# CLASIFICACIÓN DE NOTICIAS USANDO LOS ALGORITMOS NAIVE BAYES Y K-NN

**José Manuel Gavira González**

**Arturo Ronda Lucena**

**ÍNDICE:**

# 1.-INTRODUCCIÓN

Hoy en día, cualquier persona puede acceder a una cantidad de información casi infinita gracias a internet. Tan solo accediendo a cualquier buscador, en un par de segundos, se pueden encontrar miles de documentos y textos relacionados con la búsqueda realizada.

En principio para cualquier usuario parece un proceso fácil: solo hay que buscar por una palabra o frase y aparecerán textos de la misma temática; lo que usualmente suele ignorar es el proceso anterior por el cual se han clasificado estos documentos. Dicha clasificación es fundamental para lograr que se pueda encontrar el tipo de documento deseado entre la ingente cantidad de información existente.

Debido a la enorme suma de información que se genera diariamente, es imposible que ésta pueda ser indizada y categorizada manualmente, por ello se usan algoritmos para ello, haciendo esta tarea mucho más asequible en coste y tiempo.

Para esta labor, los algoritmos de aprendizaje automático son una herramienta indispensable, puesto que son capaces de categorizar nuevos documentos a partir de un conjunto de ejemplos inicial de forma mecánica, sin necesidad de ser supervisados constantemente.

Concretamente, algoritmos basados en el agrupamiento y en redes bayesianas son ampliamente utilizados en este campo, ya que proporcionan unos resultados fiables, con unos modelos relativamente simples..

Un ejemplo que será estudiado en este documento es la clasificación de noticias de diferentes periódicos utilizando los algoritmos “Naive Bayes” y “k-NN”.

El modelo probabilístico de los clasificadores “Naive Bayes” se basa en el teorema de Bayes, y el adjetivo “naive” (ingenuo en inglés) viene de la suposición de que las características de un conjunto de datos son mutuamente independientes. En la práctica, la suposición de la independencia es a menudo violada, pero los clasificadores ingenuos de Bayes todavía tienden a funcionar muy bien bajo este supuesto poco realista.

“k-NN” por su parte, utiliza el agrupamiento de individuos en base a características comunes comparando cada individuo con los *k* individuos más parecidos y escogiendo la categoría en base a esos *vecinos.* A la hora de funcionar, la fase de entrenamiento es muy rápida, aunque la evaluación de cada individuo es algo más costosa, ya que no se genera ninguna generalización durante el entrenamiento.

## 1.1.-Objetivos del proyecto

En este proyecto se ha planteado como objetivos fundamentales el aprendizaje en el uso de los algoritmos dados en clase “k-NN” y “Naive Bayes”, centrandose en la elaboración de métodos de entrenamiento propios ajustados al conocimiento aprendido durante las clases, y la capacidad de almacenamiento de los modelos entrenados para la reutilización de los mismos sin necesidad de asumir un coste reiterado de entrenamiento fuera del periodo de ajuste de los mismos.

Es destacable dentro de los objetivos planteados la necesidad de llevar a cabo una documentación adecuada del desarrollo del proyecto, fomentando y mejorando de esta manera la capacidad de documentación del trabajo realizado, punto importante de cara a el mundo laboral, así como a asignaturas futuras como podría ser el trabajo de fin de grado, asignatura en la cual el desarrollo de una documentación adecuada, formal y completa, es fundamental.

## 1.2.-Objetivos técnicos

Así mismo también se pretende estudiar el uso de python, un lenguaje de programación no visto antes en cursos anteriores permitiendo a los alumnos encargados del proyecto en cuestión, el desarrollo de competencias tales como: el aprendizaje y uso de tecnologías desconocidas, la implementación de metodologías aprendidas previamente y la mejora de habilidades relacionadas con el trabajo en equipo.

# 2.-HERRAMIENTAS UTILIZADAS

## 2.1.-Python

Python es un lenguaje de programación de alto nivel interpretado, orientado a objetos. Tiene una sintaxis sencilla y fácil de aprender que enfatiza la legibilidad y por lo tanto reduce el costo del mantenimiento del programa. Python soporta módulos y paquetes, lo que fomenta la modularidad del programa y la reutilización de código.

El trabajo ha sido desarrollado en python 3.6.0 versión estable de este lenguaje de programación, las librerías empleadas y descritas a continuación tanto propias como externas han sido probadas en esta versión del mismo para un uso adecuado.

### 2.1.1.-Librerías empleadas

**string**
Contiene una serie de constantes y clases de utilidad que permiten acelerar ciertas operaciones a la hora de trabajar con cadenas.
Es usado para definir de forma rápida el conjunto de letras que se usarán para leer las palabras de las noticias desde archivos de texto.

**re**
Proporciona operaciones para evaluar expresiones regulares.
Es usado para separar cada palabra de una noticia.

**numpy v1.12.1**
Es el paquete fundamental para la computación científica en Python.
Es usado para crear listas con contadores para las apariciones de cada palabra clave en las noticias, además de para calcular las distancias entre vectores de pesos en k-NN.

**csv**
Implementa clases para leer y escribir datos en formato CSV.
Es usado para manejar archivos de texto con valores tabulados ( por ejemplo, para leer noticias después de dividirlas en conjuntos de entrenamiento y prueba o para guardar el modelo).

**copy**
Permite copiar colecciones mutables para evitar modificar la colección original.
Es usado para hacer una copia de las listas de noticias de entrenamiento y prueba, usadas a la hora de recorrerlas.

**scikit learn (sklearn)**
Incluye múltiples herramientas relacionadas con el aprendizaje automático.
Es usado para dividir las noticias de entrada en conjuntos de entrenamiento y prueba.

**argparse**
Permite la escritura de interfaces de línea de comandos fáciles de usar.
Es usado para la creación de la interfaz de línea de comandos del programa.

**sys**
Proporciona acceso a algunas variables utilizadas o mantenidas por el intérprete.
Es usado para comprobar que la interfaz de línea de comandos es llamada con argumentos.

**collections**
Implementa tipos de datos de contenedores especializados que proporcionan alternativas a los contenedores integrados de uso general de Python.
Es usado para crear un contador y estimar la categoría de un documento a partir de la categoría más repetida en los documentos vecinos.

**PyQt5**
Es un conjunto completo de enlaces de Python para Qt v5. Se implementa como más de 35 módulos de extensión y permite a Python ser utilizado como un lenguaje de desarrollo de aplicaciones alternativo a C ++ en todas las plataformas soportadas incluyendo iOS y Android.
Es usado para la creación de la interfaz gráfica del programa.

**pandas**
Es una biblioteca de código abierto que proporciona estructuras de datos de alto rendimiento y fácil de usar y herramientas de análisis de datos.
Es usado para crear un modelo de datos apto para dividir las noticias en conjuntos de entrenamiento y prueba

**math**
Proporciona acceso a las funciones matemáticas definidas por el estándar C.
Es usado para calcular potencias en el entrenamiento del algoritmo naive Bayes.

### 2.1.1.-Entorno de desarrollo

El proyecto ha sido desarrollado en un entorno Windows, concretamente en su versión Windows 10 de 64 bits y se ha empleado Anaconda versión 4.4.0 para python 3.6.
El IDE utilizado ha sido PyCharm en su versión Community 2017.2.

# 3.-IMPLEMENTACIÓN

## 3.1-Textos y categorías consideradas

Los textos utilizados provienen de las ediciones online de los periódicos El Mundo y El País, para así poder probar los algoritmos en un supuesto real. Son un total de 119 noticias, escritas en español y divididas en las siguientes categorías:

- Cultura
- Sociedad
- Deporte
- Tecnología
- Economía
- Ciencia

Con un total de 20 noticias por categoría (excepto 19 para ciencia).
Al ser noticias reales, la extensión de las mismas es variable, desde poco más de 200 palabras, a unas 1600, dando una media de 527 palabras por noticia.

Las noticias se encuentran escritas en un archivo de texto plano, estando cada una en una línea, facilitando la tarea de añadir más noticias para utilizar.

Además de dichos documentos, se entregan otras 6 noticias (cada una perteneciente a una de las categorías definidas anteriormente) en archivos de texto plano independientes, que son usadas para comprobar el funcionamiento del algoritmo, sabiendo de antemano la categoría a la que pertenecen y viendo si concuerda con la categoría en la que la clasifica el algoritmo.

## 3.2-Obtención de conjuntos de entrenamiento y test

Del número total de noticias, el usuario puede elegir el número de ellas que destinará a entrenar el modelo y a probarlo. A mayor número de noticias usadas para entrenar el modelo, mayor será la precisión de las predicciones hechas por los algoritmos.

La forma de dividir el total de noticias de entrada, es especificarle al programa el ratio de noticias que se usarán para entrenar el modelo, dejando el resto para realizar pruebas.

Teniendo en cuenta que el número de noticias de entrada es variable, se ha intentado que la división sea equitativa entre todas las noticias, para que a la hora de escoger los textos de entrenamiento, cada categoría está representada por un número similar de noticias. Esto se consigue utilizando el módulo *model_selection* del paquete *sklearn,* que hace una división pseudoaleatoria de las noticias de entrada. Para garantizar que el programa proporcione la misma salida siempre que se ejecute con los mismo parámetros de entrada, se ha establecido una semilla fija que será usada para generar la secuencia de noticias aleatorias a escoger.

Una vez obtenido el conjunto de entrenamiento, el comportamiento de cada programa difiere sensiblemente:

**k-NN**
Una vez separados los conjuntos de noticias, se generan 4 archivos de texto plano:

- *noticias_entrenamiento.csv*
- *noticias_test.csv*
- *categorias_entrenamiento.csv*
- *categorias_test.csv*

En *noticias_entrenamiento.csv* se guardan las noticias del conjunto de entrenamiento, poniendo una en cada línea. A su vez, en *categorias_entrenamiento.csv* se guardan las categorías de éstas noticias, poniendo la categoría de cada noticia en la misma línea.

**Naive Bayes**
Cuando se obtiene el conjunto de entrenamiento, se entrena el modelo con dichos datos y luego se guarda el modelo en el archivo *entrenamiento_NB* y el conjunto de pruebas queda guardado en memoria para su uso, en caso de un uso posterior no será necesario entrenar el modelo pero si indicar el tamaño del conjunto de pruebas.

## 3.3-Conjunto de palabras clave

Ambos métodos de clasificación necesitan de un conjunto de palabras clave, que junto a la categoría dada de cada noticia de entrenamiento, permiten al algoritmo calcular la probabilidad de una noticia de prueba de pertenecer a una u otra categoría en función de las palabras clave que contiene y del número de apariciones de estas.

Dichas palabras clave se han obtenido mediante un análisis previo de los documentos, en el cual se han seleccionado las palabras más representativas de cada categoría sin que lleguen a ser totalmente disjuntas las palabras de forma que dos categorías pueden tener una alta aparición de una misma palabra en sus noticias respectivas. Las palabras clave en resumen son las palabras más repetidas después de eliminar palabras comunes a cualquier tipo de noticia (por ejemplo, preposiciones y artículos).

A continuación se muestran las palabras clave escogidas divididas por categorías:

| **Cultura** | **Sociedad** | **Deportes** | **Tecnología** | **Economía** | **Ciencia** |
|---|---|---|---|---|---|
| historia | policía | puntos | apple | millones | espacial |
| novela | guardia | temporada | pantalla | euros | universidad |
| película | civil | partido | compañía | españa | ciencia |
| obra | violencia | final | windows | crecimiento | tierra |
| flamenco | menor | madrid | usuarios | empresas | espacio |
| artista | fuentes | equipo | google | corrupción | proyecto |
| premio | agentes | nba | videojuegos | mercado | años |
| obras | drogas | partidos | nueva | sector | estudio |
| arte | investigación | carrera | euros | economía | estudio |
| teatro | hechos | ganar | móvil | comercial | incertidumbre |
| fotografía | robos | contra | nintendo | gobierno | huellas |
| escenario | nacional | piloto | iphone | banco | luna |
| canciones | alcohol | victoria | teléfono | deuda | millones |
| cultura | hombre | torneo | ordenadores | vivienda | energía |
| festival | mujer | minutos | mundo | trimestre | investigación |
| género | detenido | derrota | juego | oferta | agujero |
| concierto | hospital | playoffs | diseño | déficit | mundo |
| personajes | justicia | ganar | whatsapp | inversión | metros |
| pintor | española | título | sistema | compra | planeta |
| trabajo | denuncia | liga | microsoft | dinero | kilómetros |
| escritor | género | jugador | batería | demanda | precisión |
| bailarines | víctimas | podio | jugadores | país | especie |

| | | | | | |
|---|---|---|---|---|---|
| ballet | emergencias | mundial | fabricación | tasa | científicos |
| fotografía | droga | ganador | | | |
| | reyerta | falta | | | |
| | abusos | jornada | | | |
| | víctima | defensa | | | |

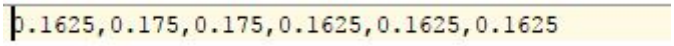
## 3.4-Extracción de las características del conjunto de entrenamiento

## 3.4.1-Naive Bayes

### 3.4.1.1-Cálculo de P(c)

P(c) en Naive Bayes es la probabilidad a priori en nuestro caso de una categoría de noticias, este valor es obtenido para cada categoría mediante la división del número de noticias pertenecientes a una categoría entre el número total de noticias que se encuentran en el conjunto de entrenamiento.

La probabilidad a priori de cada categoría es necesaria para el entrenamiento del modelo, y por lo tanto han de ser almacenadas dentro del mismo para ser usadas en futuras predicciones. Este almacenamiento se lleva a cabo en el método *training_NB_writer*, en el cual se escribe en la primera fila del documento *entrenamiento_NB* la probabilidad a priori de cada una de las categorías separadas por comas como se muestra a continuación:




### 3.4.1.2-Cálculo de P(t|c)

P(t|c) en Naive Bayes es la probabilidad condicionada en nuestro caso de la ocurrencia de una palabra clave en los textos de una categoría, este valor es obtenido calculando el total de ocurrencias de una palabra clave en los textos de una categoría, entre el número de ocurrencias de esa misma palabra clave en todos los textos del conjunto de entrenamiento.

La probabilidad condicionada ha sido suavizada en la implementación del algoritmo para evitar que se produzcan probabilidades con valor 0 puesto que a posteriori en el proceso de categorización de un texto estos valores estropearían la predicción. Este suavizado se ha llevado a cabo mediante la suma de k=1 al numerador y del número de palabras clave a considerar al denominador.

De la misma manera que ocurría con la probabilidad a priori la probabilidad condicionada ha de ser almacenada para poder reutilizar el entrenamiento del modelo en futuras predicciones. Por ello la probabilidad condicionada de cada palabra clave es almacenada en el fichero entrenamiento_NB en una línea según la categoría y separadas por comas, como se muestra en parte a continuación.

0.005368647100930566,0.005368647100930566,0.00501073729420186 1,0.00501073729420186 1,0.006084466714387974,0.0028632784538

0.0010737294201861132,0.0003579098067287043 5,0.0003579098067287043 5,0.0003579098067287043 5,0.0003579098067287043 5,0.0003

0.0007158196134574087,0.0003579098067287043 5,0.0007158196134574087,0.0003579098067287043 5,0.0003579098067287043 5,0.00035

0.0017895490336435219,0.0003579098067287043 5,0.0003579098067287043 5,0.0003579098067287043 5,0.0003579098067287043 5,0.0003

0.0010737294201861132,0.0003579098067287043 5,0.0003579098067287043 5,0.0014316392269148174,0.0003579098067287043 5,0.00035

0.0014316392269148174,0.0003579098067287043 5,0.0007158196134574087,0.0007158196134574087,0.0003579098067287043 5,0.000715

## 3.4.2-k-NN

### 3.4.2.1-Cálculo de la representación vectorial $W_d$

Este apartado tiene como fin el hacer al lector una introducción al cálculo de los vectores de pesos de las noticias usadas para entrenamiento del modelo. Dichos vectores consisten en una lista con un número de posiciones igual al número de palabras clave usadas, que en cada posición lleva un valor que simboliza el “peso” que tiene cada una de esas palabras clave en en la noticia.

El primer paso es calcular una matriz que contenga las frecuencias de cada una de las palabras clave en cada uno de los documentos. Esta matriz tendrá un tamaño de *pxn,* siendo p el número de palabras clave y n el número de noticias. En cada celda de la matriz se guardará el número de veces que la palabra clave p aparece en la noticia n.

Una vez calculadas las frecuencias, la matriz tendrá un diseño parecido a éste:

| | $Noticia_1$ | $Noticia_2$ | $Noticia_3$ | $Noticia_n$ |
|---|---|---|---|---|
| $Palabra\ clave_1$ | 0 | 2 | 1 | 0 |
| $Palabra\ clave_2$ | 3 | 1 | 0 | 2 |
| $Palabra\ clave_3$ | 1 | 2 | 1 | 0 |
| $Palabra\ clave_p$ | 1 | 0 | 0 | 2 |

Además de las frecuencias de los términos, es necesario calcular la frecuencia documental inversa de cada palabra clave, para ello, es necesario obtener antes las frecuencias documentales de cada palabra clave.
Las frecuencias documentales de las palabras clave se guardarán en un vector de tamaño igual al número de palabras clave *p*, que contendrá en cada posición el número de noticias en las que aparece.

