calcular las dos fórmulas que calculan la probabilidad de cada clase para clasificar un nuevo mensaje:

$$P(\text{Spam} | w_1, w_2, \dots, w_n) \propto P(\text{Spam}) \cdot \prod_{i=1}^n P(w_i | \text{Spam})$$

$$P(\text{Ham} | w_1, w_2, \dots, w_n) \propto P(\text{Ham}) \cdot \prod_{i=1}^n P(w_i | \text{Ham})$$

También hay que calcular  $P(w_i|Spam)$  y  $P(w_i|Ham)$  dentro de las fórmulas anteriores y para ello usaremos estas ecuaciones:

$$P(w_i|Spam) = \frac{N_{w_i|Spam} + \alpha}{N_{Spam} + \alpha \cdot N_{Vocabulary}}$$

$$P(w_i|Ham) = \frac{N_{w_i|Ham} + \alpha}{N_{Ham} + \alpha \cdot N_{Vocabulary}}$$

Algunos de estas 4 ecuaciones tienen los mismos valores independientemente del mensaje a clasificar, es decir, no dependen del mensaje, así que podemos calcularlos una sola vez y reutilizarlos siempre:

- $P(Spam)$  y  $P(Ham)$
- $N_{Spam}$ ,  $N_{Ham}$ ,  $N_{Vocabulary}$

Recordamos que:

- $N_{Spam}$  es el número de palabras de todos los mensajes spam. No es el número de mensajes spam ni es el número de diferentes palabras de los mensajes spam.
- $N_{Ham}$  es el número de palabras de todos los mensajes que no son spam.
- Usamos el suavizado de Laplace ( $\alpha = 1$ ).

```
54 # Calcular el modelo
55 sms_spam = train[train['Label'] == 'spam']
56 sms_ham = train[train['Label'] == 'ham']
57 p_spam = len(sms_spam) / len(train)
58 p_ham = len(sms_ham) / len(train)
59 n_spam = sms_spam['SMS'].apply(len)
60 n_spam = n_spam.sum()
61 n_ham = sms_ham['SMS'].apply(len)
62 n_ham = n_ham.sum()
63 n_vocabulary = len(vocabulario)
64 alfa = 1
```

Con las constantes ya calculadas, pasamos a calcular  $P(w_i|Spam)$  y  $P(w_i|Ham)$  que pueden cambiar según las palabras que tenga cada email concreto. Por ejemplo,  $P("secret"/Spam)$  tendrá cierto valor de probabilidad, mientras que  $P("cousin"/Spam)$  o  $P("Lovely"/Spam)$  tendrán otros valores. Los parámetros se calculan con estas dos ecuaciones:

$$P(w_i|Spam) = \frac{N_{w_i|Spam} + \alpha}{N_{Spam} + \alpha \cdot N_{Vocabulary}}$$

$$P(w_i|Ham) = \frac{N_{w_i|Ham} + \alpha}{N_{Ham} + \alpha \cdot N_{Vocabulary}}$$

```
65 # Inicializar y calcular los parámetros
66 param_spam = {p:0 for p in vocabulario}
67 param_ham = {p:0 for p in vocabulario}
68 for palabra in vocabulario:
69     n_wi_spam = sms_spam[palabra].sum()
70     p_wi_spam = (n_wi_spam + alfa) / (n_spam + alfa * n_vocabulary)
71     param_spam[palabra] = p_wi_spam
72     n_wi_ham = sms_ham[palabra].sum()
73     p_wi_ham = (n_wi_ham + alfa) / (n_ham + alfa * n_vocabulary)
74     param_ham[palabra] = p_wi_ham
```

## CLASIFICAR MENSAJES

Usamos una función para representar el clasificador:

- Recibe como entrada un nuevo mail con las letras ( $w_1, w_2, \dots, w_n$ ).
- Calcula  $P(Spam|w_1, w_2, \dots, w_n)$  y  $P(Ham|w_1, w_2, \dots, w_n)$ .

- Compara los valores de  $P(\text{Spam}|w_1, w_2, \dots, w_n)$  y  $P(\text{Ham}|w_1, w_2, \dots, w_n)$ , y:
  - Si  $P(\text{Ham}|w_1, w_2, \dots, w_n) > P(\text{Spam}|w_1, w_2, \dots, w_n)$ , entonces el mensaje se clasifica como ham.
  - Si  $P(\text{Ham}|w_1, w_2, \dots, w_n) < P(\text{Spam}|w_1, w_2, \dots, w_n)$ , entonces el mensaje se clasifica como spam.
  - Si  $P(\text{Ham}|w_1, w_2, \dots, w_n) = P(\text{Spam}|w_1, w_2, \dots, w_n)$ , entonces se pregunta al humano.

