

---

# Práctica 4

*Práctica final*

---



Jose Ignacio del Valle Bustillo  
Ricardo Delgado Fernández  
Miguel Ramos López  
Pablo Pascual García

## Índice

<b>Abstract</b>	<b>4</b>
<b>Introducción</b>	<b>4</b>
Contextualización	4
Qué son los delitos de odio	4
Tipos de delito de odio	6
Perfil de la víctima y el autor	7
<b>Dominio del proyecto</b>	<b>8</b>
<b>Problema a resolver</b>	<b>9</b>
<b>Solución al problema</b>	<b>10</b>
Objetivos	10
Tecnologías / librerías empleadas	11
Esquema gráfico de la arquitectura / organización del proyecto	12
Descripción de los datos de entrada	12
Datos de salida	13
<b>Descripción Proceso en Python</b>	<b>15</b>
Procesamiento de lenguaje natural (PLN)	15
Fase Entrenamiento	19
Fase Testeo/Ejecución	23
<b>Diseño</b>	<b>25</b>
Fuentes de datos y dataset generado	25
Diagrama UML de clases	26
<b>Metodología de trabajo</b>	<b>27</b>
Planificación	27
Roles	28
<b>Presupuesto</b>	<b>29</b>
<b>Diseño de pruebas y resultado de las mismas</b>	<b>30</b>
<b>Manual de instalación</b>	<b>32</b>
Requirements.txt	36
<b>Manual de usuario</b>	<b>37</b>
Fase de Entrenamiento	37

---

1. Seleccionar Noticias de Odio	38
2. Seleccionar Noticias de No Odio	38
3. Seleccionar Algoritmo de ML	39
4. Vista previa de los datos	39
5. Gráfica de los datos	40
6. Guardar Modelo	40
7. Resultados Entrenamiento	41
8. Tabla Predicciones	41
Fase de Testeo	41
1. Seleccionar Noticias sin Categorizar	42
2. Seleccionar Modelo Entrenado	43
3. Tabla de Resultados	43
4. Selección noticias	44
5. Vista previa del contenido	44
6. Gráfica Resultados	45
7. Tipo de Fichero	45
8. Guardar Resultados	46
<b>Conclusiones</b>	<b>46</b>
<b>Trabajos futuros</b>	<b>47</b>
<b>Bibliografía</b>	<b>48</b>

## Abstract

En este proyecto explicaremos cómo se ha realizado un clasificador de noticias que diferenciará entre noticias de odio y de “no odio”. Dicha aplicación la realizaremos en el lenguaje Python y para realizar la interfaz utilizaremos la librería Streamlit, que nos proporciona una interfaz web interactiva.

Para realizar esta clasificación utilizaremos varios modelos de machine learning y describiremos cómo se han llevado a cabo cada uno de ellos, realizando al final una comparativa entre ellos y sacando conclusiones a partir de diferentes parámetros obtenidos en los resultados como el recall o la precisión, entre otros.

**Resultado:** El objetivo del proyecto es realizar un proceso de clasificación de textos mediante diferentes algoritmos ML de clasificación de texto con sus diferentes resultados aplicados a unas noticias concretas de testeo que el clasificador desconoce.

**Palabras clave:** Python | Streamlit | Machine Learning | Training | Testing | Gradient Boosted Tree | Decision Tree | Support Vector Machine

## Introducción

Para abordar este proyecto lo primero que debemos hacer es contextualizar el tema de delitos de odio explicando la importancia que tienen en la sociedad junto a datos de relevancia.

### Contextualización

#### Qué son los delitos de odio

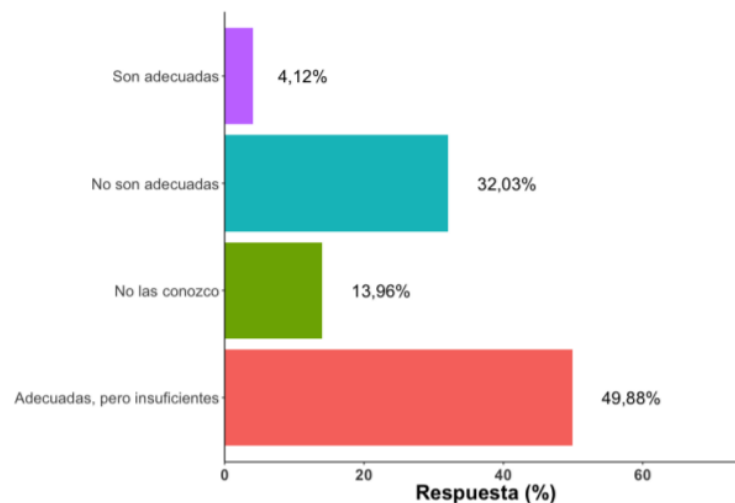
Cuando se habla de delitos de odio se debe comprender bien su significado. Un delito de odio es una expresión (o varias) que induce a una persona de forma directa por motivos de odio o discriminación dados por la apariencia, ideologías u orientación sexual de la víctima. (Qué Es Un Delito De Odio, n.d.)

Además, según la Organización para la Seguridad y la Cooperación en Europa (OSCE) una de las definiciones de delitos de odio sería: *“toda infracción penal, incluidas las cometidas contra las personas o la propiedad, dónde el bien jurídico protegido, se elige por su, real o percibida, conexión, simpatía, filiación, apoyo o pertenencia a un grupo. Un ‘grupo’ se basa en una característica común de entre sus miembros, real o percibida, como su raza, origen nacional o étnico, lengua, color de la piel, religión, edad, discapacidad, orientación sexual u otro factor similar”*.

Para conocer lo que suponen los delitos de odio para la sociedad obtenemos varios datos de una encuesta realizada sobre estos mismos con un tamaño muestral de 437, combinando personas de diferente sexo, situación laboral, provincia, lugar de nacimiento, estudios y edad.

Expondremos algunos de los datos que nos serán de mayor relevancia:

- ¿Son adecuadas las medidas aplicadas en España contra los delitos de odio?



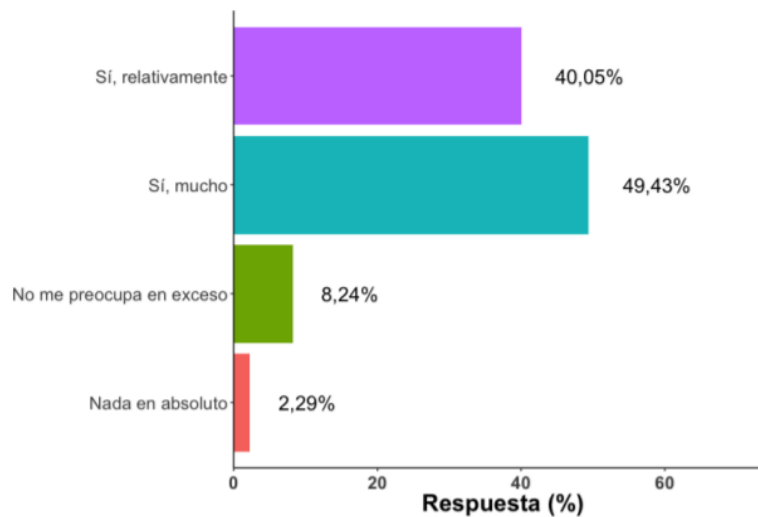
(Ministerio de Interior, 2021; Gráfica 5)

- ¿Qué piensan sobre los delitos de odio durante los últimos 5 años?

Respuesta	Número de respuestas	Frecuencia (%)
Han aumentado notablemente	261	59,73
Han aumentado ligeramente	84	19,22
No han variado	38	8,70
No lo sé	26	5,95
Han disminuido ligeramente	14	3,20
Han disminuido notablemente	14	3,20
<b>TOTAL</b>	<b>437</b>	<b>100</b>

(Ministerio de Interior, 2021; Tabla 5)

- ¿Crees que puedes ser víctima de un delito de odio?



(Ministerio de Interior, 2021; Gráfica 6)

### Tipos de delito de odio

Según la siguiente imagen, la mayoría de delitos de odio que se han dado en este último año son hacia personas con discapacidad, sexo/orientación sexual o hacia personas con algún tipo de enfermedad.

	2019	2020	% Variación	
Antisemitismo	5	3	-40	
Aporofobia	12	10	-16,7	
Creencias o prácticas religiosas	66	45	-31,8	
Persona con discapacidad	26	44		69,2
Orientación sexual e identidad de género	278	277	-0,4	
Racismo/xenofobia	515	485	-5,8	
Ideología	596	326	-45,3	
Discriminación por sexo/género	69	99		43,5
Discriminación generacional	9	10		11,1
Discriminación por enfermedad	8	13		62,5
Antigitanismo	14	22		57,1
Total delitos	1.598	1.334	-16,5	
Resto de incidentes	108	67	-38,0	
Total delitos e incidentes de odio *	1.706	1.401	-17,9	

(\*) Todos los delitos han bajado durante la pandemia

Imagen 2. Representa los tipos de delitos de odio acontecidos en España en 2019 y 2020.

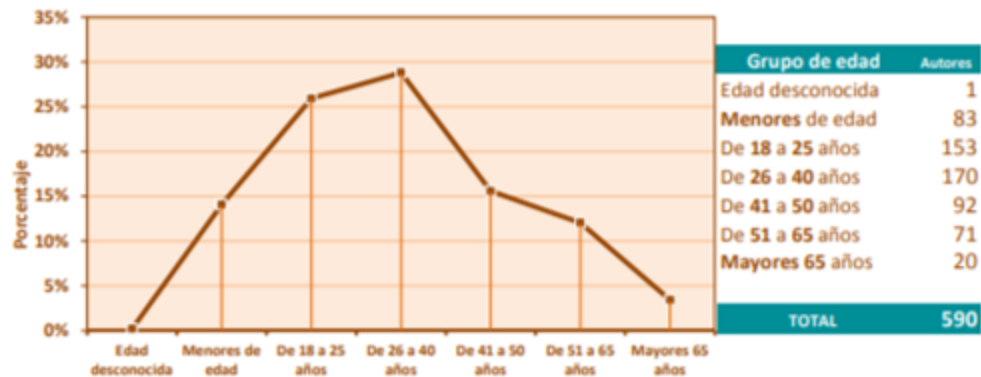
Fuente: El País (Digital)

Además, tal y como se muestra en el Informe sobre incidentes relacionados con delitos de odio de 'El Mundo' (Meyer, 2015), 4 de cada 10 delitos que son registrados son por orientación sexual y en el 25% de los casos son a menores. El total de incidentes que se registran ha variado notablemente respecto a años anteriores pudiéndose observar una menor sensibilización por los demás que en anteriores momentos.

### Perfil de la víctima y el autor

Según el Ministerio del Interior (*Estadísticas*, n.d.), el perfil del agresor medio es el siguiente: varón de entre 26 y 40 años que actúa sobre víctimas por identidad de género o racismo. Además, se pueden corroborar estas palabras según el siguiente gráfico:

>> EDAD DE LOS AUTORES



*Imagen 3. Representa la edad y el porcentaje de delitos de odio de los usuarios.*

*Fuente: Informe de la evolución de delitos de odio en España.*

Como hemos podido observar con estos hechos e ilustraciones, los delitos de odio se han incrementado y han ampliado su rango a las redes sociales. Cada año que pasa se incrementan estos tipos de delitos y no todos son debidamente resueltos. La libertad de expresión es esencial y es un derecho, pero siempre desde el debido respeto hacia los demás. No deben realizarse grandes restricciones, pero según múltiples estatutos públicos sí que hay que castigar estos delitos de odio que se cometen cada día.