Después de construirlo, este será el diseño de dicho vector:

| Frecuencia documental de la palabra clave 1 | Frecuencia documental de la palabra clave 2 | Frecuencia documental de la palabra clave 3 | Frecuencia documental de la palabra clave 4 | Frecuencia documental de la palabra clave n |
|---|---|---|---|---|
| 5 | 3 | 8 | 9 | 6 |

Una vez se tiene el vector con las frecuencias documentales de las palabras clave, ya se pueden calcular las frecuencias documentales inversas de cada término.

Las frecuencias documentales inversas se almacenarán en un vector de tamaño igual al número de palabras clave, en el que el valor de la posición i-ésima es calculado como el logaritmo en base diez de la división del número total de noticias de entrenamiento entre el valor de la posición i-ésima del vector de frecuencias documentales.

El formato del vector será de la siguiente forma:

| Frecuencia documental inversa de la palabra clave 1 | Frecuencia documental inversa de la palabra clave 2 | Frecuencia documental inversa de la palabra clave 3 | Frecuencia documental inversa de la palabra clave 4 | Frecuencia documental inversa de la palabra clave n |
|---|---|---|---|---|
| 0.301 | 0.523 | 0.097 | 0.046 | 0.222 |

Por último, ya solo es necesario multiplicar la matriz de frecuencias por el vector de frecuencias documentales inversas. Esta operación dará como resultado una matriz de tamaño *pxn,* siendo p el número de palabras clave y n el número de noticias.
En dicha matriz, cada columna representa el vector de pesos de cada palabra clave en una noticia. Tales pesos indican la relevancia que se le da  a cada palabra clave dentro de la noticia.

Una vez calculados los pesos, la matriz tendrá un diseño parecido a éste:

| | $Pesos_1$ | $Pesos_2$ | $Pesos_3$ | $Pesos_n$ |
|---|---|---|---|---|
| $Palabra\ clave_1$ | 0 | 0.602 | 0.301 | 0 |
| $Palabra\ clave_2$ | 1.569 | 0.523 | 0 | 1.046 |
| $Palabra\ clave_3$ | 0.097 | 0.194 | 0.097 | 0 |
| $Palabra\ clave_p$ | 0.222 | 0 | 0 | 0.444 |

Éstos vectores de pesos son los que se compararán con el vector de pesos de las noticias de prueba a categorizar, por lo tanto es necesario guardarlos en un archivo para su posterior uso.

El guardado de los valores se realiza en un archivo csv, en el que en cada línea se guarda el peso de una palabra clave en cada documento separado por comas. El número de líneas será igual al número de palabras clave, habiendo en cada línea un número de valores igual al número de documentos usados para el entrenamiento.

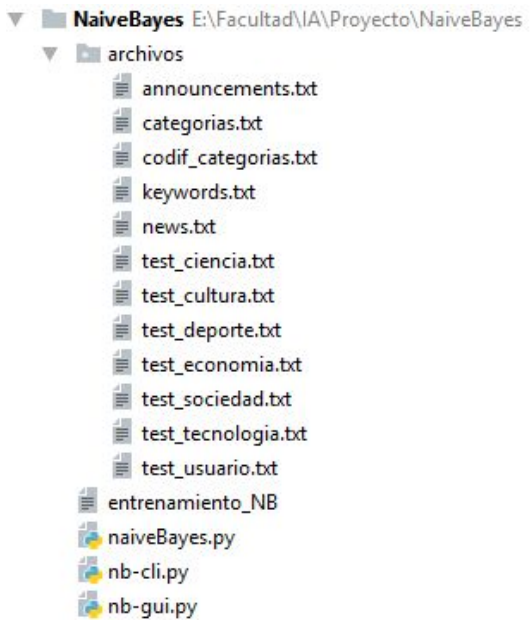
Siguiendo con el ejemplo que se ha desarrollado en el apartado, el archivo quedaría de la siguiente forma:

```
1  0.0,0.602,0.301,0.0
2  1.569,0.523,0.0,1.046
3  0.097,0.194,0.097,0.0
4  0.222,0.0,0.0,0.444
```

## 3.5-Implementación de Naive Bayes

### 3.5.1 - Estructura de ficheros

Estos son los archivos que usa el programa para funcionar:




Todos los documentos de la carpeta *archivos* son de texto plano y los que se encuentran en el directorio raíz son ficheros escritos en el lenguaje Python. A continuación se expone el contenido y la finalidad de cada uno de los archivos:

**archivos/categorias.txt**

Contiene las categorías de todos las noticias guardadas en *archivos/noticias.csv* de forma que en cada línea se encuentra la categoría de la noticia que se encuentra en la misma línea, codificada como un número entero (0=cultura,1=sociedad,2=deporte,3=tecnología,4=economía,5=ciencia).

Por ejemplo, si una noticia se encuentra en la línea 10 de *archivos/noticias.csv* y en la línea 10 de *archivos/categorias.csv* hay un 2, significa que esa noticia es de deporte.

**entrenamiento_NB**

Se genera después de entrenar el modelo. En él están guardados los datos generados al crear un modelo a partir de las noticias de entrenamiento.

Es un archivo con un número de líneas igual al número de categorías usado más una línea extra, que en la primera línea contendrá un número de valores decimales igual al número de categorías usadas y en el resto uno por cada palabra clave, separados por comas.

**archivos/news.txt**

Tiene el total de noticias de entrada. Cada noticia se encuentra en una línea.

**archivos/keywords.txt**

Contiene las palabras clave que se usarán para calcular los vectores de pesos tanto de las noticias de entrenamiento como de las que se vayan a categorizar. Cada palabra clave está escrita en una línea.

**archivos/test_ciencia.txt**

Contiene una noticia de la categoría “Ciencia”, usada para comprobar si el algoritmo es capaz de clasificarla correctamente. El usuario puede cambiar el texto para categorizar una noticia diferente. El archivo debe contener una sola línea con la noticia escrita en ella.

**archivos/test_cultura.txt**

Contiene una noticia de la categoría “Cultura”, usada para comprobar si el algoritmo es capaz de clasificarla correctamente. El usuario puede cambiar el texto para categorizar una noticia diferente. El archivo debe contener una sola línea con la noticia escrita en ella.

**archivos/test_deporte.txt**

Contiene una noticia de la categoría “Deporte”, usada para comprobar si el algoritmo es capaz de clasificarla correctamente. El usuario puede cambiar el texto para categorizar una noticia diferente. El archivo debe contener una sola línea con la noticia escrita en ella.

**archivos/test_economia.txt**

Contiene una noticia de la categoría “Economía”, usada para comprobar si el algoritmo es capaz de clasificarla correctamente. El usuario puede cambiar el texto para categorizar una noticia diferente. El archivo debe contener una sola línea con la noticia escrita en ella.

**archivos/test_sociedad.txt**

Contiene una noticia de la categoría “Sociedad”, usada para comprobar si el algoritmo es capaz de clasificarla correctamente. El usuario puede cambiar el texto para categorizar una noticia diferente. El archivo debe contener una sola línea con la noticia escrita en ella.

**archivos/test_tecnologia.txt**

Contiene una noticia de la categoría “Tecnología”, usada para comprobar si el algoritmo es capaz de clasificarla correctamente. El usuario puede cambiar el texto para categorizar una noticia diferente. El archivo debe contener una sola línea con la noticia escrita en ella.

**archivos/test_usuario.txt**

Contiene una noticia de la categoría elegida por el usuario,que será clasificada por el algoritmo. El usuario puede cambiar el texto para categorizar una noticia a su elección. El archivo debe contener una sola línea con la noticia escrita en ella.

**nb-cli.py**

Es la interfaz de línea de comandos (CLI) que permite el uso del programa desde una terminal de texto.

**nb-gui.py**

Es la interfaz gráfica de usuario (GUI) que permite al usuario ejecutar el programa de una forma sencilla.

**naiveBayes.py**

Incluye la lógica necesaria para ejecutar el algoritmo Naive Bayes sobre los datos suministrados. Proporciona métodos para crear las probabilidades a priori y condicionales en documentos y para clasificar noticias a partir de dicha representación.

### 3.5.2 - División de noticias

El primer paso para el funcionamiento del programa, es dividir las noticias de entrada en un conjunto de entrenamiento y otro de prueba. Para ello, es necesario usar el método *readNews* y *splitDataset* dentro de *naiveBayes.py*, devuelve un conjunto de entrenamiento y otro de pruebas creados de forma proporcional al ratio proporcionado, siendo el conjunto de pruebas: 1-ratio.

**Ejecución**

1º- Se leen todas las noticias de entrada desde el archivo en el que están guardadas con el método *read_aonnuncements(ruta_noticias)* del archivo *naiveBayes.py* .Dicho método, devuelve una lista de listas que en cada posición contiene una palabra de la noticia, con los símbolos de puntuación tales como puntos y comas eliminados para facilitar la identificación de palabras en la fase de entrenamiento.

2º- Se leen las categorías de las noticias anteriores con el método read_categories*(ruta_categorias)* del archivo *naiveBayes.py* que a partir de la ruta del archivo donde están guardadas las categorías, devuelve una lista de un número de posiciones igual al número de noticias, que en cada posición tiene un valor de 0 a 5 que codifica la categoría de la noticia de la fila n-ésima de la matriz obtenida en el paso anterior.

3º-Se leen las palabras clave de las noticias con el método *read_keywords(ruta_keywords)* del archivo *naiveBayes.py* que a partir de la ruta del archivo donde están guardadas las palabras clave, devuelve una lista de un número de posiciones igual al número de palabras clave.

4º- Con las matrices obtenidas en el paso 1,2 y 3,se llama al método *keyword_number_of_multiple_news* en el cual se genera una nueva matriz con las noticias por filas y el conteo de palabras clave como columnas, añadiendo una columna extra con la categoría codificada de cada noticia.
.
5º Se realiza la división de noticias de la matriz anterior, devolviendo el conjunto de las noticias que se usarán para crear el modelo y sus correspondientes categorías y las noticias que se usarán para comprobar la fiabilidad del algoritmo.

### 3.5.3 - Entrenamiento del modelo

Después de obtener el conjunto de noticias de entrenamiento, hay que proceder a crear el modelo. Para ésto, hay que usar el método *prepareTrainingSet* en el archivo *naiveBayes.py* que a partir de los archivos creados en el paso anterior, crea un archivo en el que se guarda la probabilidad a priori de las categorías y la probabilidad condicionada de las palabras clave en los diferentes documentos.

**Ejecución**

1º- Carga las noticias de entrenamiento y sus categorías.

2º- Se crea la lista probabilidades a priori mediante el método *calculatorPc(categorias, conjunto_entrenamiento, keywords)* en el archivo *naiveBayes.py* que devuelve una lista con el valor resultante de dividir la aparición de cada categoría entre el número total de documentos.

3º- Se crea la matriz de probabilidades condicionadas para cada palabra clave según la categoría empleando el método calculatorPtc*(categorias, conjunto_entrenamiento, keywords)* del archivo *naiveBayes.py* el cual devuelve una matriz con el número de palabras clave como columnas y el número de noticias como filas, con el valor resultante de dividir el número total de apariciones de la palabra clave en los documentos de esa categoría más k=1 entre el número total de apariciones de esa palabra en todos los documentos más el número de palabras clave.

4º- Se guardan los resultados anteriores en el archivo *entrenamiento_NB* empleando el método *training_NB_writer(pc, ptc, ruta_guardado)* en el archivo *naiveBayes.py* se escribe la probabilidad a priori de cada categoría en la primera línea separadas por comas y la probabilidad condicionada de cada palabra clave en una columna por palabra y línea por cada categoría separando los valores por comas.

### 3.5.4 - Testeo del modelo con noticias de prueba

Una vez se tiene el modelo guardado, ya se puede pasar a testear su efectividad. Para ello, se usará el conjunto de noticias destinadas a prueba para comprobar el porcentaje de noticias que el algoritmo es capaz de categorizar correctamente.

Esto se consigue con el método *getPredictions(pc,ptc,conjunto_prueba)* del archivo *naiveBayes.py,* que calcula la categoría más probable para cada noticia de prueba, y el método *getAccuracy(conjunto_prueba, predicciones)* devuelve el porcentaje de aciertos al clasificarlas y el metodo *load_training_NB(ruta_Modelo)* para extraer el entrenamiento guardado.

**Ejecución**

1º- Se lee el fichero en el cual se ha guardado el modelo entrenado para extraer los valores de las probabilidades a priori de cada categoría y las probabilidades condicionadas de cada palabra clave segun la categoria.

2º- Por cada noticia se eleva la probabilidad condicionada de cada palabra clave segun la categoria, al número de apariciones de la palabra en la noticia y se multiplican los resultados de la operación anterior entre ellos. Luego este resultado se multiplica con la probabilidad a priori de su categoría.

3º-Una vez obtenidas las probabilidades de cada categoría se selecciona la que tenga una mayor probabilidad y se añade a la lista de predicciones.

4º- Una vez completada la lista de predicciones se compara la predicción de cada noticia de prueba con la su categoría real y se calcula el porcentaje de aciertos contra ese conjunto.

### 3.5.5 - Estimación de la categoría de diferentes documentos

El programa permite estimar las categorías de documentos sueltos introducidos por el usuario. Para ello, es necesario poner una noticia en un archivo determinado y el programa mostrará la categoría estimada por el algoritmo.

Esto se consigue con el método *predict2(pc, ptc, noticia)* del archivo *naiveBayes.py,* que mediante el modelo guardado calcula la categoría más probable.

**Ejecución**

1º- Se carga el modelo entrenado.

2º- Para la noticia seleccionada se eleva la probabilidad condicionada de cada palabra clave segun la categoria, al número de apariciones de la palabra en la noticia y se multiplican los resultados de la operación anterior entre ellos. Luego este resultado se multiplica con la probabilidad a priori de su categoría.

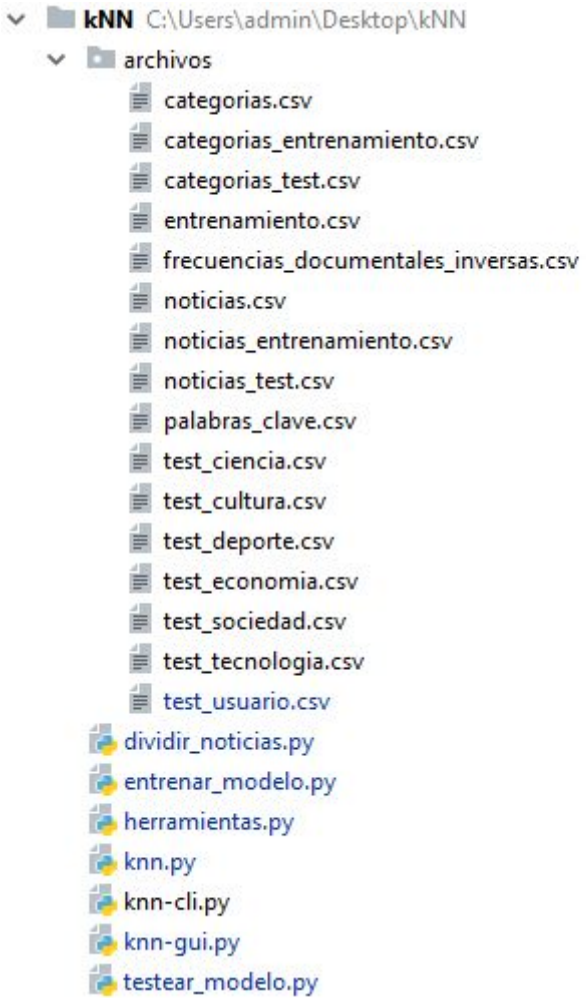
3º- Se calcula la categoría más probable según la lista de probabilidades anterior.

4º- Se traduce mediante un diccionario la codificación de la categoría resultante.

## 3.6-Implementación de k-NN

### 3.6.1 - Estructura de ficheros

Estos son los archivos que usa el programa para funcionar:




Todos los documentos de la carpeta *archivos* son de texto plano y los que se encuentran en el directorio raíz son ficheros escritos en el lenguaje Python. A continuación se expone el contenido y la finalidad de cada uno de los archivos:

**archivos/categorias.csv**

Contiene las categorías de todos las noticias guardadas en *archivos/noticias.csv* de forma que en cada línea se encuentra la categoría de la noticia que se encuentra en la misma línea, codificada como un número entero (0=cultura,1=sociedad,2=deporte,3=tecnología,4=economía,5=ciencia).

Por ejemplo, si una noticia se encuentra en la línea 10 de *archivos/noticias.csv* y en la línea 10 de *archivos/categorias.csv* hay un 2, significa que esa noticia es de deporte.