Observa que los nuevos mensajes pueden contener palabras que no están en el vocabulario del modelo. Estas palabras simplemente se ignoran porque al modelo no le aportan información para tomar su decisión.

La función que predice se va a llamar `clasifica()` y tiene estas características:

- Recibe un mensaje de mail (un texto) como parámetro.
- Realiza un poco de limpieza:
  - Eliminar signos de puntuación con la función `re.sub()`.
  - Pasar a minúsculas las letras con el método `str.lower()`.
  - Divide el texto en palabras separándolo por espacios en blanco usando el método `str.split()`.
- Calcula `p_spam_mensaje` y `p_ham_mensaje`.
- Compara los valores obtenidos e imprime la clase predicha.

```

78 def clasifica(mensaje):
79     mensaje = re.sub('\W', ' ', mensaje)
80     mensaje = mensaje.lower().split()
81     p_spam_mensaje = p_spam
82     p_ham_mensaje = p_ham
83     for palabra in mensaje:
84         if palabra in param_spam:
85             p_spam_mensaje *= param_spam[palabra]
86
87         if palabra in param_ham:
88             p_ham_mensaje *= param_ham[palabra]
89     print('P(Spam|mensaje):', p_spam_mensaje)
90     print('P(Ham|mensaje):', p_ham_mensaje)
91     if p_ham_mensaje > p_spam_mensaje:
92         print('Label: Ham')
93     elif p_ham_mensaje < p_spam_mensaje:
94         print('Label: Spam')
95     else:
96         print('Igual de probable, un humano debe decidir!')
97

```

Y ahora la probamos con un mensaje que sea spam y otro que sea ham:

```
clasifica('WINNER!! This is the secret code to unlock the money: C3421.')
```

```

P(Spam|mensaje): 1.3481290211300841e-25
P(Ham|mensaje): 1.9368049028589875e-27
Label: Spam

```

```
clasifica("Sounds good, Tom, then see u there")
```

```

P(Spam|mensaje): 2.4372375665888117e-25
P(Ham|mensaje): 3.687530435009238e-21
Label: Ham

```

## MEDIR LA EFICIENCIA CON ACCURACY

Los dos resultados anteriores parecen prometedores pero vamos a usar los datos de test que tienen 1114 mensajes etiquetados para comprobarlo. Escribimos una función muy parecida a `clasifica()` y llamada `clasifica_test()` pero que no imprima nada, simplemente que devuelva la etiqueta predicha: 'ham' o 'spam' o 'Igual de probable'.

**ENTREGA 9:** Completa la función `clasifica_test()` y usándola con los mensajes de test calcula el accuracy. Recuerda que `accuracy = mensajes bien clasificados / total de mensajes`.

- Usa estas sentencias y pasa captura del resultado:

```
test['prediccion'] = test['SMS'].apply(clasifica_test)
print("== Test con predicciones realizadas:\n", test.head())
```

- Calcula accuracy e indica cuál es con estas sentencias:

```
121 correctas = 0
122 total = test.shape[0]
123 for fila in test.iterrows():
124     fila = fila[1]
125     if fila['Label'] == fila['prediccion']:
126         correctas += 1
127 print(f'== Correctas {correctas} de {total} Accuracy: {correctas/total:.4f}')
```

## ACTIVIDAD 4: ANÁLISIS DE SENTIMIENTOS EN TEXTO.

### OBJETIVOS DE LA ACTIVIDAD:

- Buscar y adaptar datos.
- Introducirnos en el uso de librerías de procesamiento de texto.
- Utilizar varios algoritmos de clasificación además de Naïve Bayes.

Una cadena de restaurantes quiere analizar las reseñas que hacen sus clientes a los locales que visitan. Cada cliente puede valorar su experiencia indicando una puntuación y añadiendo comentarios. La aplicación web añade una etiqueta a cada comentario como '*positivo*' o '*negativo*' basándonos en la puntuación que marcan en la web de cada local. Y la empresa quiere implementar un sistema que procese el texto de las reseñas y las clasifique como positivas o negativas sin necesidad de que el cliente tenga que valorar su experiencia cuantitativamente. Para ello debemos:

1. Calcular la probabilidad de cada palabra en un texto y filtrar palabras que tengan una probabilidad menor de cierto límite, es decir, eliminar palabras que no sean muy influyentes como palabras muy comunes.
2. Para cada palabra del diccionario (las que aparecen en los comentarios) que se crea, calcular la probabilidad de que pertenezca a respuestas positivas y la probabilidad de que se encuentre en respuestas negativas. Luego, calcular la probabilidad usando un clasificador naive Bayes.
3. Predecir usando probabilidades condicionales.