## Dominio del proyecto

Partimos del punto en el que ya tenemos diversas noticias descargadas tras el proceso ETL de la práctica anterior, extraídas mediante Pentaho. Usaremos toda esa gran cantidad de noticias, de odio y de otras secciones y categorías, para poder entrenar y testear nuestros modelos.

Lo primero que debemos hacer es procesar y normalizar el texto de forma que después nuestro algoritmo de aprendizaje automático sea lo más efectivo posible, para ello tokenizaremos el texto en las diferentes palabras, las pasaremos a minúsculas todo, aplicaremos una lista de parada y finalmente normalizamos las palabras morfológicamente mediante stemming. Es decir, realizaremos lo conocido como procesamiento de lenguaje natural.



Una vez procesado el texto tendremos que aplicar algoritmos de aprendizaje automático o machine learning, en nuestro caso los elegidos han sido:

- Gradient Boosted Tree
- Decision Tree
- Support Vector Machine

En este punto, se realizará una descripción detallada de los modelos, el beneficio de usar cada uno y a qué están enfocados.

- **Gradiente Boosted Tree:** Este algoritmo de ML funciona mediante la construcción de diferentes modelos predictivos algo más simples (también catalogados como débiles) donde secuencialmente cada modelo intenta predecir el error dejado por el anterior por lo que suele sobre ajustarse bastante rápido. Este modelo funciona ligeramente mejor que otras predicciones que se pueden realizar de manera algo más aleatoria. (*Gradient Boosting*, n.d.)
- **Decision Tree:** Es un modelo de ML que realiza decisiones en base a diferentes variables teniendo en cuenta sus posibles consecuencias. Se representa en forma de árbol. (*Árbol De Decisión En Machine Learning (Parte 1)*, 2019)
- **Support Vector Machine:** Es otro algoritmo de ML que actúa como una caja negra proporcionando una salida a una entrada dada. Puede ser usado no solo para clasificación, sino también para problemas de regresión. (*Support Vector Machine — Introduction to Machine Learning Algorithms* | by Rohith Gandhi, n.d.)

## Problema a resolver

Antes de resolver el problema se debe saber que previamente hemos recogido diversa información sobre noticias de odio y “no odio” mediante la herramienta Pentaho. Con dicha herramienta hemos recogido estructuras como el título, entradilla, texto, fecha y autor/autores y los hemos pasado a un documento de texto.

Con estos documentos de texto podremos diferenciar los dos grandes grupos con los ficheros con los que trabajaremos: Odio y No odio. Con estos dos grupos con los ficheros podremos hacer un clasificador de noticias mediante el lenguaje de programación “Python” y la librería “Streamlit” que nos dará una interfaz web interactiva.

La clasificación se basará en una fase de entrenamiento y posterior de testeo con una traza de noticias no entrenadas aún, con la cual dependiendo del algoritmo de machine learning que apliquemos nos indicará que noticias de esa traza son de “Odio” o “No odio”.

## Solución al problema

### Objetivos

ID	Categoría	Descripción
IF001	Interfaz	Diseñar una interfaz web interactiva donde se apliquen las funcionalidades de nuestro clasificador.
EN001	Entrenamiento	Permitir que el usuario pueda introducir diferentes noticias de Odio y No odio para entrenar un modelo de ML.
EN002	Entrenamiento	Hacer que el usuario pueda elegir entre nuestros 3 algoritmos de ML para entrenar el modelo.
VI001	Visualización	Proporcionar una visualización de las noticias que ha escogido de cada tipo, las total y algoritmo seleccionado.
VI002	Visualización	Diseñar una gráfica que muestre las noticias de Odio y No odio elegidas.
EN003	Entrenamiento	Permitir al usuario guardar el modelo entrenado.
VI003	Visualización	Mostrar al usuario las métricas del entrenamiento como la precisión, el accuracy, recall y la matriz de confusión.
TE001	Testeo	Hacer que el usuario pueda introducir diferentes noticias de Odio y No odio para clasificarlas.
TE002	Testeo	Permitir que el usuario introduzca el modelo entrenado guardado y entrenado previamente.
VI004	Visualización	Enseñar en formato tabla el resultado de la clasificación de cada

		noticia.
VI005	Visualización	Poder visualizar una noticia seleccionada en un campo de texto.
VI006	Visualización	Mostrar los resultados de la clasificación en un diagrama de tartas.
TE003	Testeo	Dar al usuario la opción de guardar los resultados en formato CSV, EXCEL o TXT.

*Tabla 1. Tabla de objetivos específicos.*

## Tecnologías / librerías empleadas

En cuanto al lenguaje de programación por el que hemos optado es:

**Python:** es el lenguaje de programación que nos permitirá realizar las funcionalidades de nuestro programa, combinándolo con diferentes librerías como por ejemplo Streamlit que nos dará la interfaz web. (*Python*, n.d.)

Entre las librerías más relevantes que hemos utilizado están:

**Streamlit:** es una librería que nos proporciona una interfaz web. Está orientada al análisis de datos y visualización de los mismos de una manera fácil e intuitiva por lo que nos será de gran ayuda en este proyecto. (Richards, n.d.)

**Nltk:** es una conjunto de librerías que nos ayudarán a realizar el procesamiento de lenguaje natural de los textos de las noticias. (Bird, n.d.)

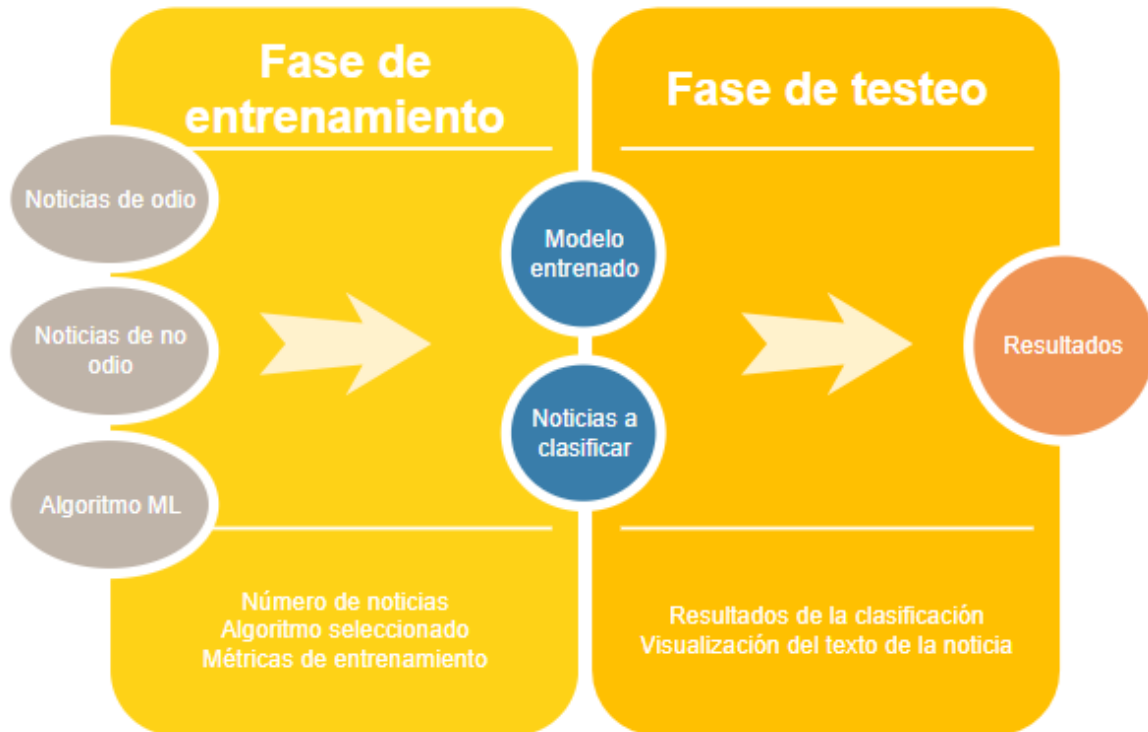
**Sklearn/ Scikit-learn:** biblioteca que incluye varios algoritmos de machine learning de los cuales utilizaremos el support vector machine, decision tree y gradient boosted tree. (Cournapeau, n.d.)

**Plotly:** biblioteca que nos ayudará a la impresión y visualización de los datos obtenidos de forma gráfica. (*Plotly*, n.d.)

**PyStemmer:** Proporciona acceso a algoritmos eficientes para calcular la forma “stemmed” de una palabra. (Boulton, n.d.)

**Pandas:** librería especializada en el manejo y análisis de estructuras de datos. (McKinney, n.d.)

Esquema gráfico de la arquitectura / organización del proyecto



*Imagen 4. Esquema gráfico de la arquitectura del proyecto.*

## Descripción de los datos de entrada

Los datos de entrada se corresponden con los ficheros de texto que incluyen el contenido de cada noticia. En el proyecto se dividen dos directorios, uno que contiene las noticias de odio y otro las de no odio.

La estructura del fichero de texto es la siguiente:

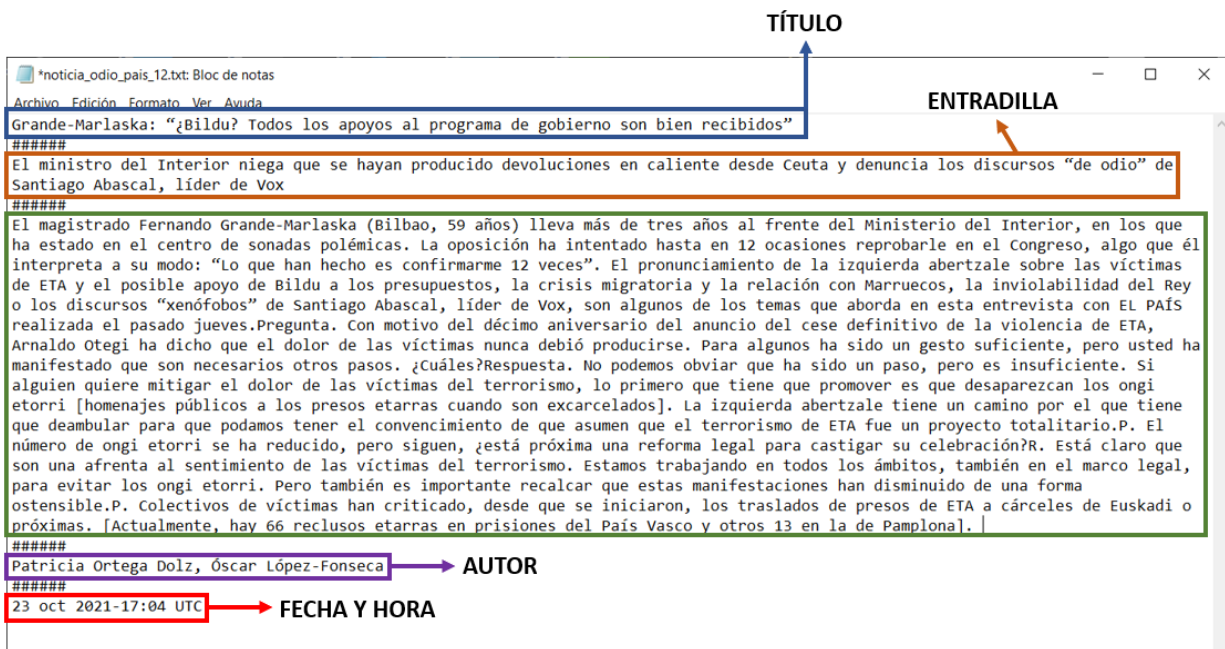


Imagen 5. Estructura y elementos de noticias guardadas.