**archivos/categorias_entrenamiento.csv**

Se genera después de dividir las noticias iniciales. Tiene el mismo formato que el fichero anterior, siendo este archivo usado para indicar las categorías de las noticias del conjunto de entrenamiento, las cuales están guardadas en *archivos/noticias_entrenamiento.csv.*

**archivos/categorias_test.csv**

Se genera después de dividir las noticias iniciales. Mismo formato que el anterior, usado para categorizar las noticias del conjunto de prueba que están en *archivos/noticias_test.csv*.

**archivos/entrenamiento.csv**

Se genera después de entrenar el modelo. En él están guardados los datos generados al crear un modelo a partir de las noticias de entrenamiento.

Es un archivo con un número de líneas igual al número de palabras clave usado, que en cada línea contendrá un número de valores decimales igual al número de noticias usadas de entrenamiento, separados por comas.

**archivos/frecuencias_documentales_inversas.csv**

Se genera después de entrenar el modelo. Contiene el vector de frecuencias documentales inversas calculado con las palabras clave y las noticias de entrenamiento. En cada línea aparece la frecuencia documental inversa de una palabra clave.

**archivos/noticias.csv**

Tiene el total de noticias de entrada. Cada noticia se encuentra en una línea.

**archivos/noticias_entrenamiento.csv**

Se genera después de dividir las noticias iniciales. En él se encuentran las noticias que se usarán para crear el modelo. Como en el archivo anterior, cada noticia se encuentra en una línea diferente, pero las palabras están separadas por comas.

**archivos/noticias_test.csv**

Se genera después de dividir las noticias iniciales. En él se encuentra el conjunto de noticias que se usarán para testear el modelo. Tiene el mismo formato que el fichero anterior.

**archivos/palabras_clave.csv**

Contiene las palabras clave que se usarán para calcular los vectores de pesos tanto de las noticias de entrenamiento como de las que se vayan a categorizar. Cada palabra clave está escrita en una línea.

**archivos/test_ciencia.csv**

Contiene una noticia de la categoría “Ciencia”, usada para comprobar si el algoritmo es capaz de clasificarla correctamente. El usuario puede cambiar el texto para categorizar una noticia diferente. El archivo debe contener una sola línea con la noticia escrita en ella.

**archivos/test_cultura.csv**

Contiene una noticia de la categoría “Cultura”, usada para comprobar si el algoritmo es capaz de clasificarla correctamente. El usuario puede cambiar el texto para categorizar una noticia diferente. El archivo debe contener una sola línea con la noticia escrita en ella.

**archivos/test_deporte.csv**

Contiene una noticia de la categoría “Deporte”, usada para comprobar si el algoritmo es capaz de clasificarla correctamente. El usuario puede cambiar el texto para categorizar una noticia diferente. El archivo debe contener una sola línea con la noticia escrita en ella.

**archivos/test_economia.csv**

Contiene una noticia de la categoría “Economía”, usada para comprobar si el algoritmo es capaz de clasificarla correctamente. El usuario puede cambiar el texto para categorizar una noticia diferente. El archivo debe contener una sola línea con la noticia escrita en ella.

**archivos/test_sociedad.csv**

Contiene una noticia de la categoría “Sociedad”, usada para comprobar si el algoritmo es capaz de clasificarla correctamente. El usuario puede cambiar el texto para categorizar una noticia diferente. El archivo debe contener una sola línea con la noticia escrita en ella.

**archivos/test_tecnologia.csv**

Contiene una noticia de la categoría “Tecnología”, usada para comprobar si el algoritmo es capaz de clasificarla correctamente. El usuario puede cambiar el texto para categorizar una noticia diferente. El archivo debe contener una sola línea con la noticia escrita en ella.

**archivos/test_usuario.txt**

Contiene una noticia de la categoría elegida por el usuario,que será clasificada por el algoritmo. El usuario puede cambiar el texto para categorizar una noticia a su elección. El archivo debe contener una sola línea con la noticia escrita en ella.

**dividir_noticias.py**

Realiza la división de las noticias leídas desde el archivo *archivos/noticias.csv* en dos conjuntos: entrenamiento y prueba.

Contiene las llamadas a los métodos para leer las noticias iniciales, dividirlas y guardarlas en archivos separados.

**entrenar_modelo.py**

Entrena el modelo a partir de las noticias de entrenamiento que se encuentran en el archivo *archivos/noticias_entrenamiento.csv* y genera la representación vectorial de pesos de cada noticia. A continuación guarda ésta representación en *archivos/entrenamiento.csv*

**herramientas.py**

Contiene todos los métodos auxiliares usados para leer y escribir información en archivos de texto y a partir de ésta información , crear estructuras de datos adecuadas para ser utilizadas por el programa.

**knn.py**

Incluye la lógica necesaria para ejecutar el algoritmo kNN sobre los datos suministrados. Proporciona métodos para crear la representación vectorial de pesos de palabras clave en documentos y para clasificar noticias a partir de dicha representación.

**knn-cli.py**

Es la interfaz de línea de comandos (CLI) que permite el uso del programa desde una terminal de texto.

**knn-gui.py**

Es la interfaz gráfica de usuario (GUI) que permite al usuario ejecutar el programa de una forma sencilla.

**testear_modelo.py**

Contiene todas las llamadas a los métodos necesarios para clasificar noticias (en conjunto o una a una) a partir del entrenamiento previo del modelo.

### 3.6.2 - División de noticias

El primer paso para el funcionamiento del programa, es dividir las noticias de entrada en un conjunto de entrenamiento y otro de prueba. Para ello, es necesario usar el archivo dividir_noticias.py, que contiene un único método *divide(ratio_noticias_entrenamiento)* que dado un número decimal mayor que 0 y menor que 1, crea 4 ficheros de texto plano -2 en los que se guardan las noticias entrenamiento/prueba y 2 en los que se guardan sus correspondientes categorías- dejando para entrenamiento un porcentaje de noticias proporcional al ratio pasado y destinando a prueba las restantes.

**Ejecución**

1º- Se leen todas las noticias de entrada desde el archivo en el que están guardadas con el método *leer_multiples_noticias(ruta_noticias)* del archivo *herramientas.py* .Dicho método, devuelve una lista de n (n=nº de noticias de entrenamiento) listas que en cada posición contienen una palabra de la noticia, con los símbolos de puntuación tales como puntos y comas eliminados para facilitar la identificación de palabras en la fase de entrenamiento.

2º- Se leen las categorías de las noticias anteriores con el método *leer_categorias(ruta_categorias)* del archivo *herramientas.py* que a partir de la ruta del archivo donde están guardadas las categorías, devuelve una lista de un número de posiciones igual al número de noticias, que en cada posición tiene un valor de 0 a 5 que codifica la categoría de la noticia de la fila n-ésima de la matriz obtenida en el paso anterior.

3º- A la matriz obtenida en el paso 1, se le agrega el vector obtenido en el paso 2, de ésta forma, en cada fila se encuentra una lista de tantas posiciones como palabras tenga la noticia más una posición adicional en la que se guarda la categoría de dicha noticia.

4º Se realiza la división de noticias de la matriz anterior, escribiendo los archivos *noticias_entrenamiento.csv* y *categorias_entrenamiento.csv* en los que se guardan las noticias que se usarán para crear el modelo y sus correspondientes categorías; y los archivos *noticias_test.csv* y *categorias_test.csv* en los que se guardan las noticias que se usarán para comprobar la fiabilidad del algoritmo.

### 3.6.3 - Entrenamiento del modelo

Después de obtener el conjunto de noticias de entrenamiento, hay que proceder a crear el modelo. Para ésto, hay que usar el archivo *entrenar_modelo.py* que contiene el método *entrenar()* que a partir de los archivos creados en el paso anterior, crea un archivo en el que se guarda la representación vectorial de pesos de las palabras clave en los diferentes documentos.

**Ejecución**

1º- Carga las noticias de entrenamiento y sus categorías desde los archivos correspondientes.

2º- Se crea la matriz de frecuencias de cada documento con el método *frecuencia_de_terminos_en_documentos(documentos,keywords)* del archivo *knn.py* que por cada palabra del vector de keywords crea una lista de largo igual al tamaño del vector de documentos, en el que cada posición indica el número de veces que aparece la palabra clave en un documento. Luego, añade esas listas a una lista para crear una matriz y la devuelve.

3º- Se calcula el vector de frecuencias documentales con el método *frecuencias_documentales(documentos,terminos)* del archivo *knn.py* que devuelve una lista de tamaño igual al vector de términos en el que cada posición indica el número de noticias en las que aparece cada palabra clave.

4º- Se calcula el vector de frecuencias documentales inversas de las palabras clave con el método *frecuencias_documentales_inversas(total_keywords,total_news,f_docs)* que a partir del número total de palabras clave y noticias y del vector de frecuencias documentales, devuelve un vector de tamaño igual al número de palabras clave en el que la posición i-ésima es calculada como el logaritmo en base 10 de la división del número total de noticias de entrenamiento entre el valor de la posición i-ésima del vector de frecuencias documentales.

5º- Se crea la matriz de pesos con el método *pesos_de_terminos_en_documentos(frecuencias,frecuencias_inversas),* que recibe la matriz de frecuencias de términos y el vector de frecuencias documentales inversas, cogiendo la matriz calculada en el paso 2 y multiplicando cada valor de la matriz situado en la fila i, columna j, por el valor i-ésimo del vector de frecuencias documentales inversas. Posteriormente, se devuelve la nueva matriz.

6º- Se guarda el vector generado en el paso 4 en el archivo *archivos/frecuencias_documentales_inversas.csv* con el método *guardar_frecuencias_documentales_inversas(f_documentales_inv,ruta_archivo_f_doc)* que escribe el vector en un archivo de texto, poniendo cada valor en una fila diferente.

7º- Se guarda la matriz calculada en el paso anterior en el archivo archivos/*entrenamiento.csv* con el método *guardar_pesos(matriz,ruta)* del archivo *herramientas.py* que escribe la matriz en un archivo de texto, guardando los valores de cada fila en una línea con los valores separados por comas.

### 3.6.4 - Testeo del modelo con noticias de prueba

Una vez se tiene el modelo guardado, ya se puede pasar a testear su efectividad. Para ello, se usará el conjunto de noticias destinadas a prueba para comprobar el porcentaje de noticias que el algoritmo es capaz de categorizar correctamente.

Esto se consigue con el método *testear_multiples_noticias(k)* del archivo *testear_modelo.py,* que dado un número de vecinos con el que comparar cada noticia de prueba, devuelve el porcentaje de aciertos al clasificarlas.

**Ejecución**

1º- Se cargan las palabras clave y la representación vectorial del modelo de pesos.

2º- A cada vector de pesos del conjunto de entrenamiento, se le añade una posición adicional, en la que se almacena la categoría de la noticia.

3º- Se cargan las noticias de prueba y sus categorías.

4º- A partir del vector de frecuencias documentales inversas, se calcula el vector de pesos de cada noticia del conjunto de prueba

5º- Por cada vector de pesos, se calcula la distancia entre ese vector y el vector de pesos de los k vecinos más cercanos.

6º- De entre los vecinos calculados, se escoge la categoría a la que pertenecen la mayoría de ellos, asignándose al documento evaluado y guardándose dicha categoría en un vector.

7º- Se compara el vector de categorías reales (paso 3) con el vector de categorías estimadas (paso 7) y se devuelve el porcentaje de aciertos

### 3.6.5 - Estimación de la categoría de diferentes documentos

El programa permite estimar las categorías de documentos sueltos introducidos por el usuario. Para ello, es necesario poner una noticia en un archivo determinado y el programa mostrará la categoría estimada por el algoritmo.

Esto se consigue con el método *testear_noticia(k,ruta_noticia_test)* del archivo *testear_modelo.py,* que dado un número de vecinos con el que comparar la noticia de prueba y la ruta del archivo, devuelve la categoría estimada.

**Ejecución**

1º- Se cargan las palabras clave y la representación vectorial del modelo de pesos.

2º- A cada vector de pesos del conjunto de entrenamiento, se le añade una posición adicional, en la que se almacena la categoría de la noticia.

3º- Se carga la noticia de prueba y su categorías.

4º- Se calcula el vector de pesos asociado a la noticia de prueba.

5º- Se calcula la distancia entre el vector de pesos de la noticia y el vector de pesos de los k vecinos más cercanos.

6º- De entre los vecinos calculados, se escoge la categoría a la que pertenecen la mayoría de ellos, asignándose al documento evaluado.

7º- Se devuelve la categoría estimada.

# 4-RESULTADOS

## 4.1-Resultados de los experimentos

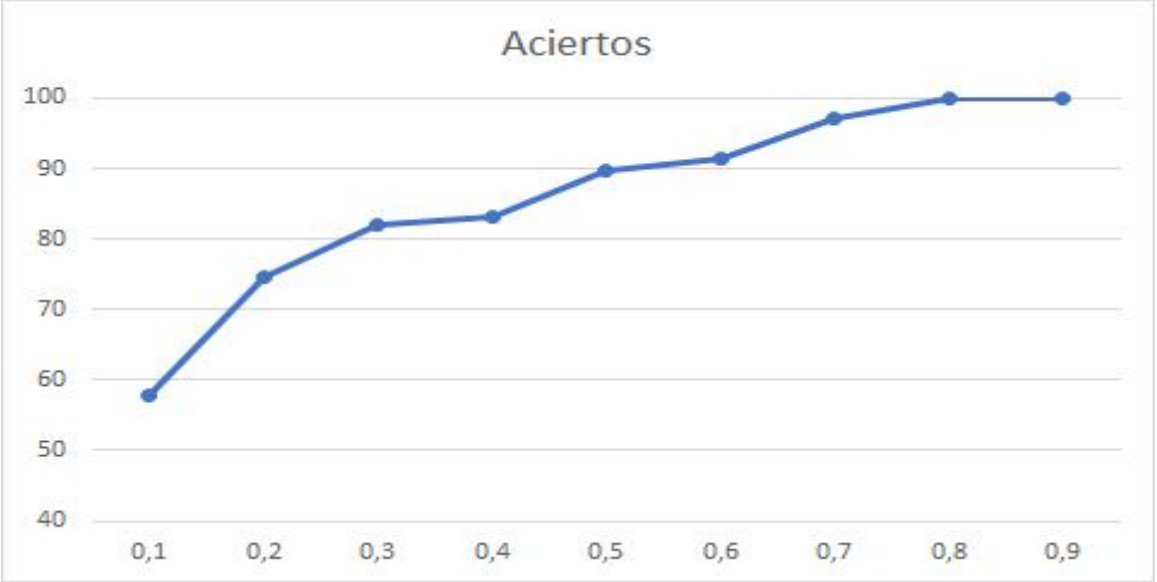
### 4.1.1-Naive Bayes

Para comprobar el desempeño del programa, se han ejecutado múltiples casos de prueba, en los que se han cambiado el número de noticias de entrenamiento.

En todas las pruebas el número de noticias de entrada total es de 119 y el número de palabras clave 147.

La gráfica representa como evoluciona el porcentaje de aciertos del algoritmo a medida que varía el porcentaje del conjunto de entrenamiento.

| nº noticias entrenamiento | nº noticias prueba | Aciertos |
|---|---|---|
| 12 | 107 | 57,94 |
| 24 | 95 | 74,74 |
| 36 | 83 | 81,93 |
| 48 | 71 | 83,09 |
| 60 | 59 | 89,83 |
| 72 | 47 | 91,49 |
| 84 | 35 | 97,14 |
| 96 | 23 | 100 |
| 108 | 11 | 100 |




Como se puede observar en la gráfica a medida que aumenta el porcentaje de entrenamiento del algoritmo el nivel de aciertos frente al conjunto de prueba aumenta hasta llegar entorno al 80% de entrenamiento y 20% prueba que el algoritmo acierta el 100% de los textos.

### 4.1.2-k-NN

Para comprobar el desempeño del programa, se han ejecutado múltiples casos de prueba, en los que se han cambiado el número de noticias de entrenamiento y el número de vecinos usados para determinar la categoría de cada noticia.