Los datos están almacenados en el fichero "reseñas\_restaurantes.csv". Tienen unos 51 comentarios cada uno con columnas *puntuación*, *comentario*, *sentimiento* donde la etiqueta es la columna *sentimiento*.

**ENTREGA 10:** Busca en las web de establecimientos que conozcas o visites (o aunque no los visites algún restaurante o cervecería de tu zona) y seleccionas 10 reseñas de clientes (al menos 5 de ellas favorables y el resto hasta llegar a 10 desfavorables). Les das el mismo formato que el usado en las que te paso y las subes en el fichero "<3\_primeras\_letras\_tu\_nombre>\_<3\_primeras\_letras\_tus\_apellidos>.txt" a una carpeta compartida en aules.

Añade también por el final tus reseñas al fichero compartido '*reseñas\_restaurantes.csv*' que será inicialmente una copia del anterior y que debería ir aumentando con las reseñas que vayamos añadiendo. Antes de comenzar el ejercicio, todos tus compañeros habrán hecho lo mismo, por tanto la cantidad de datos habrá aumentado un poco.

a) Importar datos y eliminar la primera columna (*puntuación*).

```
1 # -*- coding: utf-8 -*-
2 #=== apartado a)
3 import pandas as pd
4 df = pd.read_csv("reseñas_restaurantes.csv", sep=";")
5 print(df.info())
6 del(df['puntuación']) # Borramos puntuaciones
7 datos = [tuple(x) for x in df.values] # Transformar una lista de tuplas
```



b) Queremos normalizar los mensajes de texto. Vamos a pasar las letras a minúsculas, eliminar los signos de puntuación, eliminar las palabras con menos de 3 letras y eliminar las *stop-words* del castellano. Pasamos las palabras a su lema o forma raíz (**Lemmatización o stemización**) para eliminar variantes. En vez de hacerlo a mano usando expresiones regulares, podemos emplear librerías de manipulación de texto:

- **spacy**: una librería de software para procesamiento de lenguajes naturales, reconocimiento de nombres de entidades, análisis de redes, visualización de datos, análisis, visual analysis, análisis de contenidos, enriching y anotación desarrollado por *Matt Honnibal* en Python.
- **NLTK**: El kit de herramientas de lenguaje natural es un conjunto de bibliotecas y programas para el procesamiento del lenguaje natural simbólico y estadísticos para Python. NLTK incluye demostraciones gráficas y datos de muestra.

*Nota: Mi idea era mezclar en el ejercicio el uso de las dos, pero me han dado problemas de versiones, así que me quedo solamente con nltk.*

```

9  #=== Apartado b)
10 # Para instalar nltk puedes hacer: pip install nltk==3.9.1
11 # Si quieres elegir paquetes, corpus, datos individuales ejecuta interactiva:
12 #     import nltk
13 #     nltk.download()
14 # Y se abre página web donde eliges
15 import nltk
16 from tqdm import tqdm
17 from nltk.tokenize import word_tokenize # Tokeniza palabras
18 from nltk.corpus import stopwords      # stop-words, palabras comunes
19 nltk.download("punkt")                 # Es un tokenizador de nltk en inglés y español
20 nltk.download('punkt_tab')             # Lo pide
21
22 spanish_sw = set(stopwords.words('spanish'))
23
24 from nltk.stem.porter import PorterStemmer # Stemmer para eliminar variedad
25 porter = PorterStemmer()
26
27 def normaliza(mensajes):
28     """
29     Preprocesa los datos de texto
30     Entradas: mensajes con formato [texto, etiqueta]
31     Salidas: mensajes con formato [ (texto, label), (texto, label)... ]
32     """
33     for idx, msj in enumerate(tqdm(mensajes)):
34         tokens = word_tokenize(msj[0].lower(), language="spanish") # Paso a minú
35         filtrado = [palabra for palabra in tokens
36                     if (len(palabra) > 2
37                         and (not palabra in spanish_sw))]
38         stemmed = " ".join([porter.stem(p) for p in filtrado])
39         mensajes[idx] = (stemmed, msj[1])
40     return mensajes
41
42
43 X = normaliza(datos) # Normalizamos el texto de los comentarios