Cada apartado está dividido con un separador (#####) para distinguir dónde empieza y acaba cada campo.

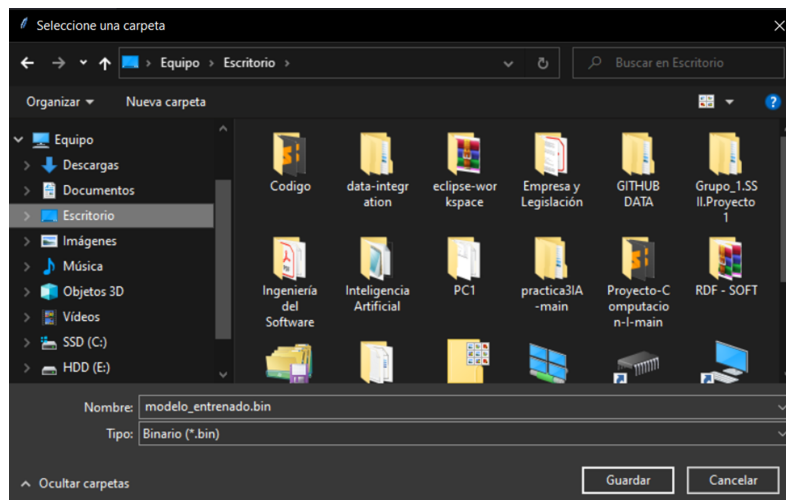
Todos los datos de entrada han sido recopilados de un proceso de webscraping realizado con la herramienta de Pentaho.

## Datos de salida

Como datos de salida tenemos:

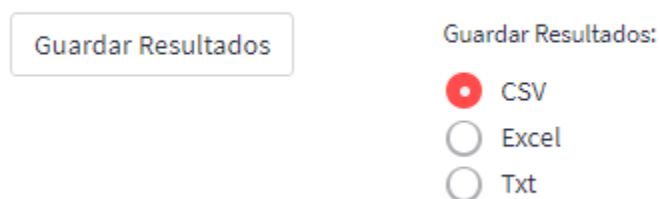
- Modelo entrenado en formato .bin
- Resultados de la fase de testeo en formato (txt, excel o csv)

El modelo entrenado se obtiene una vez introducidos los datos de entrada mencionados anteriormente y seleccionado el modelo de Machine Learning que queremos aplicar. Se nos pedirá elegir la ubicación donde queremos guardar el archivo de la siguiente forma:

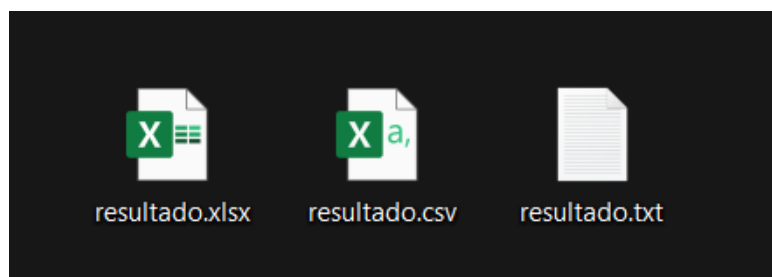


*Imagen 6. Modelo entrenado, guardado en Escritorio.*

Una vez seleccionado el modelo automáticamente se generarán los resultados de la fase de testeo. Otro elemento que se corresponde con los datos de salida, serían los ficheros de tipo csv, excel o txt, ambos se pueden generar a través de la interfaz en la fase de testeo



Los datos de salida se corresponden con los siguientes ficheros:



*Imagen 7. La salida de resultados, en 3 formatos distintos.*

## Descripción Proceso en Python

Para entender el proceso es importante conocer los tres tipos de ficheros .py que tenemos en el proyecto:

`main.py` : contiene la vista inicial de la aplicación, nos permite elegir entre la fase de testeo y la de entrenamiento.

`fase_entrenamiento.py` : contiene los métodos necesarios para realizar la fase de entrenamiento.

`fase_testeo.py` : contiene los métodos necesarios para realizar la fase de testeo.

A continuación se detalla cada fase por independiente y el código de la misma.

### Procesamiento de lenguaje natural (PLN)

Para las fases de entrenamiento o testeo es necesario realizar un procesado de lenguaje natural del texto para posteriormente realizar dichas fases. Para realizarlo en Python es necesario hacer uso de librerías como “`nltk`” o “`pystemmer`”. A continuación se detalla el proceso que se deberá de aplicar para realizar este procesado:

#### 1. Función “`generar_coleccion(lista_textos: list)`”

Esta función agrupará a todas las posteriores que se describen, pues en conjunto realizan todas las fases a hacer durante un PNL y se agruparán en esta función. Entre los procesos para realizar el PNL están:

- Pasar el texto a minúsculas
- Tokenización del texto
- Aplicar lista de parada
- Reducir tokens a stem

La función nos devolverá una lista con los textos en formato String ya correctamente procesados y así tendríamos la lista de noticias procesada y preparada para ser empleada en la fase de entrenamiento o testeo.

```
def generar_coleccion(lista_textos: list):  
  
    coleccion = []  
  
    for texto in lista_textos:  
  
        # Aplicamos tokenización del texto  
  
        tokens = tokenizar_texto(texto)  
  
        # Aplicamos lista de parada  
  
        lista_limpia = limpiar_texto(tokens)  
  
        # Aplicamos stemmer  
  
        texto = stemming(lista_limpia)  
  
        coleccion.append(texto)  
  
    return coleccion
```

### 1.1 Función “tokenizar\_texto(texto)”

Recordemos que la tokenización es el proceso de descomponer el texto palabras, símbolos u otros elementos en tokens. El objetivo de la tokenización es recoger todas las palabras del texto para posteriormente poder procesarlas en los algoritmos de aprendizaje automático. En la recuperación de información se necesitan las palabras del conjunto de datos, así que requiere un parser que procese la tokenización de los documentos y con esta funcionalidad dividiremos el documento de texto en tokens.

En esta función realizamos una tokenización del texto, para ello a la función le pasamos el texto de las noticias que ya han sido cargadas previamente y se le aplica el método “word\_tokenize” perteneciente a la librería nltk, pasándolas además a minúsculas con la función “lower()” y aplicando la tokenización en español. Esta función nos devolverá los tokens en una lista que la hemos llamado como “lista\_tokens”.

```
def tokenizar_texto(texto):  
  
    lista_tokens = word_tokenize(str(texto.read()),  
encoding="utf8").lower(), language="spanish")
```



```
return lista_tokens
```

## 1.2 Función “limpiar\_texto(lista\_tokens: list)”

Esta función nos permite aplicar a la lista anterior una lista de parada, con la que conseguiremos eliminar aquellas palabras que no nos interesan, por ejemplo porque se repiten con mucha frecuencia como preposiciones o conjunciones. Algunas de nuestras stopwords serían : #####, a, aquí, aún, con, cierto, como, cuando, ellos... Estas stopwords se corresponden con adverbios, conjunciones, preposiciones y otras palabras cuyo significado no nos aporta información relevante a la hora de la clasificación.

Una vez cargada nuestra lista de parada que recibe el nombre de “Lista\_Stop\_Words.txt”, la combinamos con una lista de caracteres especiales, pues tampoco nos interesan y mediante un bucle comparamos si las palabras están en esa lista y en el caso de que no estén se agregarán esas palabras a una nueva lista. La función nos devolverá una lista con las palabras que no se encuentren en la lista de parada, es decir, que no interesan para el posterior análisis.

```
def limpiar_texto(lista_tokens: list):  
  
    palabras = []  
  
    # Abrimos nuestra lista de parada  
  
    fichero_parada = open("Lista_Stop_Words.txt", "r", encoding="utf8")  
  
    # Hemos leído el fichero como un string y necesitamos una lista de  
    strings con las palabras a eliminar,  
  
    # para ello aplicamos la función split para separar el texto por los  
    espacios  
  
    lista_parada = fichero_parada.read().split("\n")  
  
    # Creamos una lista con los diferentes caracteres (!"#$$%&'()*+,-.  
    ./:;<=>?@[\\]^_`{|}~) y los añadimos a nuestra  
  
    # lista de parada, pues no nos son relevantes tampoco a la hora de  
    aplicar nuestro algoritmo de aprendizaje  
  
    puntuacion = list(string.punctuation)
```

```
lista_parada += puntuacion

# Aplicamos la lista de parada y nos quedamos solo con las
palabras/tokens que nos interesan

for palabra in lista_tokens:

    if palabra not in lista_parada:

        palabras.append(palabra)

return palabras
```

### 1.3 Función “stemming(lista\_palabras: list)”

Esta función nos ayudará a simplificar los tokens o palabras al stem. El stem no tiene porque ser la raíz de la palabra, sino que tiene que coincidir las letras de la palabra con otras sinónimas. El tipo de Stemming que aplicaremos será el Snowball en español. A la función le pasamos la lista de palabras que generamos en la función anterior (noticias tokenizadas, en minúsculas y ya aplicada la lista de parada) y gracias a la ayuda de la librería de PyStemmer podemos realizar de forma automática el proceso de stemmizar los tokens.

```
def stemming(lista_palabras: list):

    texto = ""

    # Cogemos el stemmer de snowball en español

    stemmer = Stemmer.Stemmer('spanish')

    for palabra in lista_palabras:

        # Cogemos el stem de cada palabra

        s = stemmer.stemWord(palabra)

        texto = texto + " " + s

    return texto
```

## Fase Entrenamiento

### 1.1 Función “contar\_ficheros(lista\_ficheros: list)”

La función contar ficheros resulta ser muy simple. Únicamente la usaremos para contar el número concreto de ficheros que se encuentran dentro de una lista. En otras palabras, devolverá la longitud de esta.

Básicamente será usada para obtener de forma muy rápida el número de ficheros de las categorías de odio y de no odio que podremos encontrarnos en listas a lo largo de todo el proceso.

```
def contar_ficheros(lista_ficheros: list):  
    return len(lista_ficheros)
```

### 1.2 Función “seleccionar\_algoritmo(algoritmo: str)”

Esta función permitirá seleccionar uno de los tres algoritmos posibles, “Gradient Boosted Tree”, “Support Vector Machine” o “Decision Tree” y ejecutarlos. Para ello, usaremos varios bucles y cuando el input sea el asignado a cada modelo, se ejecutará mediante una instancia. Podremos ejecutarlos mediante todos los módulos existentes de la librería sklearn.

```
def seleccionar_algoritmo(algoritmo: str):  
    model = ""  
  
    if algoritmo == "Gradient Boosted Tree":  
        model = GradientBoostingClassifier()  
  
    elif algoritmo == "Support Vector Machine":  
        model = SVC()  
  
    elif algoritmo == "Arbol Decision":  
        model = DecisionTreeClassifier()  
  
    return model
```

### 1.3 Función “asociar\_clase(odio: list, no\_odio: list)”

Esta función asigna el tipo de noticia a la que pertenece el texto clasificando en las dos categorías, odio o no odio. Se crean dos listas vacías inicialmente de odio y de no odio y posteriormente se van recorriendo las listas pasadas en la declaración de la función y añadiendo a las listas vacías según corresponda odio o no odio.