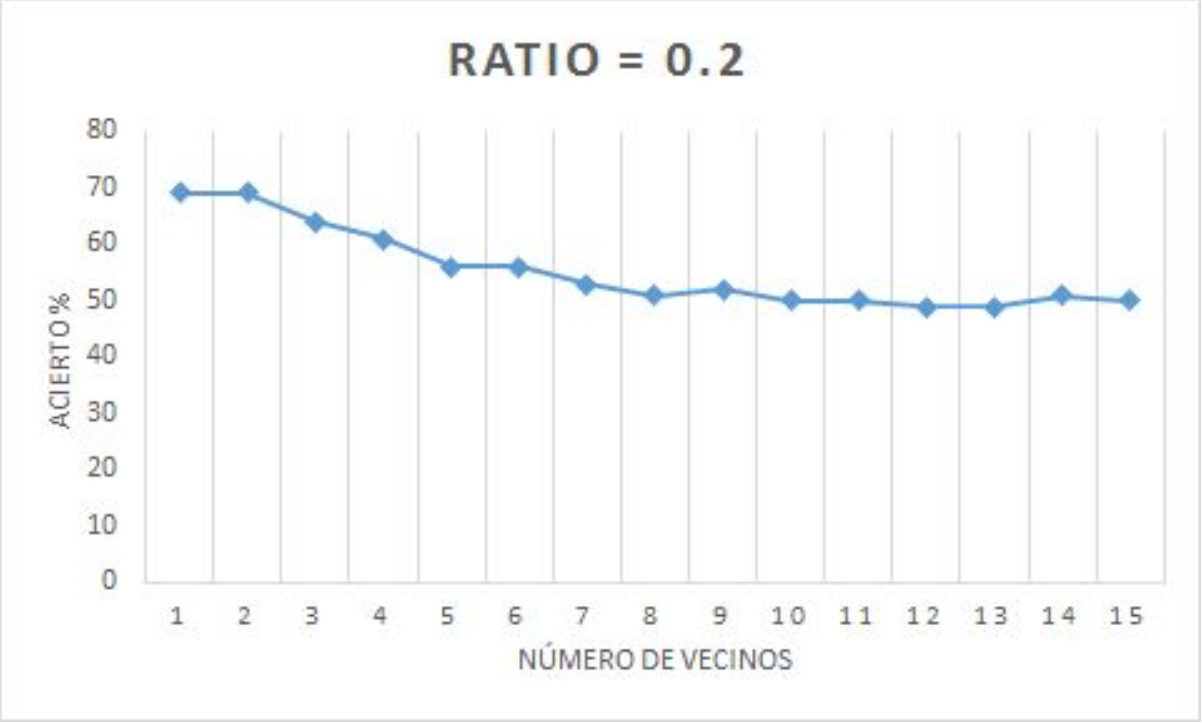
En todas las pruebas el número de noticias de entrada total es de 119 y el número de palabras clave 147.

El número de vecinos que aparece en las pruebas varía entre 1 y 15, puesto que a medida que se incrementa el k por encima de 15, el porcentaje de noticias bien categorizadas baja en el 94% de los casos.

Las gráficas representan cómo varía el porcentaje de acierto a medida que el número de vecinos para comparar cambia.

**20% de noticias destinadas a entrenamiento**

Ratio: 0.2
Noticias de entrenamiento: 24
Noticias de prueba: 95




El máximo porcentaje de acierto alcanzado es un 69% que se da cuando k={1,2}

**40% de noticias destinadas a entrenamiento**

Ratio: 0.4
Noticias de entrenamiento: 48
Noticias de prueba: 71


El máximo porcentaje de acierto alcanzado es un 87% que se da cuando k={8}

**67% de noticias destinadas a entrenamiento**

Ratio: 0.67
Noticias de entrenamiento: 80
Noticias de prueba: 39

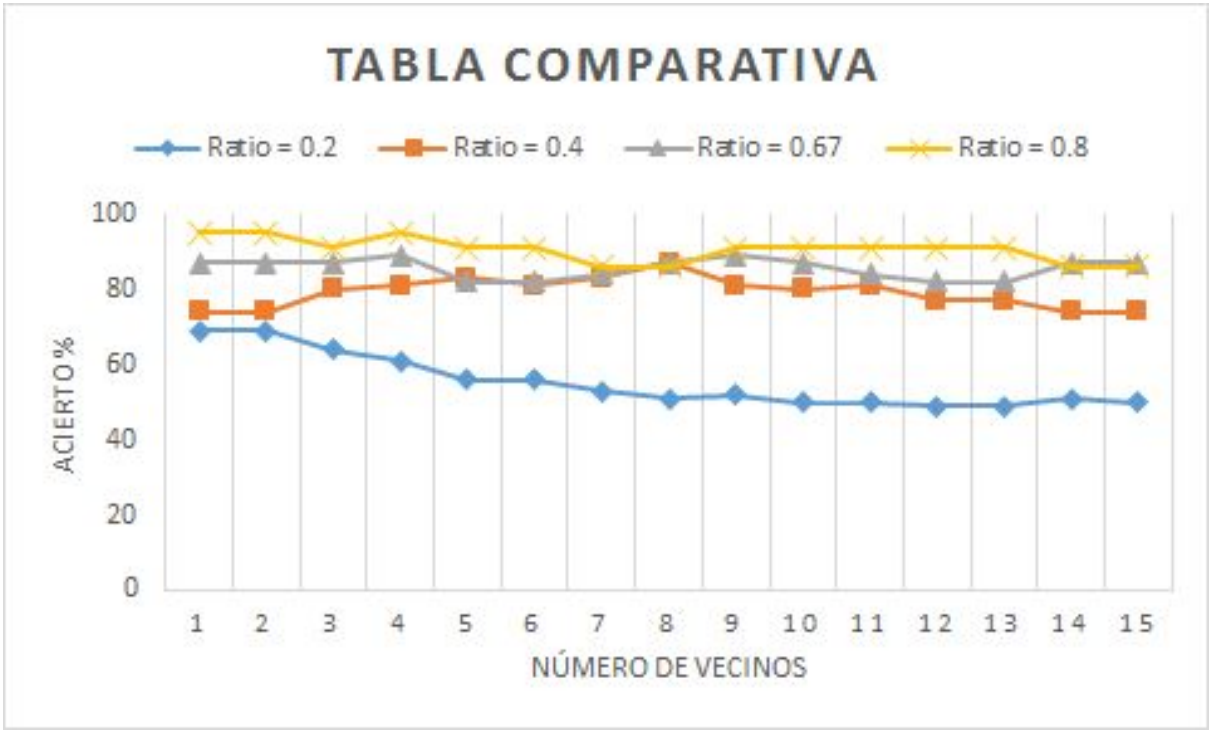

El máximo porcentaje de acierto alcanzado es un 89% que se da cuando k={4,9}

**80% de noticias destinadas a entrenamiento**

Ratio: 0.8
Noticias de entrenamiento: 96
Noticias de prueba: 23


El máximo porcentaje de acierto alcanzado es un 95% que se da cuando k={1,2,4}

**Comparativa**



Como se puede ver, cuanto mayor es el número de noticias de entrenamiento, mayor precisión tiene el algoritmo. Además de la cantidad de noticias de prueba, el parámetro k también es un aspecto fundamental a la hora de que el algoritmo trabaje correctamente, puesto que un k incorrecto, puede llevar al algoritmo a dar un resultado pésimo. Asimismo, debido a que kNN aproxima solo localmente, dos valores de k parecidos pueden dar resultados muy dispares.

# 5-MANUAL DE USUARIO

## 5.1- Naive Bayes

### 5.1.1- Interfaz de línea de comandos

Para usar la interfaz de línea de comandos hay que usar el archivo *nb-cli.py* ubicado en la raíz del proyecto. Para ejecutarlo, es necesario llamar el script con el intérprete de python.

Para obtener ayuda y visualizar todos los comandos que contiene el programa, se usa el parámetro -h.

```
E:\Facultad\IA\Proyecto\NaiveBayes>python nb-cli.py -h
usage: nb-cli.py [-h] [-p PORCENTAJE] [-e] [-t TESTEAR_NOTICIA] [-tn]

optional arguments:
  -h, --help            show this help message and exit
  -p PORCENTAJE, --porcentaje PORCENTAJE
                        Establece el porcentaje de noticias que se usaran para
                        entrenamiento o testeo, por defecto se cogera el 0.67
  -e, --entrenar        Entrena el modelo a partir de los archivos de noticias
                        y categorias pasados
  -t TESTEAR_NOTICIA, --testear_noticia TESTEAR_NOTICIA
                        Estima la categoria de un documento dado segun el
                        nombre de la categoria
  -tn, --testear_multiples_noticias
                        Estima las categorias de varios documentos a la vez
```

*Salida del parámetro -h*

#### 5.1.1.1- Dividir noticias para entrenamiento

Prepara las noticias para entrenar y testear el modelo a partir de las noticias guardadas en *archivos/noticias.csv* y categorizadas con los datos del archivo *archivos/categorias.csv.*

Si se quieren añadir noticias, tan solo hay que introducirlas una por línea en formato de texto plano en el archivo correspondiente e indicar su categoría en la misma línea en el archivo de categorías.

Para dividir las noticias iniciales en conjuntos de entrenamiento y prueba, y entrenar el modelo, se usa el parámetro -p [ratio] -e.

-p: Indica al programa que debe realizar la división de noticias y el porcentaje.

-e: Indica que se quiere entrenar el modelo con el porcentaje de noticias especificado por el parámetro -p.  Si no se especifica usa el 67% de todas las noticias suministradas para entrenar el modelo.

```
E:\Facultad\IA\Proyecto\NaiveBayes>python nb-cli.py -e
El modelo ha sido entrenado con el 67% de las noticias por defecto.
```

*Uso del parámetro -e sin establecer un ratio manualmente*

```
E:\Facultad\IA\Proyecto\NaiveBayes>python nb-cli.py -e -p 0.8
El modelo ha sido entrenado con 96 noticias.
```

*Uso del parámetro -e con un ratio de entrenamiento de 0.8*

#### 5.1.1.2- Testear modelo

Para comprobar la fiabilidad del modelo, se usa el modelo creado en el paso anterior, y un conjunto de prueba del que como se conoce su categoría, permite comparar con la categoría que el algoritmo estima para cada una de ellas, informando de la fiabilidad del algoritmo.

La ejecución de este paso necesita de dos parámetros:
-tn : indica que se testearán múltiples noticias.
-p : indica el porcentaje de noticias con las que se probara el algoritmo en caso de no especificarse se usarán el 95% de las noticias.

```
E:\Facultad\IA\Proyecto\NaiveBayes>python nb-cli.py -tn
Porcentaje de aciertos: 96.46017699115043% contra 113 noticias de prueba
```

*Uso del parámetro -tn sin -p*

```
E:\Facultad\IA\Proyecto\NaiveBayes>python nb-cli.py -tn -p 0.33
Porcentaje de aciertos: 97.43589743589743% contra 39 noticias de prueba
```

*Uso del parámetro -tn con -p= 0.33*

#### 5.1.1.3- Categorizar una noticia

Si se quiere que el algoritmo clasifique una noticia en una categoría, es necesario usar los siguientes parámetros:

-t "*categoría"* : indica que se categoriza la noticia guardada en el archivo indicado por su categoría.

Con el programa se incluyen 7 archivos de prueba (uno de cada categoría) dentro de la carpeta *archivos* el nombre de estos archivos son:

test_ciencia.txt Contiene una noticia de la categoría "Ciencia".
test_cultura.txt Contiene una noticia de la categoría "Cultura".
test_deporte.txt Contiene una noticia de la categoría "Deporte".
test_economia.txt Contiene una noticia de la categoría "Economía".
test_sociedad.txt Contiene una noticia de la categoría "Sociedad".
test_tecnologia.txt Contiene una noticia de la categoría "Tecnología".
test_usuario.txt Contiene una noticia de la categoría "Desconocida" y aqui el usuario podrá introducir una noticia a su elección.

Para usar otro archivo, hay que tener en cuenta que este debe tener solo una línea en la cual se encontrará la noticia escrita en texto plano.

```
E:\Facultad\IA\Proyecto\NaiveBayes>python nb-cli.py -t Economía
Su noticia es de Economía!!
```

*Uso del parámetro -t con una noticia ya incluida*

```
E:\Facultad\IA\Proyecto\NaiveBayes>python nb-cli.py -t Desconocida
Su noticia es de Tecnología!!
```

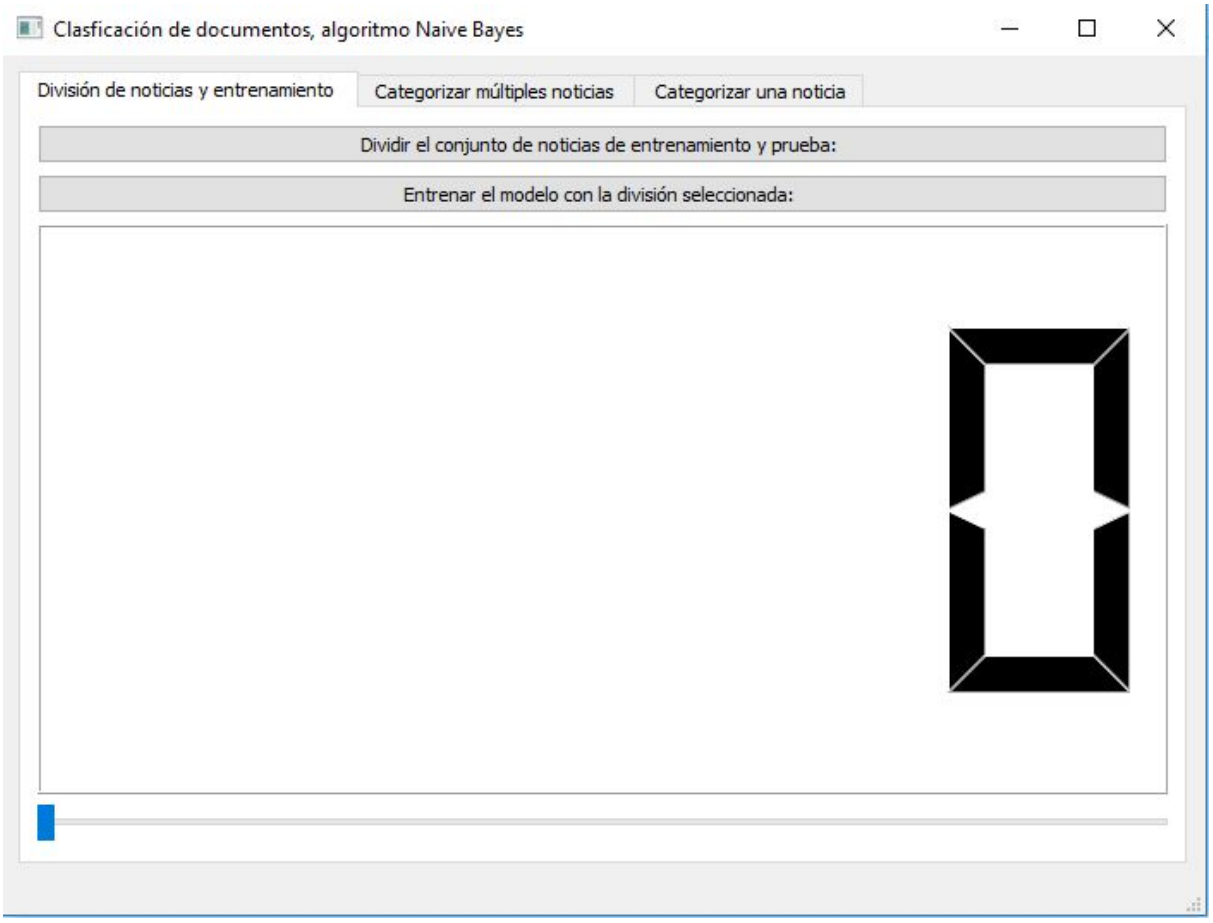
*Uso del parámetro -t con una noticia introducida por el usuario*

## 5.1.2- Interfaz gráfica

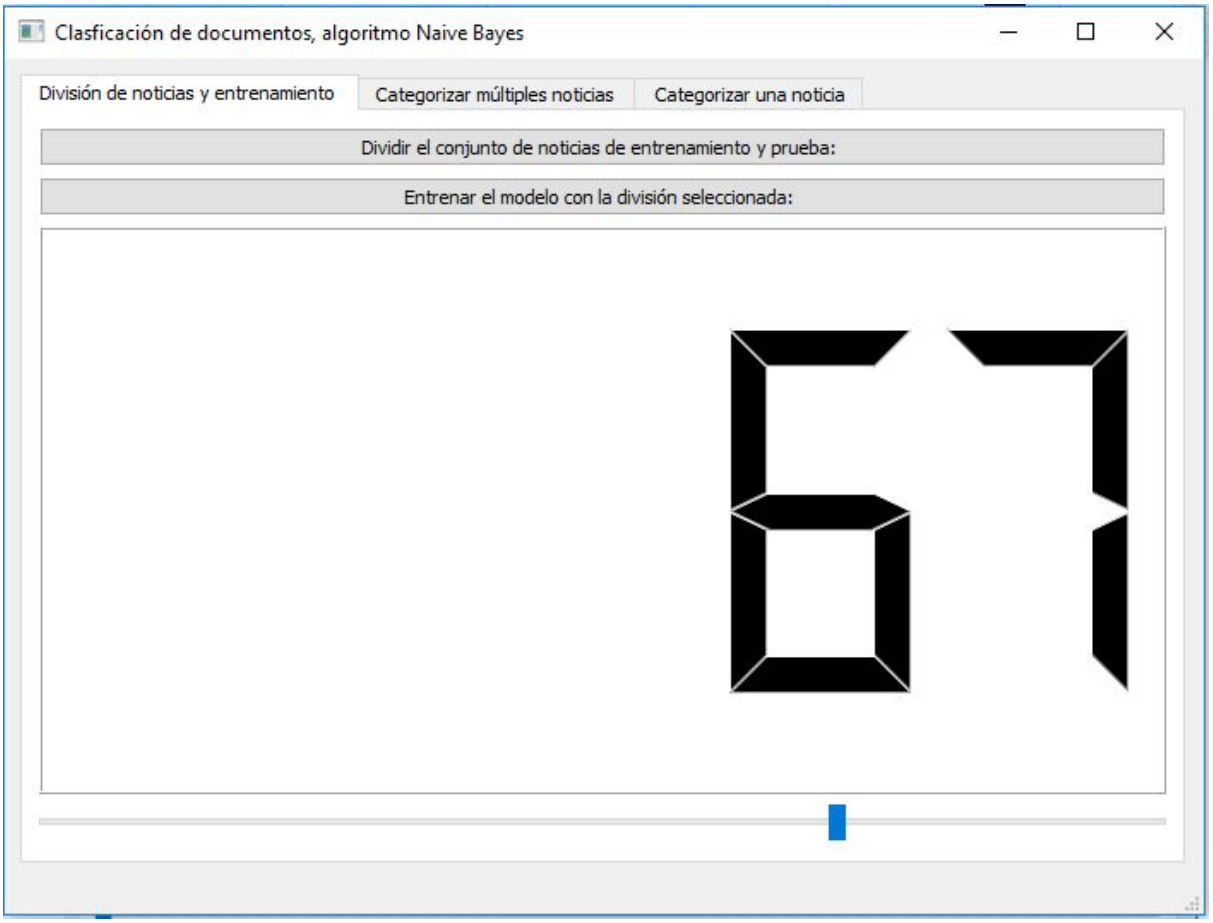
Para usar la interfaz gráfica hay que usar el archivo *nb-gui.py* ubicado en la raíz del proyecto. Para ejecutarlo, es necesario llamar el script con el intérprete de python.