```

c) Dividir los datos en el 80% para train y comprobar si hay un % adecuado de cada clase en train y en test.

```

34 #=== Apartado c)
35 y = [y[1] for y in X]
36 n_positivos = y.count('positivo')
37 n_total = len(y)
38 r_positivos = n_positivos / n_total
39 r_negativos = 1 - r_positivos
40 print(f"\nReseñas en total {n_total} Positivos {(100.0 * r_positivos):.4f}%")
41 from sklearn.model_selection import train_test_split

```

**ENTREGA 11:** completa el siguiente código hasta conseguir salidas similares a las de las figuras que aparecen a continuación, pasa captura de la ejecución como esta:

```
Reseñas en total 51 Positivos 66.6667%
Reseñas para train: 40 positivos: 67.5000%
Reseñas para test: 11 positivos: 63.6364%
```

**d) Crear la bolsa de palabras.** Ahora crearemos un diccionario (no es un diccionario de Python) que contenga todas las diferentes palabras que aparecen en las reseñas de entrenamiento. Va a ser una colección de tuplas que contiene en la primera posición un booleano por cada palabra del diccionario indicando si la palabra aparece o no en la reseña y en la segunda posición la clase de la frase. Para hacerla creamos dos funciones que preparan los datos a como los necesita **nltk**:

```
63  #=== Apartado d)
64  def palabras_diferentes(datos):
65      """Devuelve una lista con todas las palabras únicas de las frases"""
66      todas = []
67      for (texto, sentimiento) in datos:
68          todas.extend(nltk.word_tokenize(texto, language="spanish"))
69      return list(set(todas))
70
71  diccionario = palabras_diferentes(X) # Creamos el diccionario de palabras
72
73  def extrae_caracteristicas(texto):
74      """Crea el conjunto de entrenamiento del clasificador
75      1: Para cada palabra del diccionario
76      3: Escribe {'contains(palabra)':True,...} si aparece palabra en texto
77      Escribe {'contains(palabra)':False,...} si no aparece
78      """
79      palabras_del_texto = set(texto.split())
80      caracteristicas = {}
81      for palabra in diccionario:
82          caracteristicas['contains(%)' % palabra] = (palabra in palabras_del_texto)
83      return caracteristicas
84
85  # Creamos las estructuras de datos para el clasificador con NLTK
86  X_train_nltk = nltk.classify.apply_features(extrae_caracteristicas, X_train)
87  X_test_nltk = nltk.classify.apply_features(extrae_caracteristicas, X_test)
```

**e) Con el diccionario creado podemos usar cualquier clasificador.** Vamos a crear y entrenar un naive Bayes, naive Bayes Multinomial, naive Bayes Bernoulli, Regresión logística y SVM (máquina de vector soporte).

```
85  # Creamos las estructuras de datos para el clasificador con NLTK
86  X_train_nltk = nltk.classify.apply_features(extrae_caracteristicas, X_train)
87  X_test_nltk = nltk.classify.apply_features(extrae_caracteristicas, X_test)
88
89  # Apartado e)
90  from sklearn.naive_bayes import MultinomialNB, BernoulliNB
91  from sklearn.linear_model import LogisticRegression
92  from sklearn.svm import SVC
93
94  print('Entrenando Naive Bayes')
95  nb = nltk.classify.NaiveBayesClassifier.train(X_train_nltk)
96  print('Entrenando Naive Bayes Multinomial')
97  nbm = nltk.classify.SklearnClassifier(MultinomialNB()).train(X_train_nltk)
98  print('Entrenando Bernoulli Naive Bayes')
99  nbb = nltk.classify.SklearnClassifier(BernoulliNB()).train(X_train_nltk)
100 print('Entrenando Regresión Logística')
101 rl = nltk.classify.SklearnClassifier(
102     LogisticRegression(solver='lbfgs', max_iter=1000)).train(X_train_nltk)
103 print('Entrenando Support Vector Machine')
104 svm = nltk.classify.SklearnClassifier(SVC(kernel='linear')).train(X_train_nltk)
105 clasificadores = {'Naive Bayes': nb,
106                  'Naive Bayes Multinomial': nbm,
107                  'Naive Bayes Bernoulli': nbb,
108                  'Regresión Logística': rl,
109                  'Support Vector Machine': svm}
```