Finalmente se unifican en una misma lista y se retorna.

```
def asociar_clase(odio: list, no_odio: list):  
  
    clase_odio = []  
  
    clase_no_odio = []  
  
    for texto_odio in odio:  
  
        clase_odio.append('Odio')  
  
    for texto_no in no_odio:  
  
        clase_no_odio.append('No Odio')  
  
    return clase_odio + clase_no_odio
```

### 1.4 Función “precision(tp: int, fp: int)”

Esta función permite calcular la precisión. Se le pasan los valores de “True Positives” y “False Positives” y simplemente mediante el uso de la función correspondiente ( $tp/(tp + fp)$ ), se calcula la precisión.

Esta función será usada a la hora de entrenar el modelo en la función `entrenar_modelo()`.

```
def precision(tp: int, fp: int):  
  
    return tp/(tp + fp)
```

### 1.5 Función “recall(tp: int, fn: int)”

Esta función es exactamente igual que la anterior pero en vez de calcular la precisión se calcula el recall. En este caso se le pasarán los valores correspondientes a “True Positives” y “False Negatives” y mediante el uso de la operación  $tp/(tp + fn)$  se calculará el recall.

Esta función será usada a la hora de entrenar el modelo en la función `entrenar_modelo()`.

```
def recall(tp: int, fn: int):  
  
    return tp/(tp + fn)
```

### 1.6 Función “visualizacion\_previa(odio, no\_odio, algoritmo)”

Esta función nos permitirá visualizar el número de ejemplares de odio, de no odio, los totales y el modelo seleccionado. También nos muestra un gráfico de barras (mediante el uso de la librería Plotly) comparando los ejemplares de odio y de no odio además de su representación correspondiente.

```
def visualizacion_previa(odio, no_odio, algoritmo):  
  
    num_odio = contar_ficheros(odio)  
  
    num_no_odio = contar_ficheros(no_odio)  
  
    ejemplares = {'Odio': num_odio, 'No Odio': num_no_odio}  
  
    st.text_area("Vista Previa", "Ejemplares 'Odio': " + "\t" +  
str(num_odio) + "\nEjemplares 'No Odio': " + "\t" + str(num_no_odio) +  
"\nTotal: " + "\t" + str(num_odio + num_no_odio) + "\nAlgoritmo  
Seleccionado: " + "\t" + algoritmo, height=130)  
  
    df = pd.DataFrame.from_dict(ejemplares, orient='index',  
columns=['Ejemplares'])  
  
    df = df.rename_axis('Clase')  
  
    figura = px.bar(df, y="Ejemplares", color="Ejemplares")  
  
    st.plotly_chart(figura)
```

### 1.7 Función “entrenar\_modelo(algoritmo: str, coleccion\_documentos: list, clases: list)”

Como el nombre de la misma función expone, esta función permite entrenar el modelo. Lo primero es obtener los valores calculados en procesos anteriores como la matriz de los vectores de noticias obtenidos y los valores de las características obtenidos en el proceso de TF-IDF.

Después, mediante la función `seleccionar_algoritmo()` explicada anteriormente guardamos en una variable el modelo a trabajar y nos ponemos manos a la obra. Para ello, mediante la función proporcionada por `sklearn` `train_test_split` dividimos la matriz anterior en subconjuntos de entrenamiento y prueba. En este caso, usaremos el 70% de los datos para entrenamiento y el 30% para el proceso de testeo.

Posteriormente entrenamos el modelo usando los datos de entrenamiento anteriores y después obtenemos la matriz de confusión. Más tarde calculamos valores útiles para analizar los resultados obtenidos como la precisión, el accuracy o el recall mediante la llamada a las funciones anteriores especificadas que se encargaban de realizar estos cálculos.

Por último, mostramos la matriz de confusión y los resultados mediante la visualización como un heatmap y guardamos los resultados del `tfidf` en `vocabulario.bin` que usaremos más tarde.

```
def entrenar_modelo(algoritmo: str, coleccion_documentos: list, clases: list):

    tf = TfidfVectorizer()

    matriz_idf = tf.fit_transform(coleccion_documentos).toarray()

    df = pd.DataFrame(matriz_idf, columns=tf.get_feature_names_out())

    clf = seleccionar_algoritmo(algoritmo)

    X_train, X_test, Y_train, Y_test = train_test_split(df, clases,
test_size=0.3)

    X_train = pd.get_dummies(X_train)

    X_test = pd.get_dummies(X_test)

    modelo = clf.fit(X_train, Y_train)

    y_pred = modelo.predict(X_test)

    tn, fp, fn, tp = confusion_matrix(Y_test, y_pred).ravel()

    cm = [[tn, fp], [fn, tp]]

    col1, col2, col3 = st.columns(3)
```

```
with col1:

    st.metric("Precision: ", "{:.2f}".format((precision(tp, fp) *
100)) + "%", delta="{:.2f}".format((precision(tp, fp) * 100) - 95) + "%")

with col2:

    st.metric("Accuracy: ", "{:.2f}".format(modelo.score(X_test,
Y_test) * 100) + "%", delta="{:.2f}".format((modelo.score(X_test, Y_test)
* 100) - 95) + "%")

with col3:

    st.metric("Recall: ", "{:.2f}".format((recall(tp, fn) * 100)) +
"%", delta="{:.2f}".format((recall(tp, fn) * 100) - 95) + "%")

st.write("Resultados Entrenamiento: ")

figura = plt.create_annotated_heatmap(cm, x=['Verdadero No Odio',
'Verdadero Odio'], y=['Predicho No Odio', 'Predicho Odio'],
colorscale='Viridis')

st.plotly_chart(figura)

joblib.dump(tf.vocabulary_, 'vocabulario.bin')

return modelo
```

## Fase Testeo/Ejecución

### 1.1 Función “predecir\_clases(modelo, coleccion\_noticias: list)”

Esta función implementa varias funcionalidades. Por un lado, mediante el uso del módulo `tfidfVectorizer` de `sklearn` conseguimos convertir una colección de documentos sin procesar a una matriz de funciones TF-IDF. Adicionalmente le pasamos nuestro fichero de vocabulario creado y exportado en la fase de entrenamiento, concretamente en la función `entrenar_modelo`.

Después una vez tenemos la matriz de funciones TF-IDF, calculamos y transformamos esa colección de noticias que luego podremos visualizar como matriz pasándole justo los vectores de noticias obtenidos y los valores de las características obtenidos en el proceso de TF-IDF.

Por último, usamos esta matriz para el proceso de predicción al modelo que le pasamos al comienzo de la función.

```
def predecir_clases(modelo, coleccion_noticias: list):  
  
    tf = TfidfVectorizer(vocabulary=joblib.load('vocabulario.bin'))  
  
    vectores_noticias = tf.fit_transform(coleccion_noticias).toarray()  
  
    matriz_idf = pd.DataFrame(vectores_noticias,  
columns=tf.get_feature_names_out())  
  
    predicciones = modelo.predict(matriz_idf)  
  
    return predicciones
```

## 1.2 Función “grafica\_resultados(lista\_ficheros: pd.DataFrame, cont\_odio: int, cont\_no: int)”

Esta función nos permitirá graficar los resultados, para ello le pasaremos una lista de ficheros en forma de dataframe de pandas, y dos contadores que indicarán el número de noticias de odio y de no odio.

La función permitirá obtener los valores finales de odio y de no odio que el modelo ha predicho. Para ello, recorre la lista de ficheros y por cada noticia que sea de odio suma 1 al contador de odio y obviamente, para cada noticia que no se relacione con odio suma 1 al contador de no odio.

En un primer momento, la lista de ficheros será un df conformado por otras dos listas, la de noticias y la que especifica si son odio/ no odio, una vez realizada la predicción. Esta función nos permitirá obtener los contadores finales (a los que luego accederemos con .values()) para poder graficar los resultados mediante el uso de la librería de Python Plotly.

```
def grafica_resultados(lista_ficheros: pd.DataFrame, cont_odio: int,  
cont_no: int):  
  
    for clase in lista_ficheros['Odio']:  
  
        if clase == 'Si':  
  
            cont_odio += 1
```



```
else:

    cont_no += 1

return {'Odio': contodio, 'No Odio': contno}
```

## Diseño

### Fuentes de datos y dataset generado

Uno de los primeros pasos del clasificador será seleccionar la ruta de noticias de una categoría concreta para después aplicar el clasificador y ejecutarlo. Por lo tanto, contar con fuentes de noticias en el formato correspondiente ocupa un papel fundamental.

Partimos del correcto web scraping de noticias del periódico digital “El País”. Se han descargado las correspondientes noticias de las secciones “Odio” y “No Odio”. De odio, el clasificador contará con 580 noticias mientras que de la sección opuesta contará con 500 ejemplares.

Los ficheros siguen el formato especificado: “Noticia” + “odio/no Odio” + “medio” + identificador, usando como separador un “\_”. Todas estas noticias serán encapsuladas en dos carpetas ligadas a cada categoría, que se incluirán dentro del propio proyecto.

Para realizar la clasificación y entrenar al modelo escogeremos el path de las noticias deseadas por el usuario. Lo mismo ocurrirá con la fase de testeo, añadiendo a “unlabeled” todas las noticias con las que se desea testear el modelo guardado anteriormente. Una vez generado el resultado final de las clasificaciones, el programa mostrará diferentes datos estadísticos y además, dará la opción al usuario de guardar el modelo en uno de los formatos posibles.

El modelo quedará guardado según 3 tipos de ficheros (csv, txt o xls) y será posible gracias al paquete tkinter filedialog y concretamente al módulo ask saveasfile, donde se podrá seleccionar la ubicación a guardar y especificar el formato concreto. Con esto sería posible ya guardar el modelo en uno de los 3 tipos de ficheros posibles, guardándolo además en la ubicación deseada por el usuario.