### 5.1.2.1- Dividir noticias de entrada

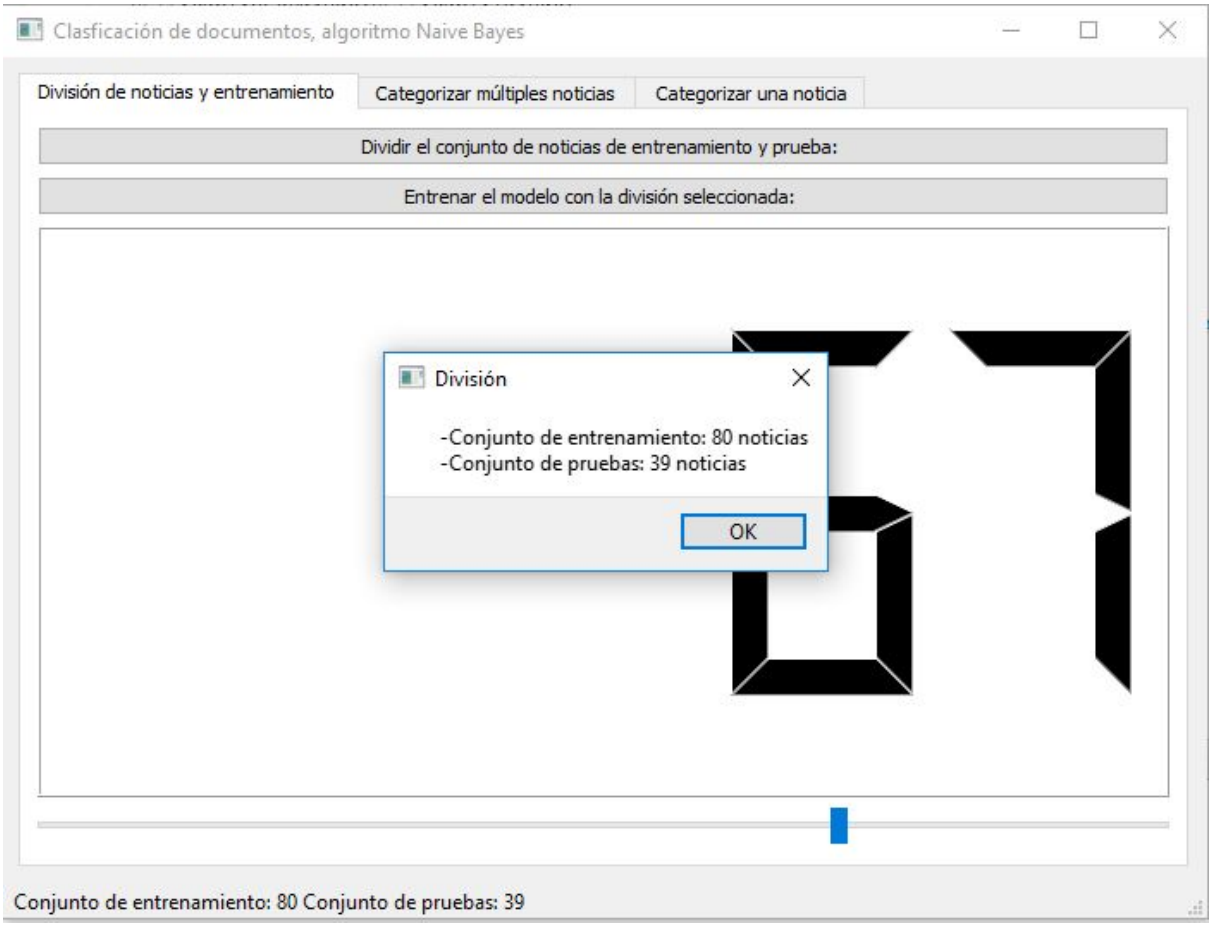
Se realiza en la pestaña "*División de noticias y entrenamiento*".
Para dividir las noticias de entrada, es necesario especificar el porcentaje de noticias destinadas a entrenar y a prueba, con la barra de la parte baja de la ventana y a continuación pulsar en el botón "*Dividir el conjunto de noticias de entrenamiento y prueba:*"




*Vista inicial de la pestaña*



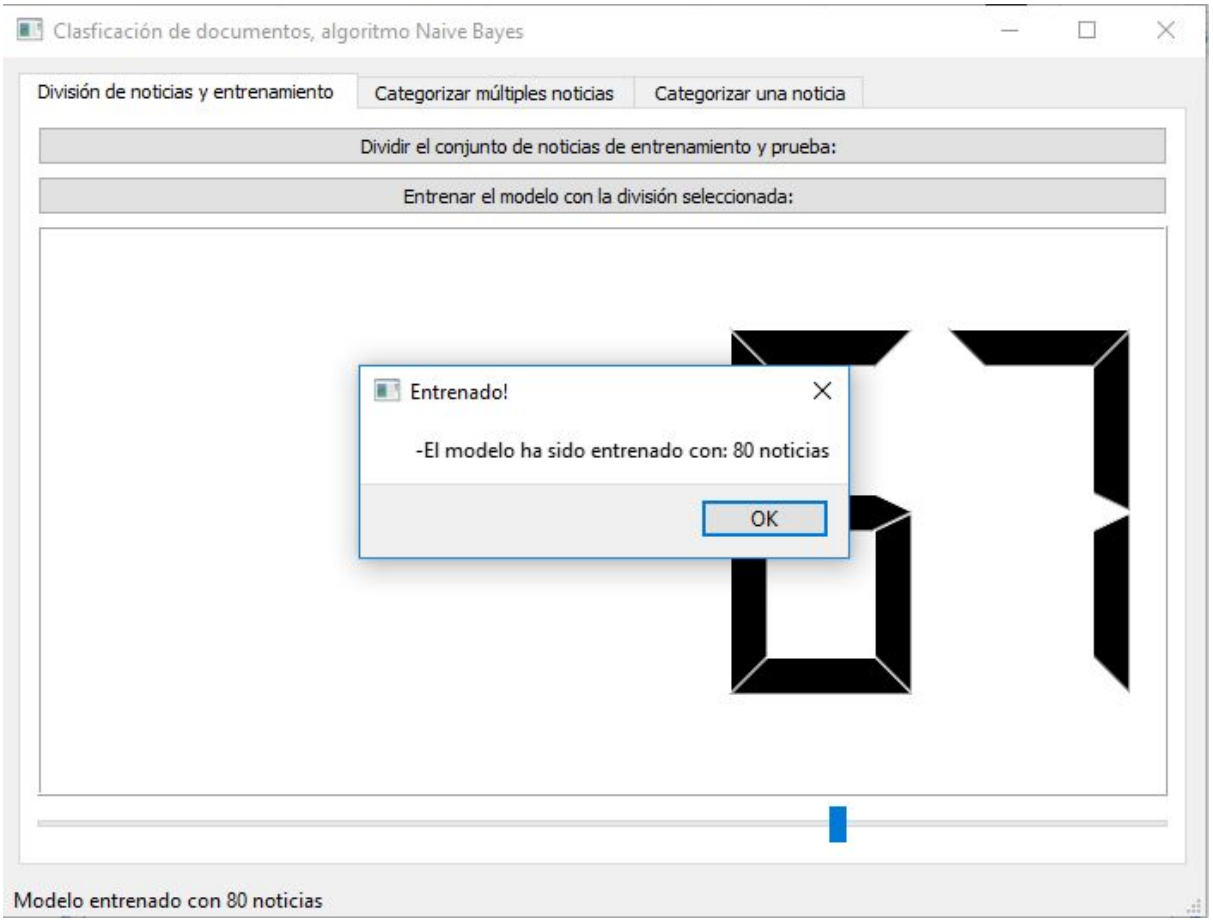

*Cambiando el porcentaje de noticias de entrenamiento y prueba*




*Mensaje mostrado después de pulsar el botón*

### 5.1.2.2- Entrenar modelo

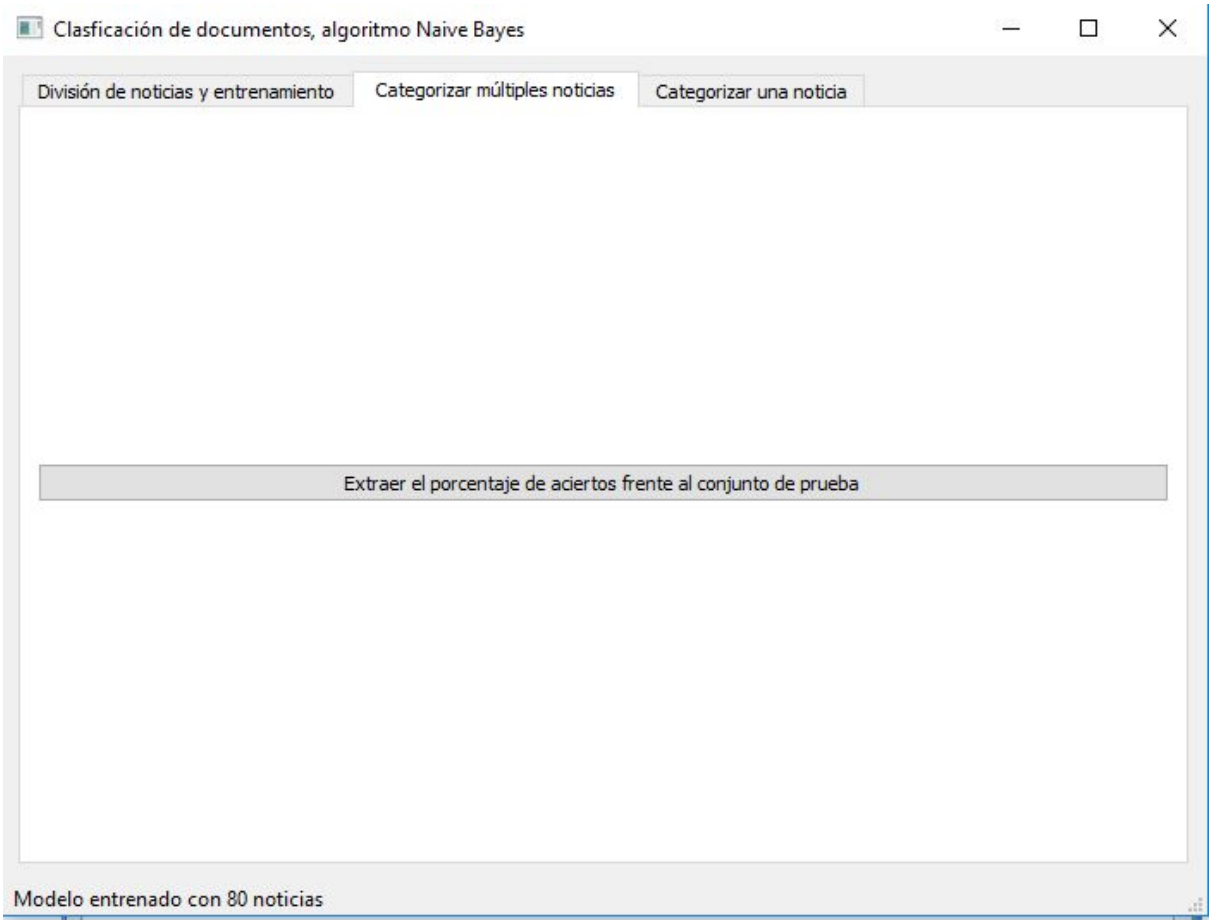
Se realiza en la pestaña "*División de noticias y entrenamiento*".
Para crear el modelo, es necesario pulsar en el botón "*Dividir el conjunto de noticias de entrenamiento y prueba:*" y luego pulsar el botón "*Entrenar el modelo con la división seleccionada:*".



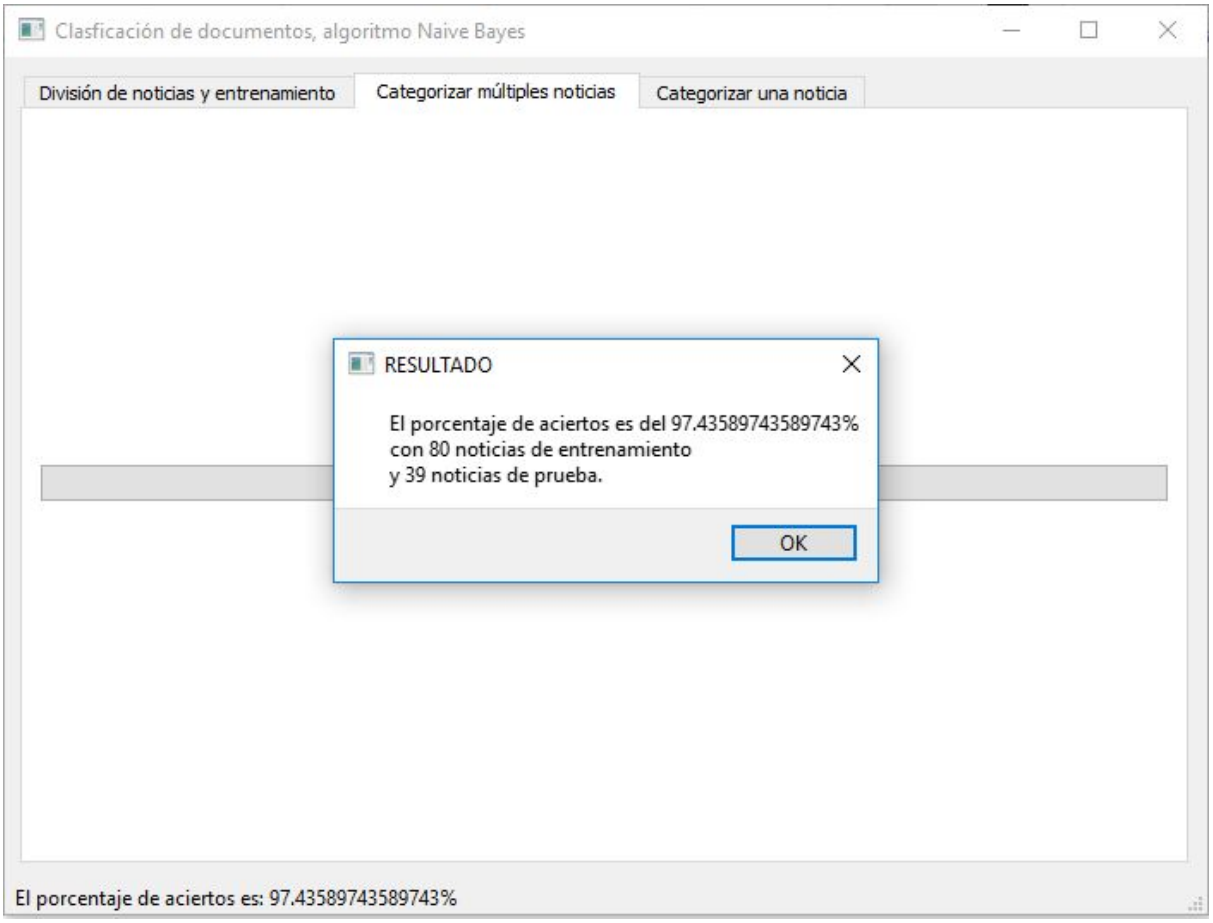

*Mensaje mostrado después de pulsar el botón*

#### 5.1.2.3- Testear modelo

Se realiza en la pestaña "*Categorizar múltiples noticias*".
Para ejecutarlo, es necesario pulsar en el botón "*Extraer un porcentaje de aciertos frente al conjunto de prueba:*" y haber seleccionado una división previamente..