f) Ahora vamos a medir el funcionamiento de todos ellos calculando sus métricas y vamos a presentarlas en forma tabular creando un *DataFrame* de Pandas para poder compararlas de manera sencilla.

```

111 #== Apartado f)
112 import warnings
113 warnings.filterwarnings("ignore")
114 from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
115
116 evaluacion = list()
117 for k, v in clasificadores.items():
118     print(f'Evaluado Modelo: {k}')
119     # Obtengo las predicciones
120     predicciones = [v.classify(texto[0]) for texto in X_test_nltk]
121     modelo = {}
122     modelo['nombre'] = k
123     modelo['accuracy'] = accuracy_score(y_true=y_test, y_pred=predicciones)
124     modelo['precision'] = precision_score(y_true=y_test, y_pred=predicciones, average='weighted')
125     modelo['recall'] = recall_score(y_true=y_test, y_pred=predicciones, average='weighted')
126     modelo['f1'] = f1_score(y_true=y_test, y_pred=predicciones, average='weighted')
127     evaluacion.append(modelo)
128
129 # Pasamos los resultados a un DataFrame para visualizarlos mejor
130 df = pd.DataFrame.from_dict(evaluacion)
131 df.set_index("nombre", inplace=True)
132 print(df)

```

**ENTREGA 12:** Crea el programa y lo ejecutas y mira las métricas de los diferentes clasificadores. Pasa una captura de pantalla de los resultados obtenidos

**ENTREGA 13:** Escribe una evaluación razonada de los resultados (cuál te quedarías, el motivo...).

**ENTREGA 14:** Repite el ejercicio pero ahora entrena los clasificadores con el fichero original, es decir, antes de que tú y tus compañeros hayan añadido nuevos datos (los datos originales serán las primeras 51 filas de datos). Muestra captura de pantalla de los resultados como en la entrega 17.

**ENTREGA 15:** Valora si al añadir más datos se mejora la eficiencia de los clasificadores, a cuáles les afecta más el aumento en la cantidad de datos, etc.

## ACTIVIDAD 5: NUEVO DETECTOR DE SPAM CON DATOS + REALISTAS.

Ya hemos visto 3 implementaciones diferentes y ahora vas a realizar otra, pero me voy a centrar en la preparación de los datos. Como ya hemos visto en la unidad y en esta práctica, los clasificadores naive Bayes se calculan de manera sencilla y funcionan bien en condiciones de fuerte independencia. En las actividades anteriores hemos trabajado con datos ya preparados. En esta actividad os propongo que entre todos creemos nuestro propio dataset. Nuestros objetivos en este ejercicio:

- Implementar una clase que represente un mail.
- Mejorar un dataset de partida.
- Usar modelos de tokenización de texto
- Minimizar los errores que comete el modelo
- Mejorar en lo posible el rendimiento cuando trabaje, alcanzar una accuracy mejor del 80%.

Cuando se realice el testeo, centraremos nuestra atención en la diferencia entre los falsos positivos y los falsos negativos. En un problema de detección de spam los falsos positivos (clasificar un mail como spam cuando en realidad no lo es) puede ser muy perjudicial para las comunicaciones de un negocio.

## DATASOURCE

Partimos de un dataset con 4327 mensajes en total, de los que 2949 son ham y 1378 son spam. Esto ya sería suficiente, pero lo vamos a ampliar aportando cada uno de nosotros 5 mails de spam y 5 mails de ham.

```

Return-Path: <jorgegomez@varsur.com>
Received: from maquina1 (line129.varsur.net [192.168.73.129]
by ns.compumedicina.net (8.9.3/8.9.3) with ESMTP id RAA20801
for<info@compumedicina.com>; Tue, 29 Aug 2000 17:08:21 -0400
From: "Jorge Gomez" <jorgegomez@varsur.com>
To: <info@compumedicina.com>
Subject: Importante
Date: Tue, 29 Aug 2000 18:08:21 -0300
Message-ID: <NEBBIKIBELIEKKDCNGHGIEFBCAAA.jorgegomez@varsur.com>
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary="====_NextPart_000_0000_01C011DF.871D08A0"
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 5.00.2919.6700
X-MimeOLE: Produced By Microsoft MimeOLE V5.00.2919.6700
X-UIDL: 102a4a2c4e95ed6dd5ac92e9f6e1b072

```

El formato de los ficheros **.eml** está definido en la **RFC822** y la información sobre los formatos estándar más recientes de un mail (por ejemplo **MIME**, *Multipurpose Internet Mail Extensions*) se pueden encontrar en la **RFC2045-2049**.