## Diagrama UML de clases

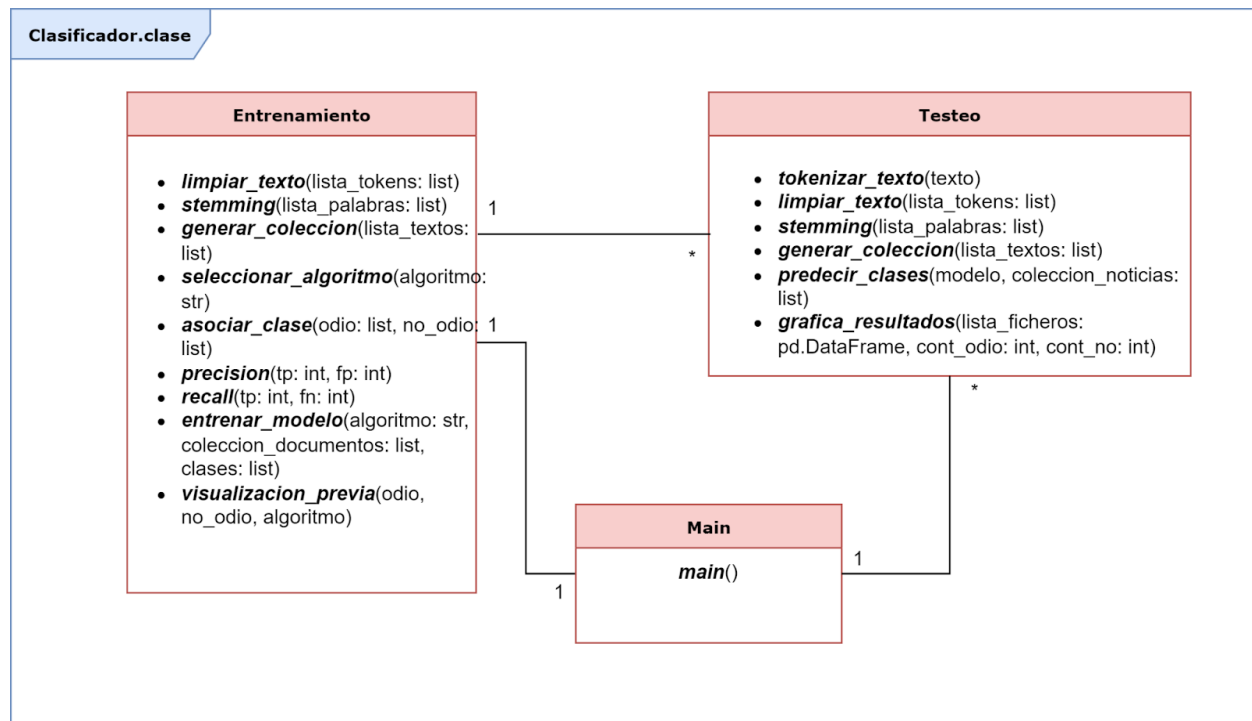


Imagen 8. Diagrama de clases, para representar el clasificador de noticias

## Metodología de trabajo

### Planificación

En cuanto a la planificación seguida a lo largo de este proyecto ha sido la siguiente:

		FECHA INICIO	FECHA FIN	DÍAS
<b>Pentaho</b>	<b>P</b>	<b>10/11</b>	<b>11/14</b>	<b>35</b>
Investigación sobre delitos de odios	P1	10/11	10/17	7
Estudio y selección de fuentes de información	P2	10/18	10/24	7
Aprendizaje y uso de la herramienta Pentaho	P3	10/25	10/31	7
Extracción datos	P4	11/1	11/6	7
Corrección de errores y pruebas	P5	11/7	11/13	7
<b>RapidMiner</b>	<b>R</b>	<b>11/15</b>	<b>12/5</b>	<b>21</b>
Estudio de algoritmos ML a utilizar	R1	11/15	11/17	3
Aprendizaje y uso de la herramienta RapidMiner	R2	11/18	12/2	15
Corrección de errores y pruebas	R3	12/3	12/5	3
<b>Desarrollo App Python</b>	<b>APP</b>	<b>12/6</b>	<b>1/9</b>	<b>35</b>
Desarrollo Fase de entrenamiento	APP1	12/6	12/15	10
Desarrollo Fase de Testeo	APP2	12/16	12/22	7
Creación de la interfaz en Streamlit	APP3	12/23	12/25	3
Presentación y visualizaciones de	APP4	12/26	12/27	2

algunos datos				
Integración de la fase de entrenamiento y testeo con la interfaz	APP5	12/28	1/2	6
Corrección de errores y pruebas	APP6	1/3	1/9	7
Documentación	D	12/13	1/21	40

Tabla 2. Fases y tareas del proyecto.

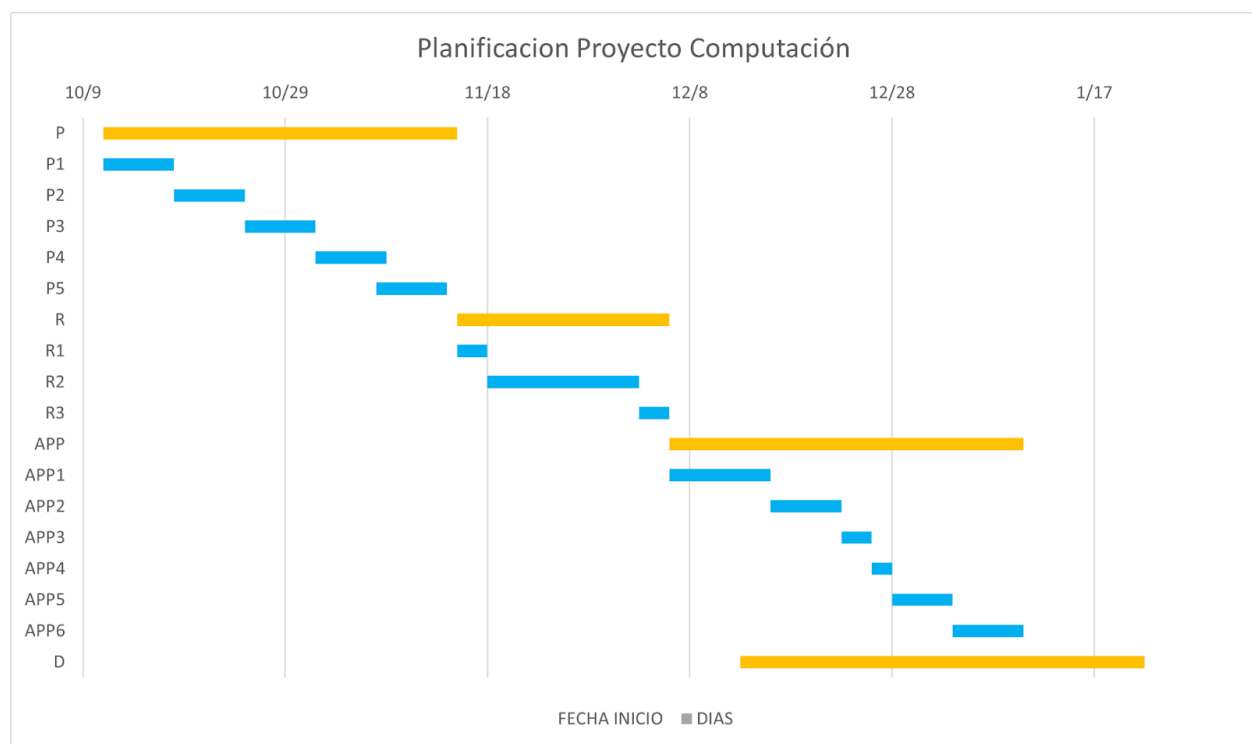


Imagen 9. Diagrama de Gantt, representación de fases y tareas.

## Roles

El equipo ha estado formado por 4 integrantes cuyas funciones han sido las mismas; todos han trabajado de forma cooperativa, definiendo tareas y plazos, cumpliéndolos y haciendo reuniones de puesta en común pues todos debían de tener conocimiento en todo el proyecto desarrollado.

## Presupuesto

Categoría de costes	Descripción			Coste	Mantenimiento (mensual)
SW					
Visual Studio (IDE)	IDE usado para realizar el proyecto junto con sus add-ons, el cual es de licencia libre.			0,00 €	0,00 €
Streamlit	Librería que nos facilita el desarrollo de una interfaz web			0,00 €	0,00 €
Librería	No hemos incluido ninguna librería que pudiera ser de pago			0,00 €	0,00 €
Control de versiones	Repositorio de GitHub, no tiene coste asociado			0,00 €	0,00 €
Pentaho	Herramienta utilizada para la extracción de noticias. Es una herramienta OpenSource			0,00 €	0,00 €
Rapid Miner	Herramienta utilizada previamente para aplicar PLN(Procesamiento de lenguaje natural), algoritmos de Machine Learning y para clasificar y mostrar gráficos sobre la clasificación de las noticias			0,00 €	10,00€
RRHH o servicios (personal de desarrollo)					
Formación	Aprendizaje de ciertas herramientas utilizadas con supervisión profesional	4 Personas	250,00 €/Persona	1.000,00 €	0,00 €
Desarrolladores	4 personas	5 meses	2.200 €/mes	44.000,00 €	0,00 €
				Total	Por Mes
Unificar				45.000,00 €	10,00 €

Tabla 3. Presupuesto del proyecto.

## Diseño de pruebas y resultado de las mismas

Tabla de pruebas y resultados

ID	Tipo	Descripción	Resultado
P-01	Entrenamiento	En la interfaz de entrenamiento seleccionamos y cargamos las noticias de odio y no odio mediante el elemento "Browse Files". La salida esperada sería que se contabilizarán la cantidad de noticias de odio, no odio y el total final de noticias. Además se debería de mostrar una gráfica representando ambas categorías. La salida real se corresponde con lo descrito en la salida esperada, se muestra una gráfica que representa el total de noticias de cada categoría.	✓
P-02	Entrenamiento	Una vez cargadas las noticias y seleccionado el algoritmo de ML procedemos a guardar el modelo en el botón "Guardar Modelo". La salida esperada mostrar al usuario un lugar donde guardar el modelo y mostrar los resultados del entrenamiento del mismo. La salida real se corresponde con mostrar al usuario un lugar donde guardar el modelo en formato ".bin" y mostrar por un lado los porcentajes de "Precision", "Recall" y "Accuracy". También se muestra la tabla de resultados de las predicciones.	✓
P-03	Testeo	En esta prueba se comprueba que el usuario puede introducir el modelo ya entrenado y el	✓

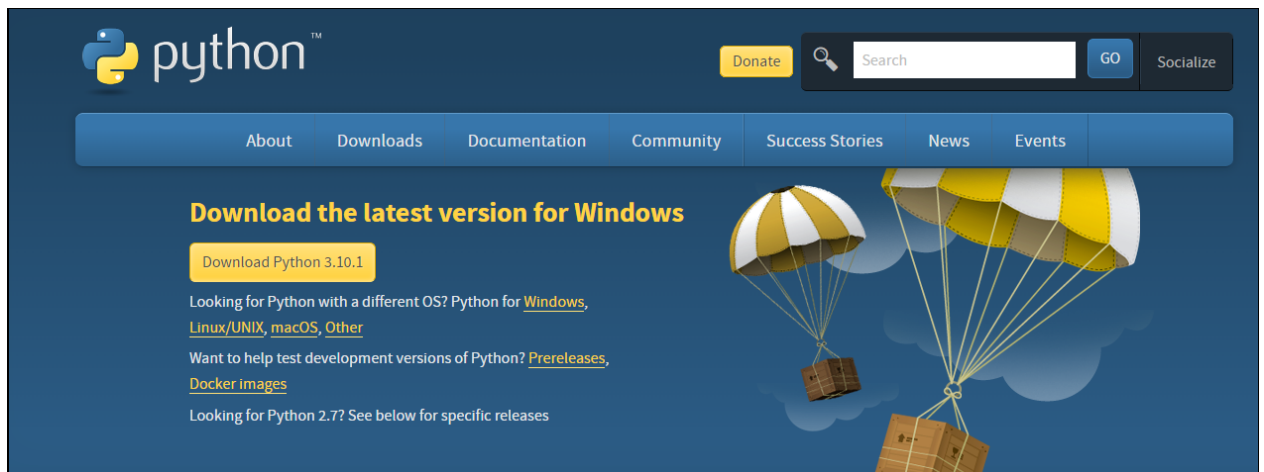
		<p>conjunto de noticias “Unlabeled” sin categorizar.</p> <p>La salida esperada se corresponde con una correcta carga de ambos ficheros y una visualización del contenido de las noticias sin categorizar. Como salida real se obtiene que los archivos se cargan correctamente en la fase de testeo y se muestra una tabla que enumera las noticias junto con su nombre y predicción. Además se puede seleccionar cada noticia y visualizarla de forma independiente. Por otra parte se muestra una gráfica circular con los resultados de dicha clasificación.</p>	
P-04	Testeo	<p>Esta prueba se realiza para comprobar que una vez realizada la fase de testeo, la aplicación sea capaz de guardar los resultados en varios formatos. La salida esperada sería que el usuario pudiera elegir en qué formato guardar los resultados y poder luego visualizarlos.</p> <p>La salida real se corresponde con la generación de ficheros de tipo csv, excel o txt que se pueden elegir dichas opciones dentro de la aplicación. Se generan los archivos y se visualizan correctamente ambos tipos.</p>	✓