*Vista inicial de la pestaña*

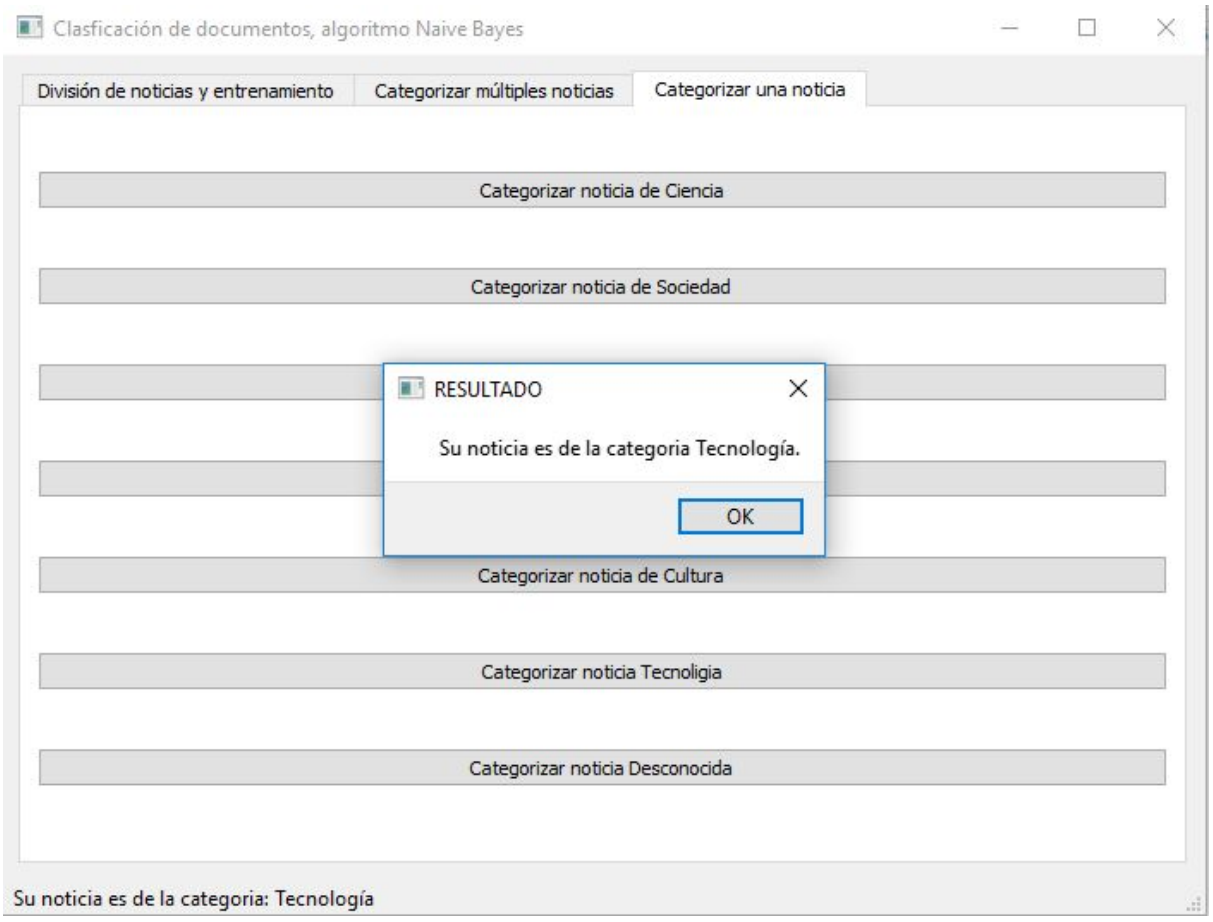



*Mensaje mostrado después de pulsar el botón*

#### 5.1.2.4- Categorizar una noticia

Se realiza en la pestaña "*Categorizar una noticia*".
Para ejecutarlo, es necesario establecer el número de vecinos con el que comparar cada noticia y luego pulsar en el botón correspondiente a la noticia que se quiera categorizar.
Los primeros 6 botones testean una noticia de una categoría conocida. El último botón "*Categorizar noticia Desconocida* " permite estimar la categoría de una noticia introducida por el usuario. Dicha noticia debe estar guardada en el fichero "*archivos/test_usuario.txt*" en una única línea como texto plano.


*Vista inicial de la pestaña*




*Mensaje mostrado después de pulsar el botón "Categorizar noticia de categoría desconocida!*"

## 5.2- k-NN

### 5.2.1- Interfaz de línea de comandos

Para usar la interfaz de línea de comandos hay que usar el archivo *knn-cli.py* ubicado en la raíz del proyecto. Para ejecutarlo, es necesario llamar el script con el intérprete de python.

Para obtener ayuda y visualizar todos los comandos que contiene el programa, se usa el parámetro -h.

```
(C:\Users\admin\Anaconda3) C:\Users\admin\Desktop\kNN>python knn-cli.py -h
usage: knn-cli.py [-h] [-d] [-r RATIO] [-e] [-k K_VECINOS]
                  [-t TESTEAR_NOTICIA] [-tn]

optional arguments:
  -h, --help            show this help message and exit
  -d, --dividir         Indica si se quiere dividir el conjunto de
                        entrenamiento
  -r RATIO, --ratio RATIO
                        Establece el ratio de noticias que se usarβn para
                        entrenamiento, por defecto se cogerβn todas.
  -e, --entrenar        Entrena el modelo a partir de los archivos de noticias
                        y categorΎas pasados
  -k K_VECINOS, --k_vecinos K_VECINOS
                        Indica con cuantos vecinos se compararβ una noticia
  -t TESTEAR_NOTICIA, --testear_noticia TESTEAR_NOTICIA
                        Estima la categorΎa de un documento dado
  -tn, --testear_multiples_noticias
                        Estima las categorΎas de varios documentos a la vez
```

*Salida del parámetro -h*

#### 5.2.1.1-Dividir noticias de entrada

Prepara las noticias para entrenar y testear el modelo a partir de las noticias guardadas en *archivos/noticias.csv* y categorizadas con los datos del archivo *archivos/categorias.csv*

Si se quieren añadir noticias, tan solo hay que introducirlas una por línea en formato de texto plano en el archivo correspondiente e indicar su categoría en la misma línea en el archivo de categorías.

Para dividir las noticias iniciales en conjuntos de entrenamiento y prueba, se usan dos parámetros: -d y -p

-d : indica al programa que debe realizar la división de noticias. Si no se acompaña del parámetro -p usa todas las noticias suministradas para entrenar el modelo.

-r *ratio* : se usa conjuntamente con -d y establece el ratio de noticias del total que se usarán para entrenar el modelo, dejando las restantes para testearlo.

```
(C:\Users\admin\Anaconda3) C:\Users\admin\Desktop\kNN>python knn-cli.py -d
El n·mero de noticias de entrenamiento es: 119
El n·mero de noticias de test es: 0
El ratio de noticias de entrenamiento es: 0.99999
```

*Uso del parámetro -d sin establecer un ratio manualmente*

```
(C:\Users\admin\Anaconda3) C:\Users\admin\Desktop\kNN>python knn-cli.py -d -r 0.8
El n·mero de noticias de entrenamiento es: 96
El n·mero de noticias de test es: 23
El ratio de noticias de entrenamiento es: 0.8
```

*Uso del parámetro -d con un ratio de entrenamiento de 0.8*

#### 5.2.1.2-Entrenar modelo

La creación del modelo se hace a partir de los datos generados en el paso anterior.
Para ejecutarla, es necesario introducir el parámetro -e

```
(C:\Users\admin\Anaconda3) C:\Users\admin\Desktop\kNN>python knn-cli.py -e
Modelo entrenado con 96 noticias de entrenamiento.
```

*Uso del parámetro -e*

#### 5.2.1.3-Testear modelo

Para comprobar la fiabilidad del modelo, se usa el modelo creado en el paso anterior y las noticias de prueba guardadas en el primer paso, ya que como se conoce su categoría, permite comparar con la categoría que el algoritmo estima para cada una de ellas, informando de la fiabilidad del algoritmo.

La ejecución de este paso necesita de dos parámetros:
-tn : indica que se testearán múltiples noticias.
-k *k_vecinos* : indica el número de vecinos con el que se comparará cada texto.

```
(C:\Users\admin\Anaconda3) C:\Users\admin\Desktop\kNN>python knn-cli.py -tn -k 4
El n·mero de documentos categorizados es:  23
El n·mero de documentos categorizados correctamente es:  22
El porcentaje de acierto es: 95.65217391304348 %
```

*Uso del parámetro -tn con k=4*

Además es útil para comprobar cómo pueden cambiar los resultados al modificar el valor del parámetro k

```
(C:\Users\admin\Anaconda3) C:\Users\admin\Desktop\kNN>python knn-cli.py -tn -k  8
El n·mero de documentos categorizados es:  23
El n·mero de documentos categorizados correctamente es:  20
El porcentaje de acierto es: 86.95652173913044 %
```

*Uso del parámetro -tn con k=8*

Hay que tener en cuenta que el número de vecinos no puede ser superior en ningún caso al número de noticias usado para entrenar el modelo.

```
(C:\Users\admin\Anaconda3) C:\Users\admin\Desktop\kNN>python knn-cli.py -tn -k 500
El k es demasiado grande, no puede ser mayor que  96
```

*Uso del parámetro -tn con un k mayor al número de noticias usadas para entrenar el modelo*

El número de vecinos siempre debe ser mayor a 0, para que el algoritmo pueda tener alguna referencia con la que comparar cada noticia testeada.

```
(C:\Users\admin\Anaconda3) C:\Users\admin\Desktop\kNN>python knn-cli.py -tn -k 0
El parßmetro k debe ser mayor que cero(0).
```

*Uso del parámetro -tn con un k demasiado bajo*

#### 5.2.1.4-Categorizar una noticia

Si se quiere que el algoritmo clasifique una noticia en una categoría, es necesario usar los siguientes parámetros:

-t *"ruta"* : indica que se categorizará la noticia guardada en el archivo indicado por su ruta relativa a la raíz del proyecto.
-k *k_vecinos* : indica el número de vecinos con el que se comparará la noticia.

Con el programa se incluyen 6 archivos de prueba (uno de cada categoría) dentro de la carpeta *archivos* el nombre de estos archivos son:

test_ciencia.csv Contiene una noticia de la categoría "Ciencia".
test_cultura.csv Contiene una noticia de la categoría "Cultura".
test_deporte.csv Contiene una noticia de la categoría "Deporte".
test_economia.csv Contiene una noticia de la categoría "Economía".
test_sociedad.csv Contiene una noticia de la categoría "Sociedad".
test_tecnologia.csv Contiene una noticia de la categoría "Tecnología".

Para usar otro archivo, hay que tener en cuenta que este debe tener solo una línea en la cual se encontrará la noticia escrita en texto plano.

```
(C:\Users\admin\Anaconda3) C:\Users\admin\Desktop\kNN>python knn-cli.py -t "archivos/test_cultura.csv" -k 4
La categoría estimada del documento evaluado es:  Cultura
```

*Uso del parámetro -t con una noticia ya incluida*

```
(C:\Users\admin\Anaconda3) C:\Users\admin\Desktop\kNN>python knn-cli.py -t "archivos/noticia_de_prueba.csv" -k 4
La categoría estimada del documento evaluado es:  Tecnologia
```

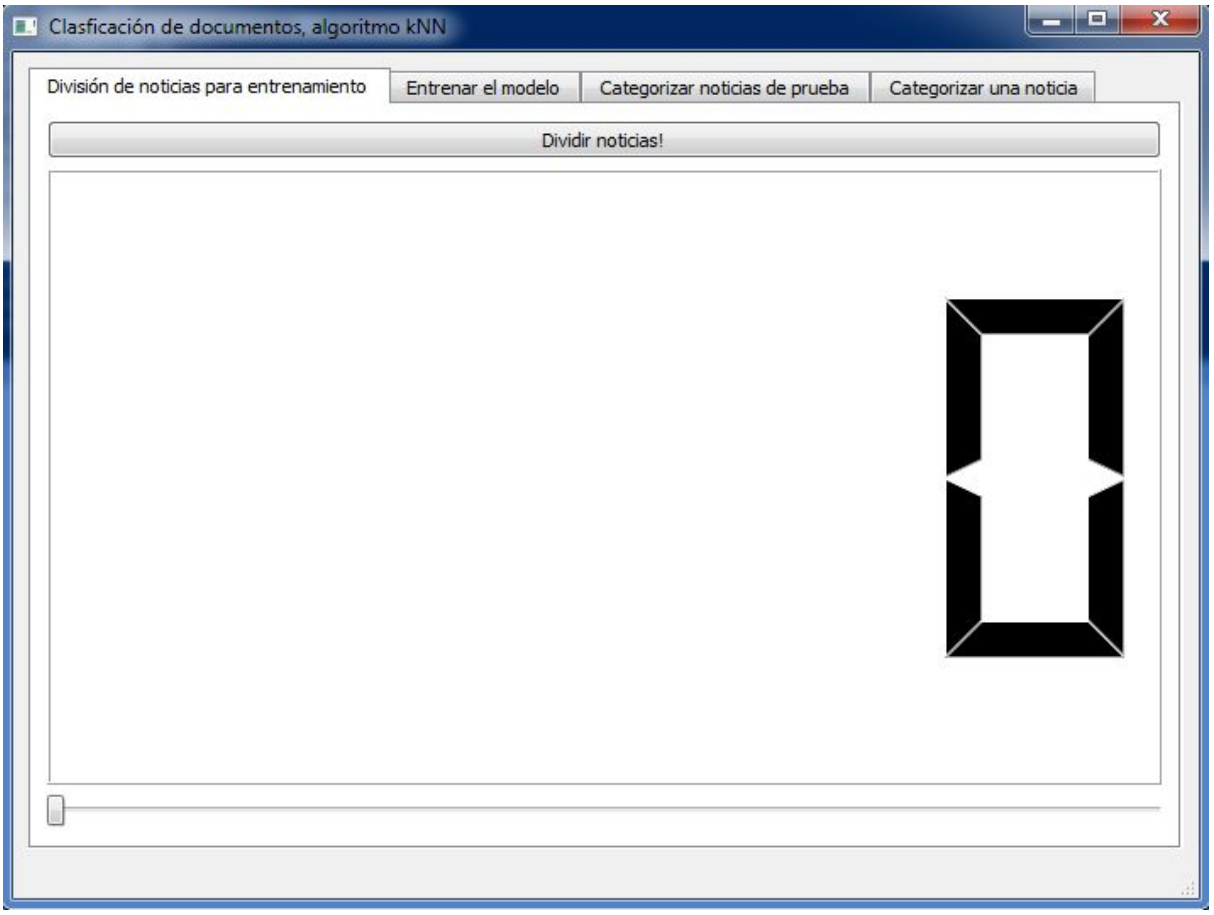
*Uso del parámetro -t con una noticia introducida por el usuario*

## 5.2.2- Interfaz gráfica

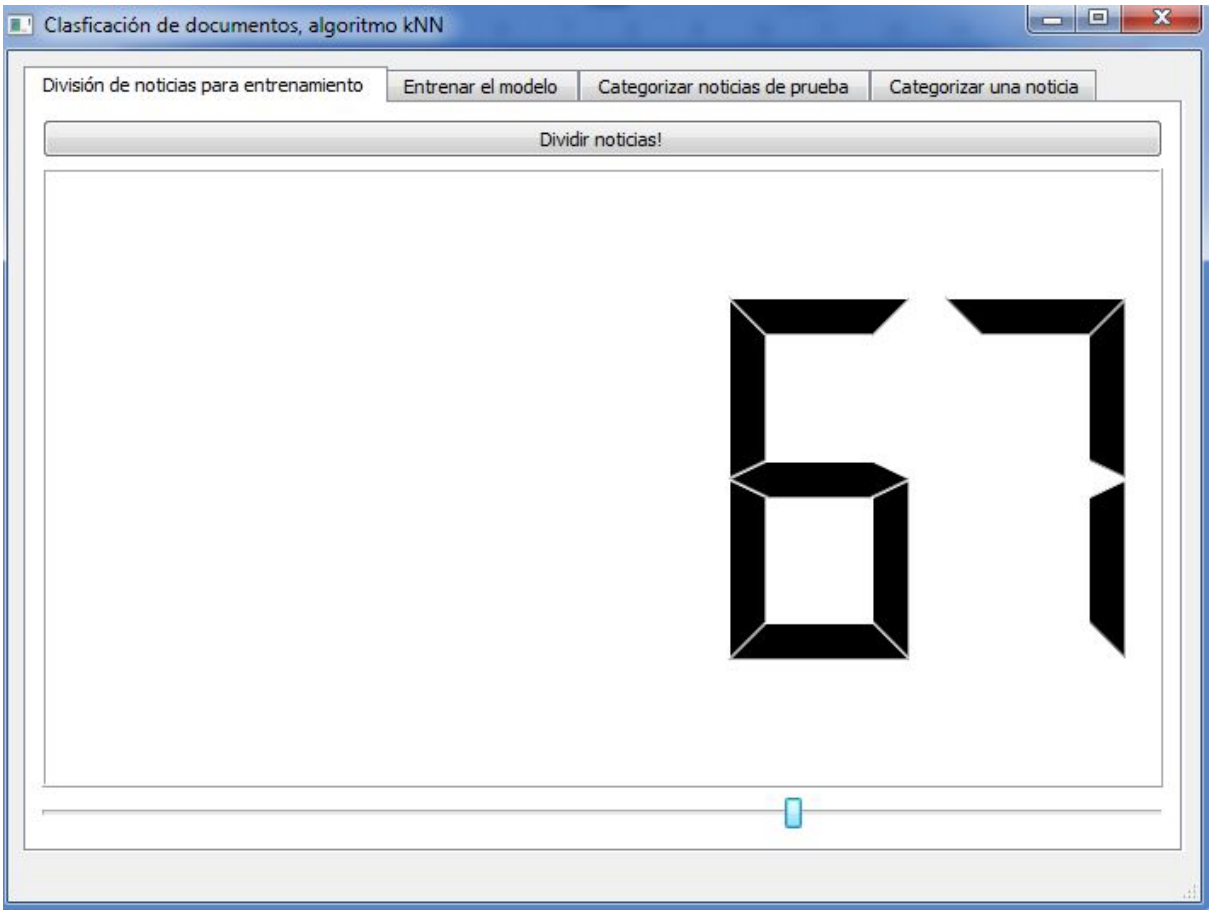
Para usar la interfaz gráfica hay que usar el archivo *knn-gui.py* ubicado en la raíz del proyecto. Para ejecutarlo, es necesario llamar el script con el intérprete de python.