Un mail (por si lo desconoces) organiza la información con cierta estructura, lo que se conoce como formato de los mensajes. Básicamente tiene 3 grandes componentes: cabecera, cuerpo y ficheros adjuntos. Aunque dicho así está muy simplificado.

En el dataset todas las cabeceras de los mensajes están completas, aunque algunas direcciones están ofuscadas (ocultadas con la intención de mantener el anonimato de emisarios y receptores o cualquier información de tipo personal que permita su identificación).

Al proceso de asegurar el anonimato de las personas cuya información sensible pueda ser usada de manera pública se le conoce como anonimización. Por este mismo motivo, los nombres de algunos hosts se han remplazado por **"csmining.org"** (que tiene un registro **MX** válido en el servicio **DNS**) y que en la mayoría de casos se asocia con **'hibody.csming.org'**.

En la carpeta **TRAINING** se guardan estos ficheros, uno para cada mail, con el nombre **TRAIN\_XXXXX.eml** donde las **X** representan un dígito de 0 a 9 y que juntas forman un número de 5 cifras que identifican cada mail.

Nombre	Fecha	Descripción	Tamaño
TRAIN_04321.eml	02/06/2021 7:14	Mensaje de correo...	8 KB
TRAIN_04322.eml	02/06/2021 7:14	Mensaje de correo...	9 KB
TRAIN_04323.eml	02/06/2021 7:14	Mensaje de correo...	4 KB
TRAIN_04324.eml	02/06/2021 7:14	Mensaje de correo...	1 KB
TRAIN_04325.eml	02/06/2021 7:14	Mensaje de correo...	7 KB
TRAIN_04326.eml	05/09/2024 18:47	Mensaje de correo...	7 KB

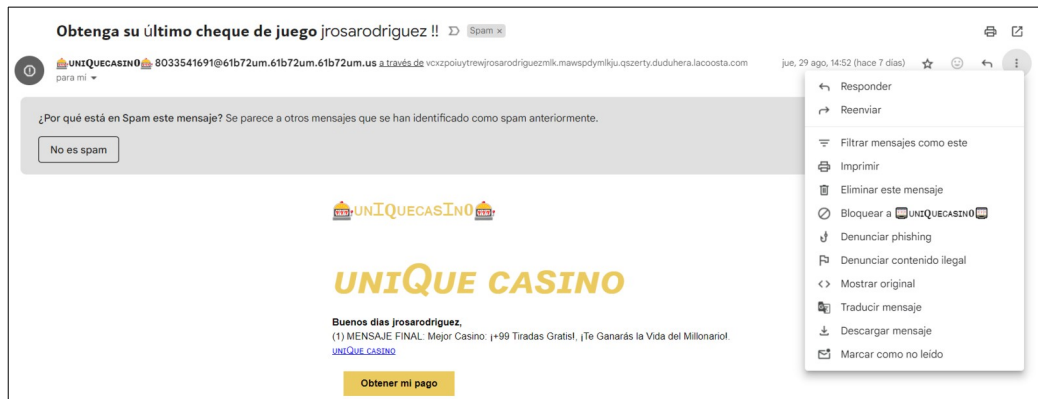
¿Cómo se saben cuales son ham y cuales spam? Bueno, hay otro fichero de texto llamado **SPAMTrain.label** que tiene una primera columna con 0 si es *spam* o un 1 en caso de que sea *ham* (una columna *label* que usaremos para aprender) y al lado el nombre del fichero mail que etiqueta.

Nombre	Fecha	Propiedades	Tamaño
TRAINING	05/09/2024 18:48	Carpeta de archivos	
SPAMTrain.label	02/06/2021 7:14	Property List	77 KB

Label	Nombre del fichero
0	TRAIN_00000.eml
0	TRAIN_00001.eml
1	TRAIN_00002.eml
0	TRAIN_00003.eml
0	TRAIN_00004.eml

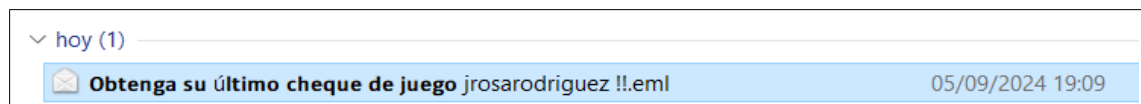
Ve a una de tus cuentas personales de mail. Busca 5 correos que estén etiquetados como spam por tu programa de correo. En la figura 7 ves uno de mi cuenta de google visualizado en el navegador. Si lo

abro y hago clic en el botón de los 3 puntos de la derecha, se despliega un menú. Escojo la opción **"Descargar mensaje"** e indicas un fichero donde guardarlo con formato **.eml**. Si usas un programa distinto de cliente de correo tendrá alguna funcionalidad similar aunque esté en otro lugar. Lo puedes investigar.