*Tabla 4. Tabla de pruebas y resultados.*

## Manual de instalación

1. Instalar última versión python en tu sistema operativo:

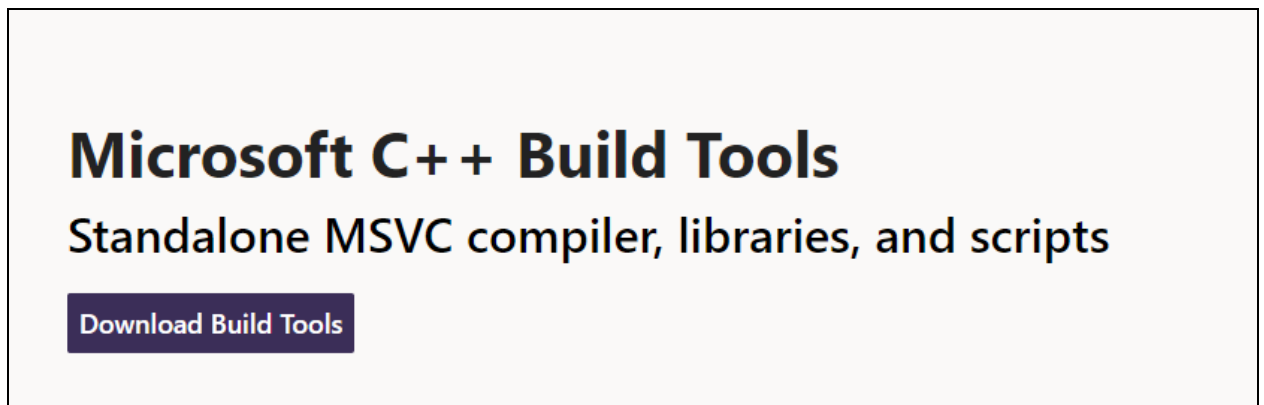
<https://www.python.org/downloads/>



*Imagen 10. Página de descarga de Python.*

2. Instalar el paquete de desarrollo Microsoft Visual 14 C++ o superior:

<https://visualstudio.microsoft.com/visual-cpp-build-tools/>



*Imagen 11. Representa la edad y el porcentaje de delitos de odio de los usuarios.*



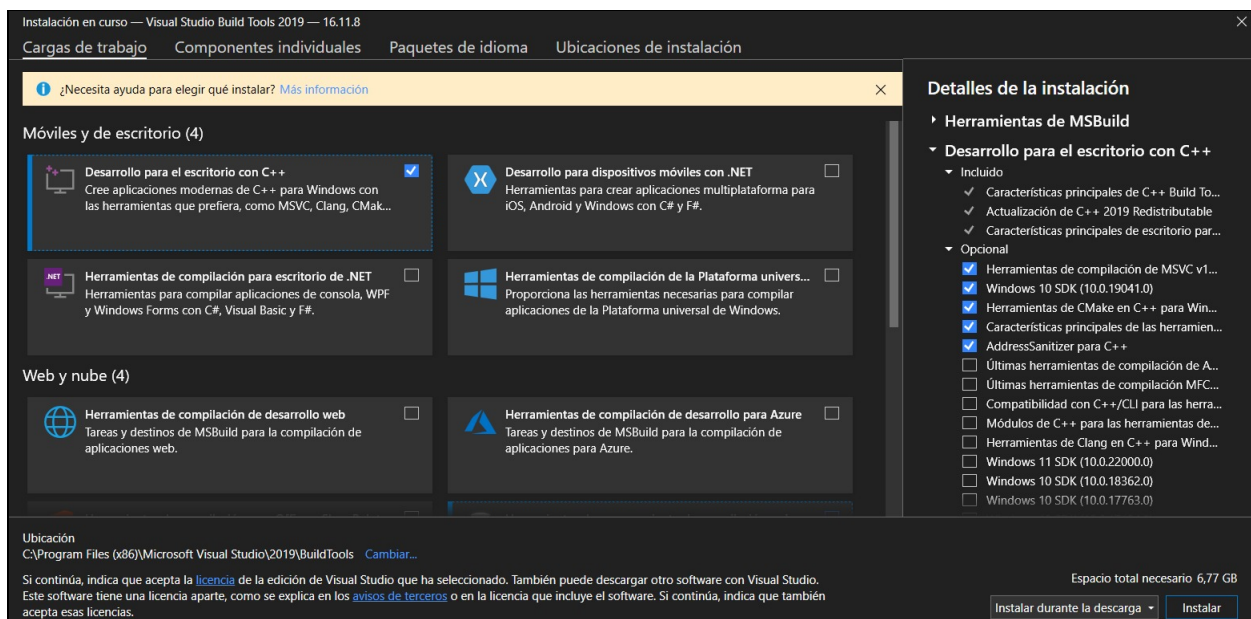


Imagen 12. Herramientas de desarrollo C++.

Seleccionar opción **Desarrollo para el escritorio con C++ y Instalar**

3. Descargar el proyecto desde el repositorio de Github mediante el siguiente enlace:

<https://github.com/joseIgnaciody/Proyecto-Computacion-I> o Zip asociado a la entrega:

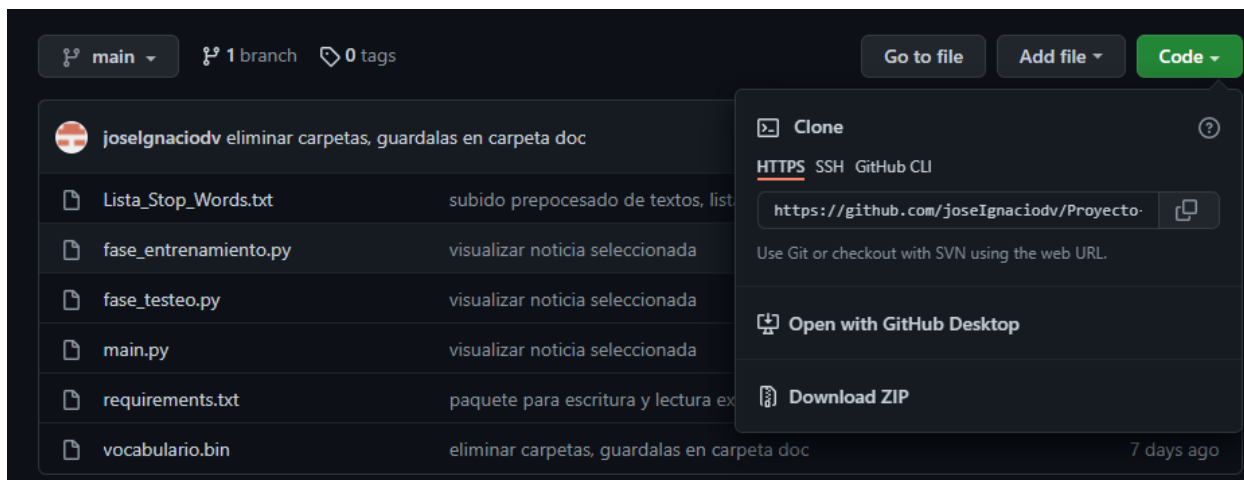
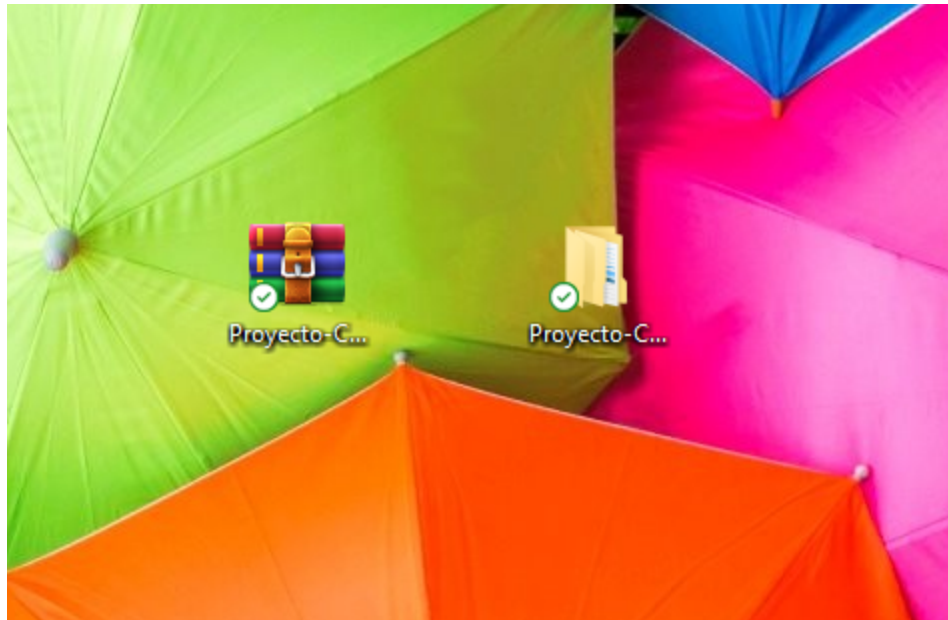


Imagen 13. Repositorio del proyecto en Github.

Descargar zip asociado al repositorio.