#### 5.2.2.1-Dividir noticias de entrada

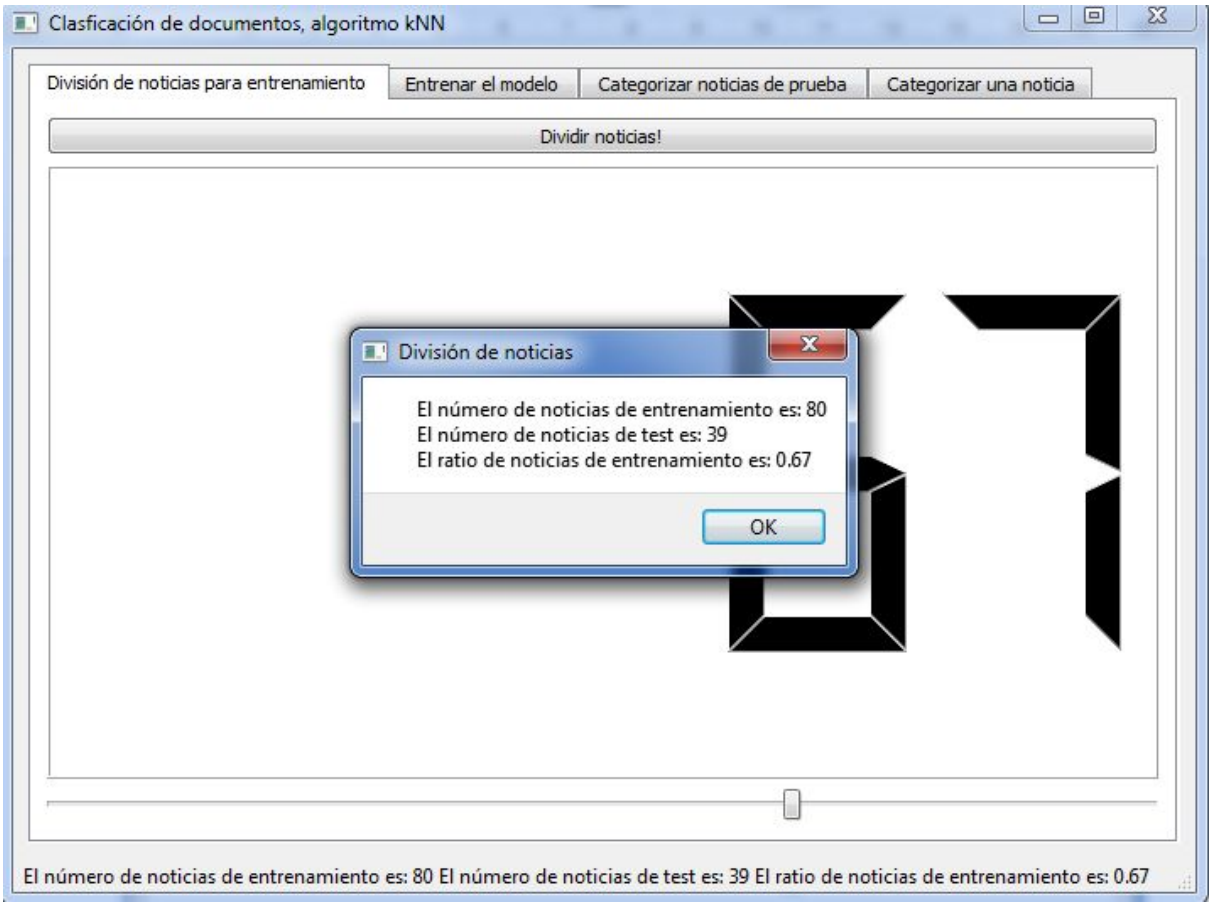
Se realiza en la pestaña "*División de noticias para entrenamiento*".
Para dividir las noticias de entrada, es necesario especificar el porcentaje de noticias destinadas a entrenar el modelo con la barra de la parte baja de la ventana y a continuación pulsar en el botón "*Dividir noticias!*"




*Vista inicial de la pestaña*



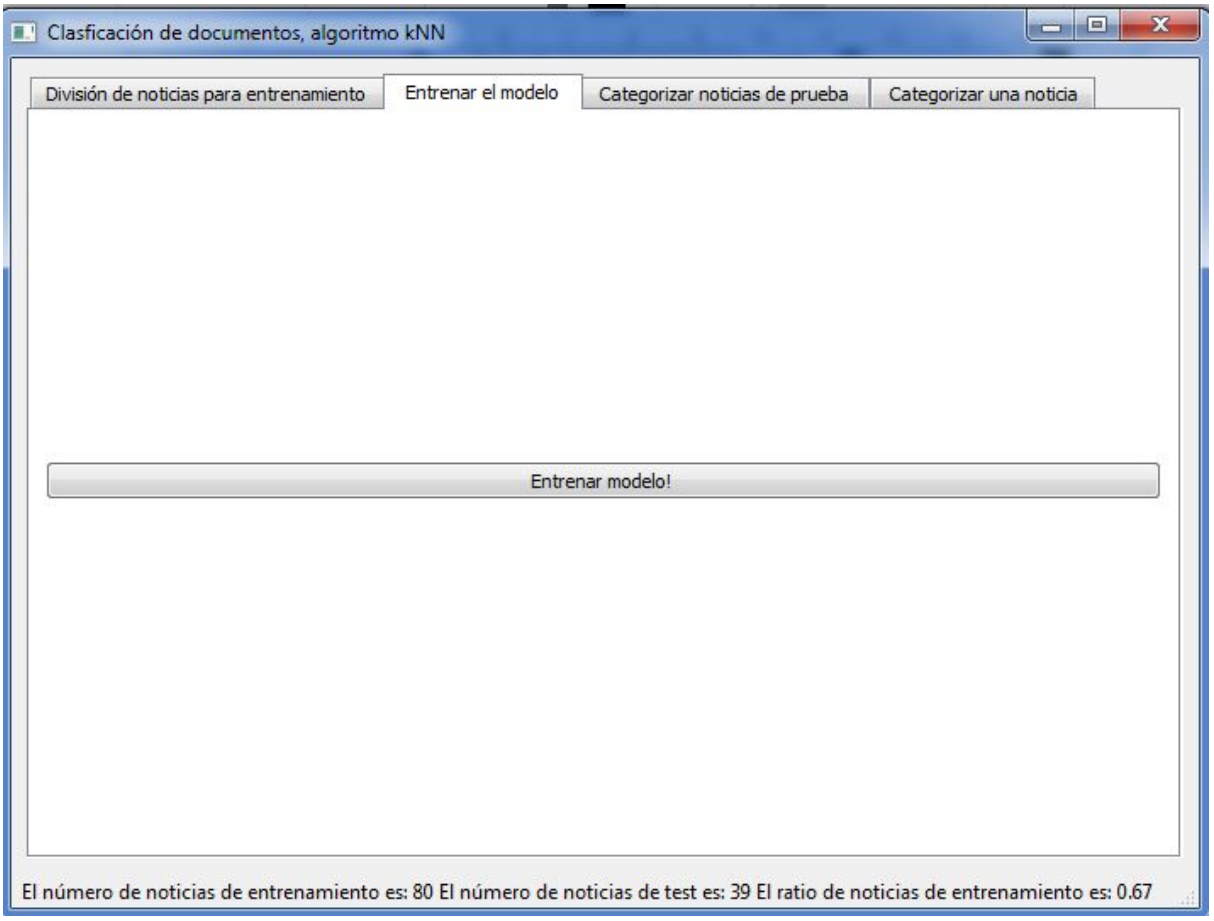

*Cambiando el porcentaje de noticias de entrenamiento*



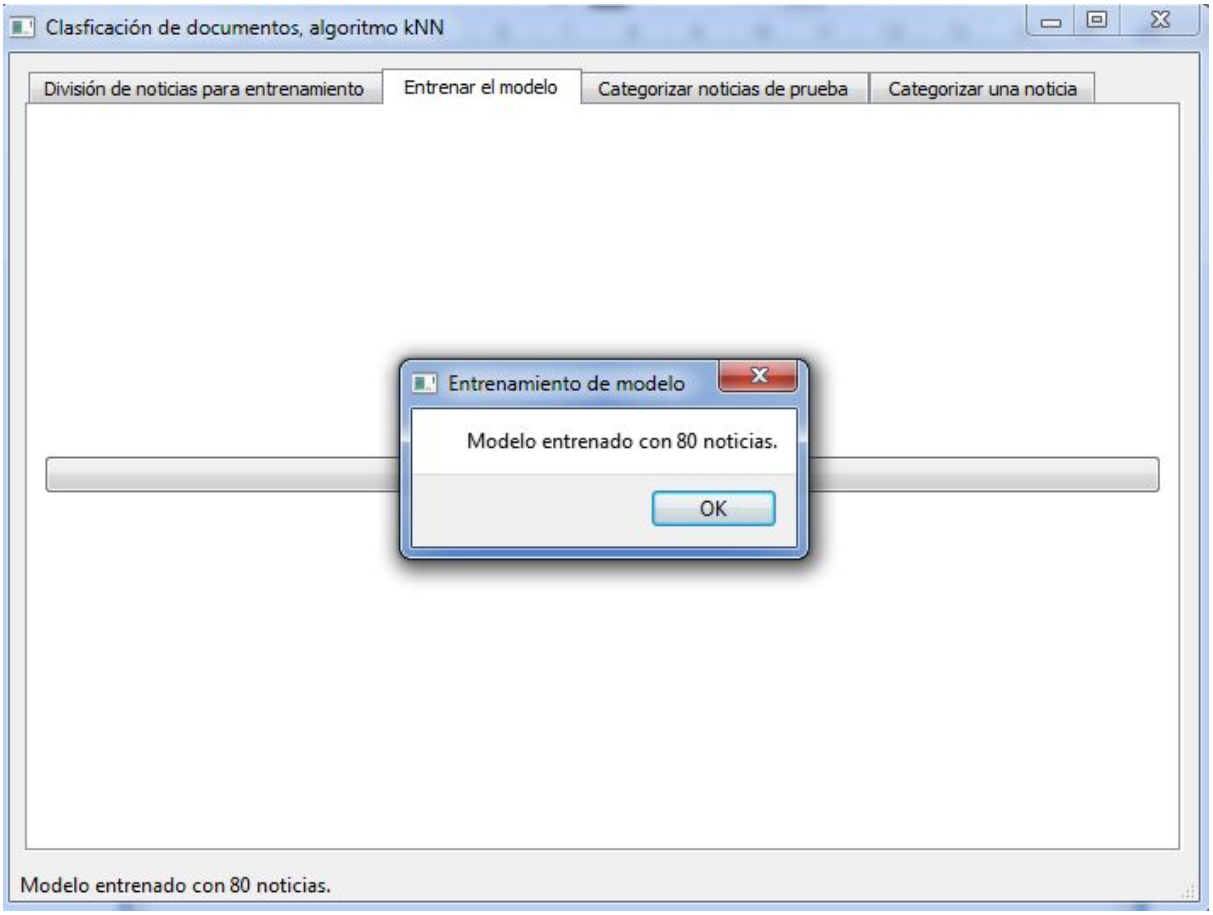

*Mensaje mostrado después de pulsar el botón*

#### 5.2.2.2-Entrenar modelo

Se realiza en la pestaña "*Entrenar el modelo*".
Para crear el modelo, es necesario pulsar en el botón "*Dividir noticias!*"