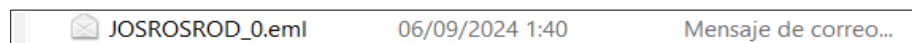


*Figura: contenido de un correo que voy a exportar.*

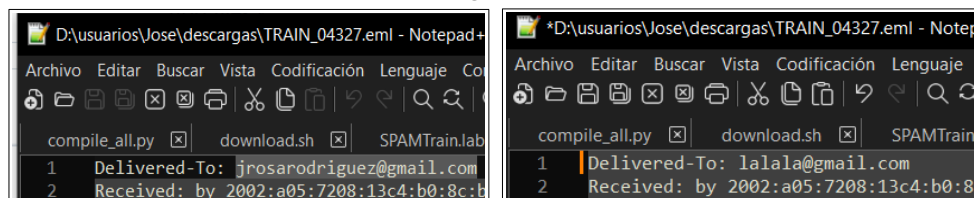
Una vez descargado, le cambiaremos el nombre y editamos el fichero con un editor de texto para eliminar contenido de índole personal y anonimizarlo.



Le cambio el nombre y lo dejo en la carpeta compartida de ficheros mail numerando de 0 a 9:



Y en cuanto al contenido, originalmente tiene una cuenta (una de mis cuentas de correo, el de la izquierda) y la cambio por otra inexistente (la figura de la derecha es como queda):



**ENTREGA 16:** Escoge de tu cuenta de correo entre 3 y 5 mails que sean spam y completa hasta un total de 10 mails que no lo sean y los guardas en la carpeta **datos** en formato **.eml**. Les cambias el nombre por **<3PrimerasLetrasDeTuNombre><3PrimerasLetrasDeTusApellidos>.eml**. Tras eliminar información sensible los subes al recipiente compartido por todos (carpeta de aules de la práctica/datos). Además, edita el fichero compartido **SPAMTrain.Label** y añade entradas donde los etiquetas (0 para spam y 1 para ham). Una vez que todos hayamos realizado esta aportación de nuevos datos, descargas el fichero de etiquetas y los ficheros con los mail y guardas los mail en la carpeta **./datos/training** y el fichero **SPAMTrain.Label** en **./datos**. A modo de ejemplo estos son mis emails y sus etiquetas:

Nombre	Fecha de modificación	Tipo
JOSROSROD_0.eml	06/09/2024 1:40	Mensaje de correo...
JOSROSROD_1.eml	06/09/2024 1:40	Mensaje de correo...
JOSROSROD_2.eml	06/09/2024 1:41	Mensaje de correo...
JOSROSROD_3.eml	06/09/2024 1:41	Mensaje de correo...
JOSROSROD_4.eml	06/09/2024 1:42	Mensaje de correo...
JOSROSROD_5.eml	06/09/2024 2:07	Mensaje de correo...
JOSROSROD_6.eml	06/09/2024 2:08	Mensaje de correo...
JOSROSROD_7.eml	06/09/2024 2:10	Mensaje de correo...
JOSROSROD_8.eml	06/09/2024 2:13	Mensaje de correo...
JOSROSROD_9.eml	06/09/2024 2:15	Mensaje de correo...
TRAIN_00000.eml	05/09/2024 18:44	Mensaje de correo...

SPAMTrain.Label	
4322	1 TRAIN_04321.eml
4323	0 TRAIN_04322.eml
4324	1 TRAIN_04323.eml
4325	1 TRAIN_04324.eml
4326	0 TRAIN_04325.eml
4327	0 TRAIN_04326.eml
4328	0 JOSROSROD_0.eml
4329	0 JOSROSROD_1.eml
4330	0 JOSROSROD_2.eml
4331	0 JOSROSROD_3.eml
4332	0 JOSROSROD_4.eml
4333	1 JOSROSROD_5.eml
4334	1 JOSROSROD_6.eml
4335	1 JOSROSROD_7.eml
4336	1 JOSROSROD_8.eml
4337	1 JOSROSROD_9.eml



Como el dataset inicial tiene muchos emails, seguramente estará comprimido con todos ellos en vez de estar independientes. Tras descargarlo, extraes los emails antes de entrenar el modelo.