4. Extraer el zip descargado previamente en nuestro Desktop o lugar de preferencia:



*Imagen 14. Descarga y compresión del zip del proyecto.*

5. Abrir una nueva terminal, acceder a la carpeta descomprimida, y descargar las librerías y las dependencias desde el fichero de requisitos de aplicación(**requirements.txt**), mediante el siguiente comando:

```
pip install -r ruta/al/fichero/requirements.txt
```

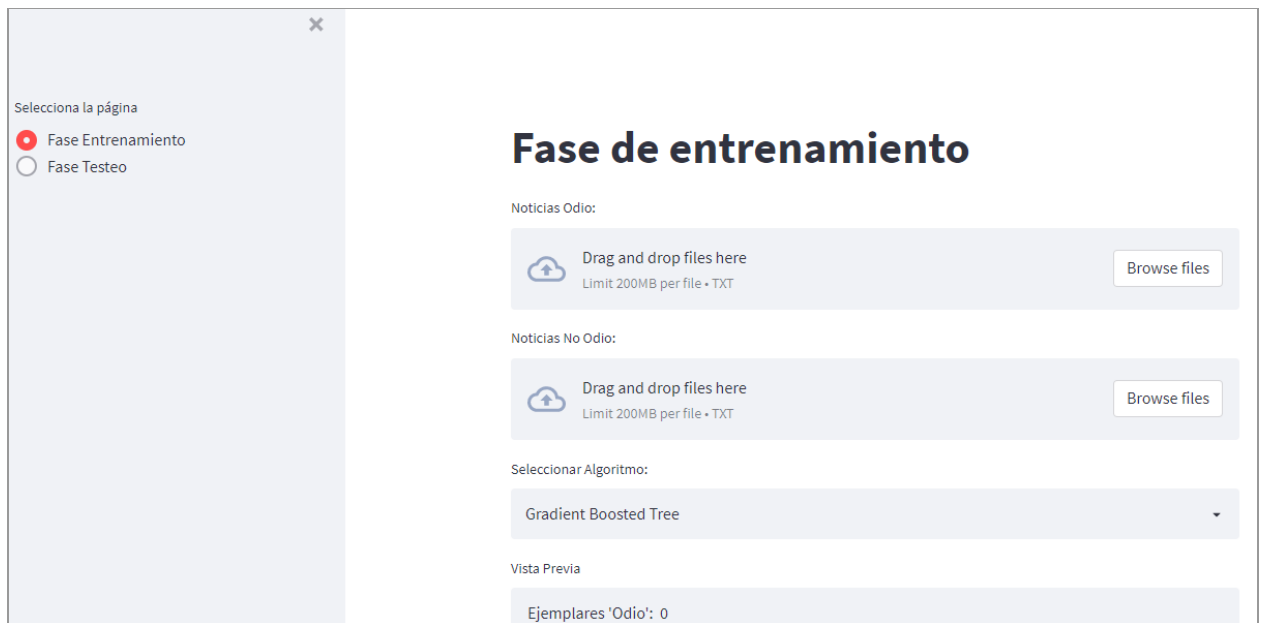
6. Tras finalizar la descarga de las librerías y dependencias, hay que ubicarse en la carpeta del proyecto, mediante el siguiente comando:

```
cd ruta/hasta/carpeta/Proyecto-Computacion-I-main
```

Una vez ubicado en la carpeta de proyecto, para ejecutar la aplicación, se introduce el comando que se muestra a continuación:

```
streamlit run main.py
```

Cuando salga el mensaje **"You can now view your streamlit app in your browser"**, se podrá visualizar la aplicación en el navegador:



*Imagen 15. Aplicación mostrada en el navegador web.*

Si ha ocurrido algún problema con la instalación previamente mostrada, se recomienda realizar una instalación con un entorno virtual, siguiendo los pasos que se muestran en el siguiente tutorial de instalación,

<https://github.com/streamlit/streamlit/wiki/Installing-in-a-virtual-environment>

## Requirements.txt

El fichero que se muestra a continuación, representa el fichero de requisitos, **requirements.txt**, donde se reflejan las librerías junto con las versiones concretas, usadas para la aplicación, este fichero se encuentra dentro de la carpeta **Código**. Para poder ejecutar correctamente la aplicación, ver el paso 5 del **Manual de Instalación**, donde se explica claramente cómo descargar las librerías y dependencias correctas, usando este fichero **requirements.txt**.

```
1  # Automatically generated by https://github.com/damnever/pigar.
2
3  # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_entrenamiento.py: 12
4  # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_testeo.py: 4
5  PyStemmer == 2.0.1
6
7  # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_entrenamiento.py: 16
8  # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_testeo.py: 6
9  joblib == 1.1.0
10
11 # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_entrenamiento.py: 4
12 # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_testeo.py: 3
13 nltk == 3.6.5
14
15 # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_entrenamiento.py: 13
16 # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_testeo.py: 7
17 pandas == 1.3.3
18
19 # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_entrenamiento.py: 14,15
20 # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_testeo.py: 8
21 plotly == 5.4.0
22
23 # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_entrenamiento.py: 5,6,7,8,9,10
24 # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_testeo.py: 5
25 sklearn == 0.0
26
27 # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_entrenamiento.py: 1
28 # C:\Users\delva\OneDrive\Desktop\Proyecto-Computacion-I\fase_testeo.py: 1
29 streamlit == 1.3.0
30
31 openpyxl == 3.0.9
```

## Manual de usuario

### Fase de Entrenamiento

1. Seleccionar Noticias de Odio

2. Seleccionar Noticias de No Odio

3. Seleccionar Algoritmo de ML

4. Vista previa de los datos

5. Gráfica de los datos


6. Guardar modelo

Noticias Odio:  
Drag and drop files here  
Limit 200MB per file • TXT  
Browse files

Noticias No Odio:  
Drag and drop files here  
Limit 200MB per file • TXT  
Browse files

Seleccionar Algoritmo:  
Gradient Boosted Tree

Vista Previa  
Ejemplares 'Odio': 0  
Ejemplares 'No Odio': 0  
Total: 0  
Algoritmo Seleccionado: Gradient Boosted Tree



Guardar Modelo

## 7. Resultados Entrenamiento

Fecha realizacion entrenamiento: 19-01-2022 13:08

Precision:

99.44%

↑ 4.44%

Accuracy:

96.84%

↑ 1.84%

Recall:

95.16%

↑ 0.16%

Resultados Entrenamiento:


## 8. Tabla Predicciones

	Verdadero No Odio	Verdadero Odio
Predicho Odio	9	177
Predicho No Odio	129	1

## 1. Seleccionar Noticias de Odio

Seleccionar las noticias de odio que se van a clasificar en el modelo. Se puede arrastrar directamente un directorio o bien abrirlo y seleccionar cada fichero de tipo txt por independiente.

Noticias Odio:




Drag and drop files here  
Limit 200MB per file • TXT

Browse files

## 2. Seleccionar Noticias de No Odio

Seleccionar las noticias de no odio que se van a clasificar en el modelo. Se puede arrastrar directamente un directorio o bien abrirlo y seleccionar cada fichero de tipo txt por independiente.

Noticias No Odio:

 Drag and drop files here  
Limit 200MB per file • TXT

Browse files

### 3. Seleccionar Algoritmo de ML

---

Para aplicar un algoritmo en concreto de Machine Learning, hacer clic en el desplegable y elegir una de las opciones disponibles como se muestra a continuación:

Seleccionar Algoritmo:

Gradient Boosted Tree ▾

Gradient Boosted Tree

Support Vector Machine

Arbol Decision

### 4. Vista previa de los datos

---

En función de todo lo seleccionado en los puntos 1, 2 y 3, se mostrará un breve resumen de la cantidad de noticias elegidas y el algoritmo que se va a emplear. Ejemplo:

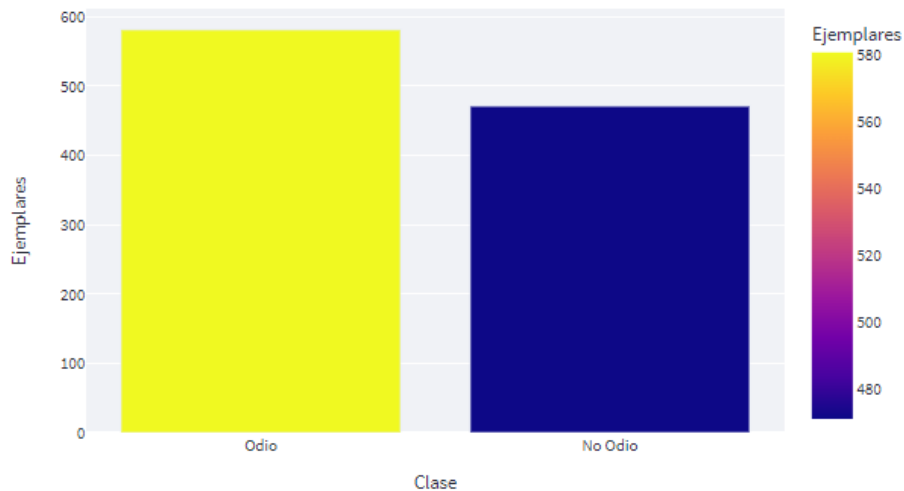
Vista Previa

Ejemplares 'Odio': 581  
Ejemplares 'No Odio': 471  
Total: 1052  
Algoritmo Seleccionado: Support Vector Machine

## 5. Gráfica de los datos

---

Este elemento hace referencia a una gráfica de barras que contrasta el número de noticias de cada categoría. Se crea automáticamente después de introducir los datos de entrada. Ejemplo:



## 6. Guardar Modelo

---

Para aplicar el modelo escogido en función de nuestros datos de entrada, hacemos clic en el botón de “Guardar”. De forma automática se entrenará nuestro modelo y arrojará los resultados en los siguientes puntos.

Guardar Modelo



## 7. Resultados Entrenamiento

---

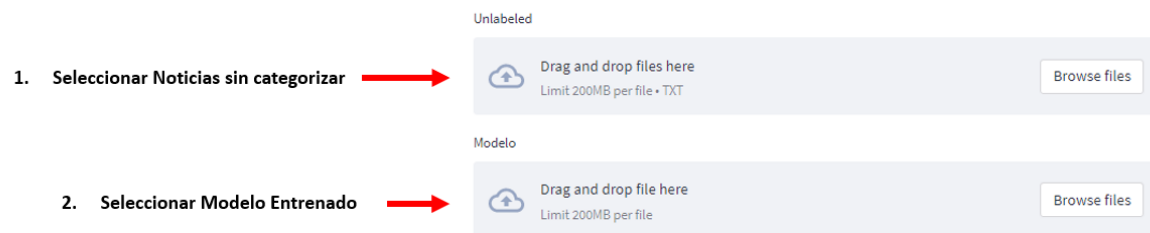
Se pedirá al usuario que elija un directorio donde guardar el modelo entrenado y luego se mostrarán los resultados de esta fase. Se muestran los porcentajes de “Accuracy, Recall y Precision”. También se muestra la fecha de realización del entrenamiento tal y como se ve en las captura del manual.

## 8. Tabla Predicciones

---

La aplicación genera automáticamente una tabla que refleja los resultados de las predicciones de odio y no odio y la cantidad de noticias que efectivamente han sido de odio y no odio respectivamente. Un ejemplo de esta tabla se muestra al inicio del manual.