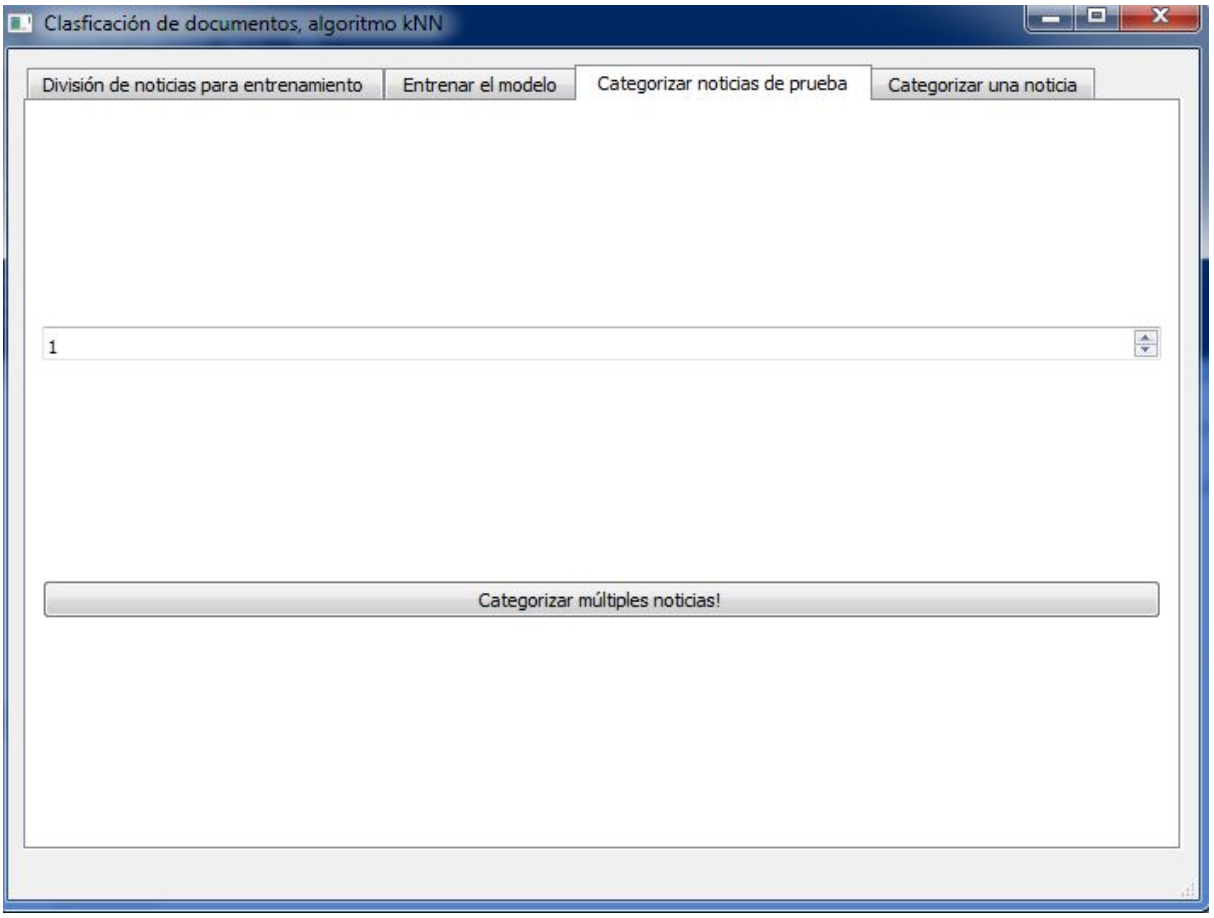

*Vista inicial de la pestaña*



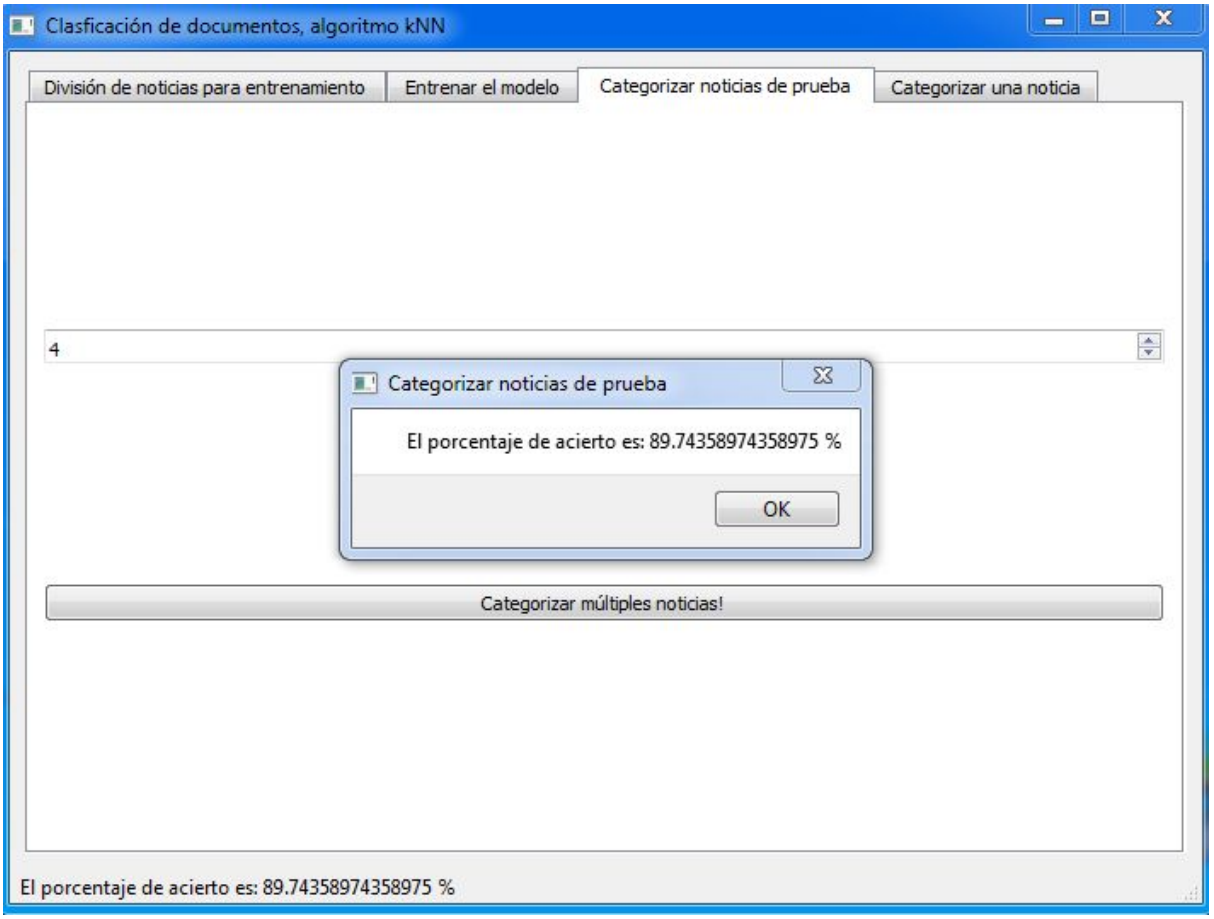

*Mensaje mostrado después de pulsar el botón*

#### 5.2.2.3-Testear modelo

Se realiza en la pestaña "*Categorizar noticias de prueba*".
Para ejecutarlo, es necesario establecer el número de vecinos con el que comparar cada noticia y luego pulsar en el botón "*Categorizar múltiples noticias!*".

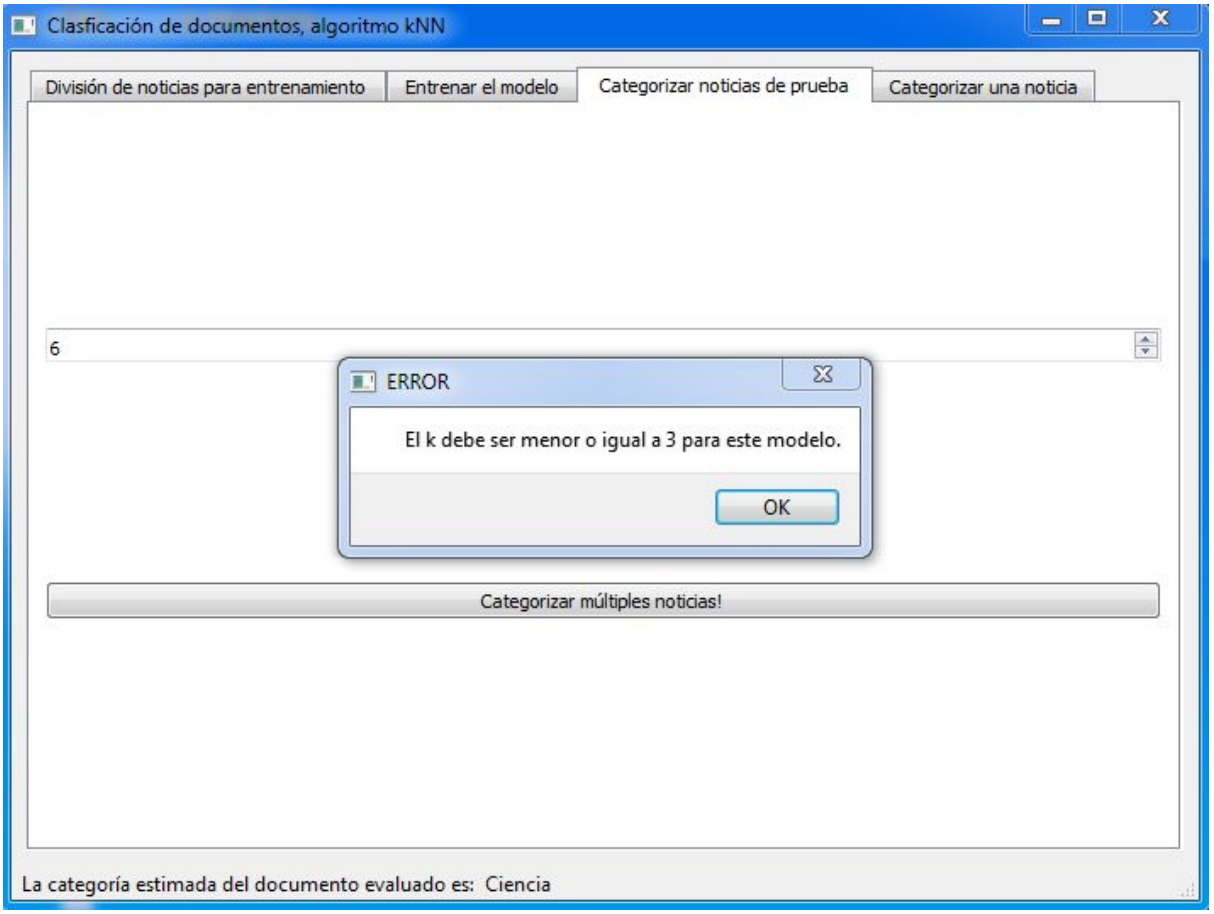



*Vista inicial de la pestaña*




*Mensaje mostrado después de pulsar el botón*

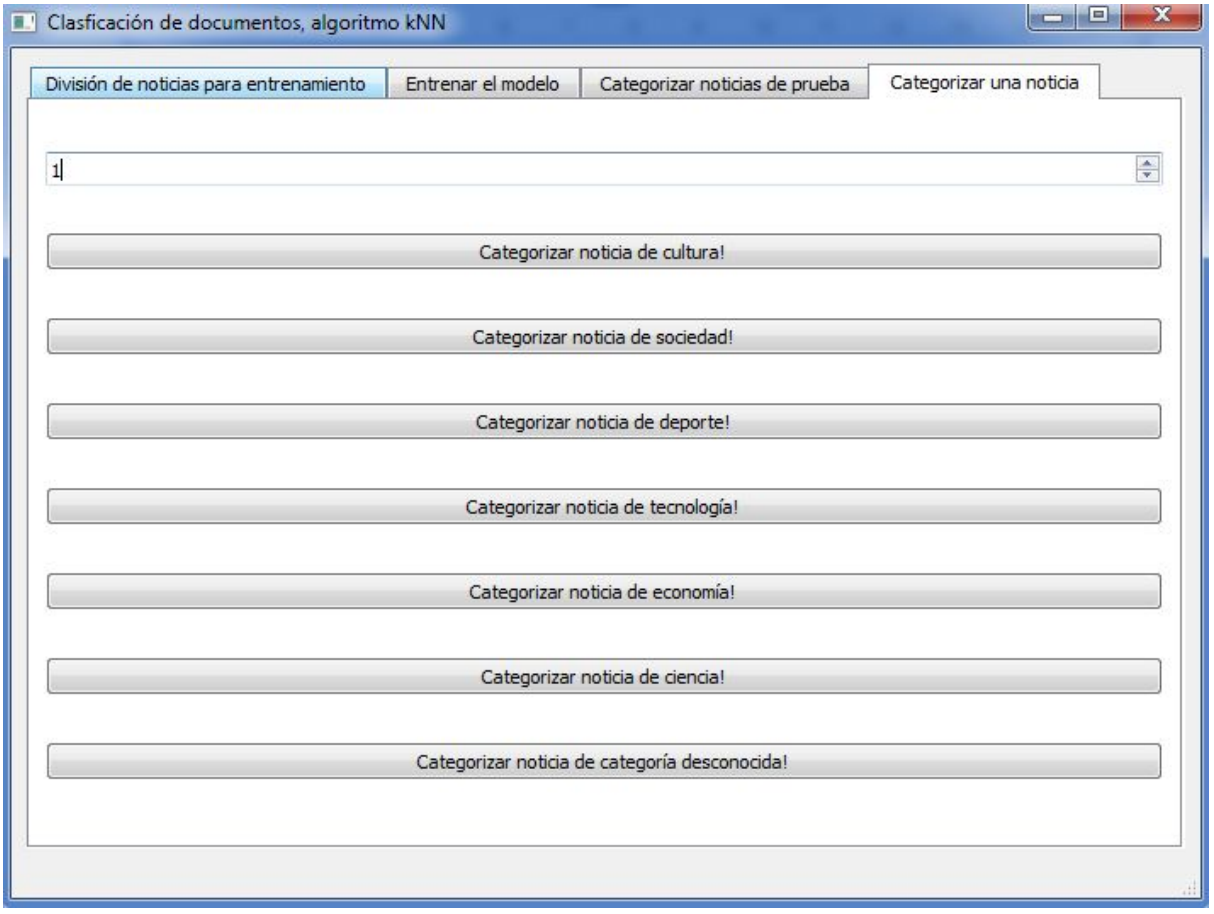
Hay que tener en cuenta que el valor de k no puede ser superior al número de noticias usadas para entrenamiento, si se introduce un k incorrecto, el programa mostrará un mensaje de error.



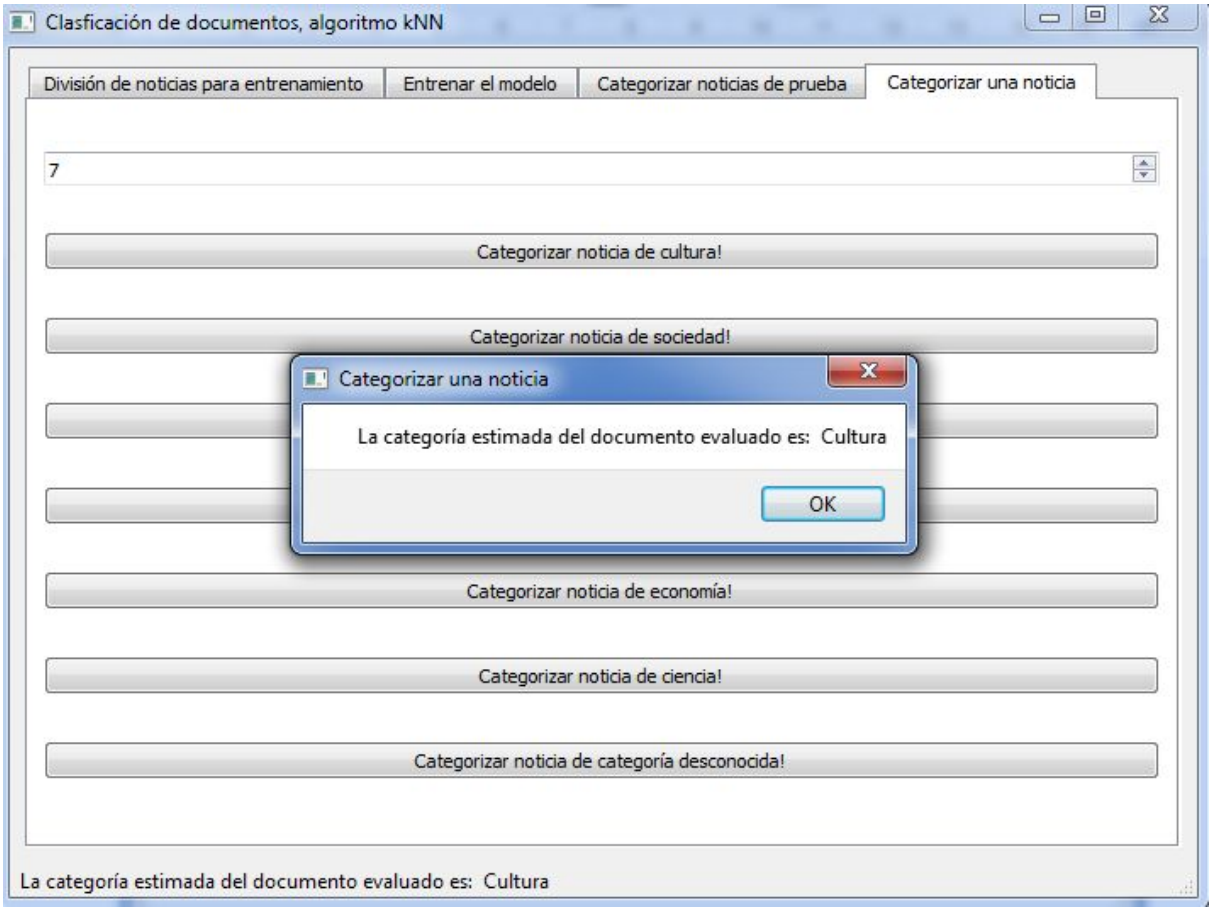

*Mensaje de error mostrado tras introducir un k incorrecto para el modelo*

#### 5.2.2.4-Categorizar una noticia

Se realiza en la pestaña "*Categorizar una noticia*".
Para ejecutarlo, es necesario establecer el número de vecinos con el que comparar cada noticia y luego pulsar en el botón correspondiente a la noticia que se quiera categorizar.
Los primeros 6 botones testean una noticia de una categoría conocida. El último botón "*Categorizar noticia de categoría desconocida!* " permite estimar la categoría de una noticia introducida por el usuario. Dicha noticia debe estar guardada en el fichero "*archivos/test_usuario.csv*" en una única línea como texto plano.




*Vista inicial de la pestaña*




*Mensaje mostrado después de pulsar el botón "Categorizar noticia de categoría desconocida!*"

# 6-CONCLUSIONES

Gracias a este proyecto, hemos podido comprobar de una forma satisfactoria el funcionamiento de dos algoritmos muy importantes dentro del ámbito de la clasificación de documentos en un entorno parecido al real.

Además, nuestro manejo del lenguaje python y la adecuación a su filosofía ha sido satisfactoria, además de enriquecedora, gracias a poder haber aprendido un lenguaje nuevo.

En cuanto a los resultados de los algoritmos implementados por nosotros mismos, hemos comprobado que funcionan de una forma correcta, llegando incluso a funcionar, con un conjunto de entrenamiento suficiente, con una efectividad del 100%. Por tanto, consideramos que el objetivo de implementarlos correctamente está satisfecho.

Así mismo, hemos desarrollado una interfaz de usuario gráfica y una por comandos para facilitar el uso del programa implementado en el proyecto, lo cual nos ha llevado a investigar acerca de diferentes librerías de Python, entre ellas, la implementación para Python de Qt5.