## LA CLASE Email

La clase **Email** tiene la responsabilidad de representar un mail de entrada que siga los formatos indicados por las RFC para los emails. Leerá un archivo **.eml** y creará un objeto a partir de él. Para manejar esto usaremos una parte pequeña de todo el mensaje, a nuestro sistema solo le interesa el campo **subject** (el asunto) y el campo **body** (el mensaje o contenido). Cuando los mensajes contengan **HTML**, **texto plano** o **multipart** lo usaremos. Cualquier otra cosa la ignoramos, porque en realidad hay muchísimas opciones. Crea el fichero **objeto\_email.py** y copia en él el siguiente código:

```

1  import email
2  from bs4 import BeautifulSoup
3
4  class Email(object):
5
6      CLRF = "\n\r\n\r"
7
8      def __init__(self, archivo, categoria=None):
9          self.categoria = categoria
10         self.mail = email.message_from_binary_file(archivo)
11
12     def subject(self):
13         return self.mail.get("Subject")
14
15     def body(self):
16         payload = self.mail.get_payload()
17         if self.mail.is_multipart():
18             partes = [self._body_unico(parte) for parte in list(payload)]
19         else:
20             partes = [self._body_unico(self.mail)]
21         partes_decodificadas = []
22         for parte in partes:
23             if len(parte) == 0:
24                 continue
25             if isinstance(parte, bytes):
26                 partes_decodificadas.append(parte.decode("utf-8", errors="ignore"))
27             else:
28                 partes_decodificadas.append(parte)
29         return self.CLRF.join(partes_decodificadas)
30
31     @staticmethod
32     def _body_unico(parte):
33         tipo_de_contenido = parte.get_content_type()
34         try:
35             body = parte.get_payload(decode=True)
36         except Exception:
37             return ""
38         if tipo_de_contenido == "text/html":
39             return BeautifulSoup(body, "html.parser").text
40         elif tipo_de_contenido == "text/plain":
41             return body
42         return ""

```

**ENTREGA 17:** Crea el fichero **objeto\_email.py** con el código anterior. Debes usarlo en el fichero **genera\_dataset.py** que se encargará de procesar los emails guardados en los ficheros **.eml** que habrás preparado en la carpeta **./datos/training/**. Para cada uno, el programa **genera\_dataset.py** creará un ejemplo con dos campos **Label + texto**. En el campo de texto aparecerán las palabras del **subject** del email unidas a las palabras de cuerpo. El campo **label** tendrá una indicación de si es spam o ham. La marca concreta que utilices puede ser la que tu quieras (un código numérico, una palabra, etc.). El algoritmo del programa **genera\_dataset.py** será el siguiente:

- PASO 1. Para cada línea del fichero **SPAMTrain.Label:**
- PASO 2. **label, f** ← primer y segundo campo de la línea
- PASO 3. **mail** ← objeto Email creado a partir del fichero **./datos/training/f**
- PASO 4. Añade al fichero **./datos/dataset.csv** una línea con: **label + frases de subject y body**

Entrega el código de ambos programas Python y el fichero `dataset.csv` generado al ejecutarlos.

**ENTREGA 18.** Usa el dataset para crear varios clasificadores de emails (al menos 6 distintos) en spam o ham. Tienes libertad para hacerlos de cualquiera de las formas que hemos visto en la unidad o en ejercicios anteriores de esta práctica (lógicamente, se puntúa que realices procesamientos correctos, etc.). Ten en cuenta además que uses más o menos librerías, no estaría mal que ahora que los datos van a ser más abundantes, descartes “palabras” muy comunes que no aportan nada y para ello piensa que nuestros spam usarán castellano o valenciano y los iniciales utilizan el inglés (por si descartas palabras, etc.) Si usas *stop-words* adicionales o personalizados, pásalos. Entrega el código.

**ENTREGA 19.** Mide métricas de eficiencia de los clasificadores que hayas implementado.

**ENTREGA 20.** El mejor modelo, lo guardas en un fichero para usarlo en posibles aplicaciones. Investiga como se almacena un modelo en disco para usarlo posteriormente.