### Fase de Testeo



3. Tabla de resultados

	Noticia	Odio
0	noticia_no_odio_pais_1321.txt	No
1	noticia_no_odio_pais_1322.txt	No
2	noticia_no_odio_pais_1323.txt	No
3	noticia_no_odio_pais_1324.txt	No
4	noticia_no_odio_pais_1325.txt	No
5	noticia_odio_pais_4355.txt	Si
6	noticia_odio_pais_4356.txt	Si
7	noticia_odio_pais_4357.txt	Si
8	noticia_odio_pais_4358.txt	Si
9	noticia_odio_pais_4359.txt	Si

Ver Noticias

noticia\_no\_odio\_pais\_1... ▾

4. Selección noticias

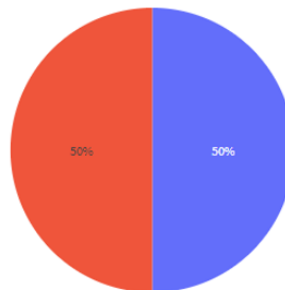
5. Vista previa del contenido

Noticia Seleccionada

130 millones de habitantes. Esta semana, los expertos deberán examinar la efectividad de la alerta para determinar si podrá haber una rápida vuelta a la normalidad, como se hizo durante la primera ola, o se prorroga esta precaución más allá del 7 de febrero previsto en un primer momento. La primera vez estuvo vigente seis semanas. Entre otras cosas, la medida recomienda a los ciudadanos que no salgan después de las ocho de la tarde. Se trata solo de una recomendación, dado que las leyes japonesas no conceden al gobierno competencias para prohibirlo. Sí que ha quedado prohibido que los bares y restaurantes permanezcan abiertos más allá de esa hora. Hasta el momento, la medida parece haber arrojado menos provecho del esperado, en parte debido a la fatiga ante la pandemia. Calles que durante el primer estado de alarma estaban vacías en Tokio ahora continúan viendo aglomeraciones. El tráfico de peatones ha disminuido, sobre todo a partir

6. Gráfica Resultados

Resultados Clasificación



■ Odio  
■ No Odio

8. Guardar Resultados

Guardar Resultados

Guardar Resultados:


- ☒ CSV
- ☐ Excel
- ☐ Txt

7. Tipo de fichero

## 1. Seleccionar Noticias sin Categorizar

Seleccionar las noticias sin categorizar. Se puede arrastrar directamente un directorio o bien abrirlo y seleccionar cada fichero de tipo txt por independiente.

Unlabeled




Drag and drop files here  
Limit 200MB per file • TXT

Browse files

## 2. Seleccionar Modelo Entrenado

Seleccionar el modelo de Machine Learning ya entrenado. Este modelo se obtiene en la fase de entrenamiento. Se puede arrastrar el modelo directamente desde un directorio o bien seleccionarlo a través del gestor de archivos de nuestro sistema operativo. Es obligatorio que el modelo se encuentre en formato “.bin”.

Modelo



Drag and drop file here  
Limit 200MB per file

Browse files

## 3. Tabla de Resultados

En función del modelo entrenado ya cargado, se calcula la predicción para las noticias sin categorizar cargadas en el punto 1. Se creará una tabla de resultados que contiene el resultado de la predicción de cada una de las noticias.

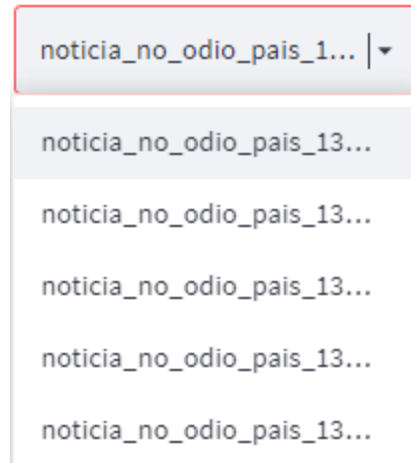
	Noticia	Odio
0	noticia_no_odio_pais_1321.txt	No
1	noticia_no_odio_pais_1322.txt	No
2	noticia_no_odio_pais_1323.txt	No
3	noticia_no_odio_pais_1324.txt	No
4	noticia_no_odio_pais_1325.txt	No
5	noticia_odio_pais_4355.txt	Si

## 4. Selección noticias

---

Desplegable “Ver Noticias” que nos permite visualizar el contenido de cada noticia cargada y ya clasificada.

Ver Noticias



noticia\_no\_odio\_pais\_1... | ▾

noticia\_no\_odio\_pais\_13...

noticia\_no\_odio\_pais\_13...

noticia\_no\_odio\_pais\_13...

noticia\_no\_odio\_pais\_13...

noticia\_no\_odio\_pais\_13...

## 5. Vista previa del contenido

---

Una vez seleccionada la noticia, este será el elemento que mostrará el contenido de la misma tal y como se muestra a continuación:

Noticia Seleccionada

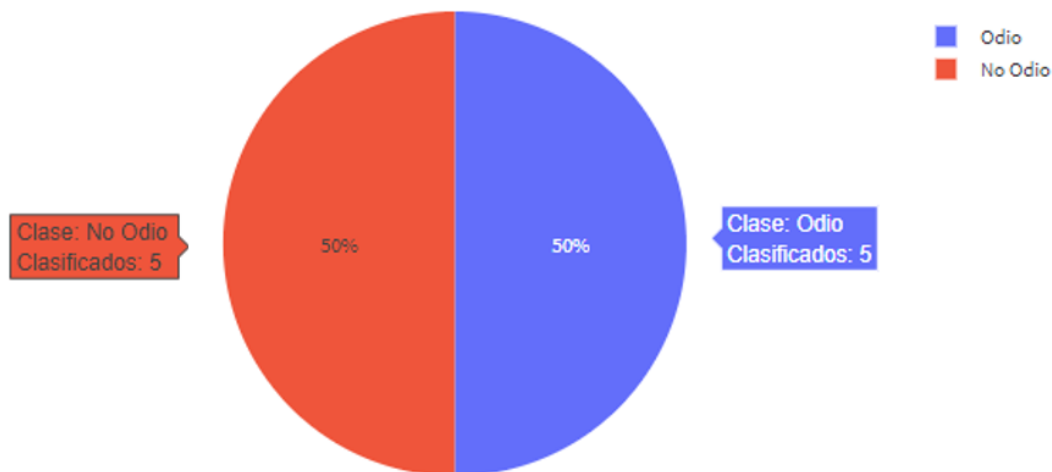
‘Cómo aprender a tratarse bien a uno mismo’, la guía no escrita que hace falta elaborar ya  
#####  
No siempre es fácil, no nos han enseñado a hacerlo, pero hay ciertos conceptos básicos de autocuidado, autoconsideración y autocompasión que podemos desarrollar desde hoy  
#####  
Si te resulta más fácil ser tu peor crítico que ser considerado contigo mismo, no estás solo. Tal vez pienses cosas hirientes sobre ti o te insultes verbalmente de ti mismo. Probablemente no te des el tiempo suficiente para descansar, comas mal y abusos de sustancias que dañan tu cuerpo o, quizás, simplemente estés más preocupado por complacer a los demás que hacer cosas buenas por ti. Experimentar esto es común, y hasta cierto punto normal, la buena noticia es que podemos

En este elemento se permite realizar scroll vertical para visualizar el contenido restante o bien ampliar el campo de visualización situando nuestro cursos en el lado inferior derecho.

## 6. Gráfica Resultados

Los resultados de la clasificación de nuestras noticias sin categorizar se muestran en una gráfica que permite tener una mayor claridad de cómo ha funcionado nuestro modelo ya entrenado. Se trata de una gráfica interactiva que permite ampliar su tamaño o descargar una copia de la misma en formato “.png”.

*Resultados Clasificacion*



## 7. Tipo de Fichero

Seleccionar el tipo de fichero en el queremos guardar nuestros resultados. Se ofrecen tres tipos:

- CSV
- Excel
- Txt

Guardar Resultados:

- ☐ CSV
- ☒ Excel
- ☐ Txt

## 8. Guardar Resultados

---

Para guardar los resultados obtenidos en la fase de testeo, hacemos clic en el botón de “Guardar Resultados”. Se nos pedirá seleccionar la ubicación de donde queremos guardar nuestro archivo.

Guardar Resultados

## Conclusiones

Este proyecto ha sido una combinación total de tres versiones: en la primera utilizamos la herramienta Pentaho para hacer el web scraping de todas las noticias que necesitáramos, en el segundo utilizamos la aplicación RapidMiner que nos permitía realizar nuestro clasificador de noticias a partir de las generadas en Pentaho gracias a utilidades que incluía como el procesamiento de lenguaje natural o la aplicación de modelos de machine learning, y por último desarrollamos este clasificador en el lenguaje Python, aplicando y desarrollando de forma similar las utilidades de procesamiento de lenguaje natural o machine learning entre otras que disponía RapidMiner, concluyendo el trabajo con una aplicación con una interfaz web interactiva donde puedes seleccionar las noticias para entrenarlas y posteriormente meter otras de prueba para que la aplicación las clasifique.

En líneas generales podemos decir que hemos concluido este proyecto con éxito, gestionando de forma escalada versión de este proyecto hasta llegar a la final y utilizando diversas herramientas que nos facilitaron trabajo pero que posteriormente desarrollamos nosotros para ofrecer a un posible usuario una aplicación web útil, robusta y funcional.

## Trabajos futuros

La trazabilidad de este proyecto en un futuro se puede llevar por diversos caminos y añadir más funcionalidades gracias a la escalabilidad llevada desde un principio. Entre algunas mejoras futuras en la aplicación están:

- ☐ Desarrollar en Python un web scraping que tome los datos de las diferentes noticias y que se actualice cada cierto tiempo para tener la información actualizada.
- ☐ Utilizar más algoritmos de machine learning.
- ☐ Hacer recomendaciones basadas en noticias que le gusten al usuario.
- ☐ Permitir que el usuario elija los periódicos sobre los que le gustaría que se generara su clasificación o la recomendación en caso de que se implementara
- ☐ Mostrar estadísticas y gráficas de elementos relevantes que han definido la clasificación.
- ☐ Crear más categorías de recomendación, como deporte, ciencia o tecnología por ejemplo y no solamente quedarnos en noticias de odio.
- ☐ Puesto que hemos desarrollado una interfaz web, subir la aplicación web a internet para que pueda ser probada.

## Bibliografía

- *Árbol de decisión en Machine Learning (Parte 1)*. (2019, December 14). sitiobigdata.com.  
Retrieved December 11, 2021, from  
<https://sitiobigdata.com/2019/12/14/arbol-de-decision-en-machine-learning-parte-1/#>
- Bird, S. (n.d.). *NLTK*. Wikipedia. <https://es.wikipedia.org/wiki/NLTK>
- Boulton, R. (n.d.). *PyStemmer · PyPI*. PyPI. <https://pypi.org/project/PyStemmer/>
- Cournapeau, D. (n.d.). *Scikit-learn*. Wikipedia. <https://es.wikipedia.org/wiki/Scikit-learn>
- *Estadísticas*. (n.d.). Ministerio del Interior.  
<http://www.interior.gob.es/web/servicios-al-ciudadano/delitos-de-odio/estadisticas>
- *Gradient boosting*. (n.d.). Wikipedia. Retrieved December 11, 2021, from  
[https://es.wikipedia.org/wiki/Gradient\\_boosting](https://es.wikipedia.org/wiki/Gradient_boosting)
- McKinney, W. (n.d.). *Pandas (software)*. Wikipedia.  
[https://es.wikipedia.org/wiki/Pandas\\_\(software\)](https://es.wikipedia.org/wiki/Pandas_(software))
- Meyer, S. (2015, April 14). *Cuatro de cada diez delitos de odio, vinculados a la orientación sexual*. El Mundo.  
<https://www.elmundo.es/espana/2015/04/14/552d1fbe268e3e8f1d8b4575.html>
- Ministerio de Interior. (2021, Junio). *Informe Delitos de Odio*. Informe con porcentajes y datos recogidos a partir de una encuesta.  
[http://www.interior.gob.es/documents/642012/13622471/Informe+de+la+encuesta+sobre+delitos+de+odio\\_2021.pdf/0e6ffacb-195e-4b7b-924e-bf0b9c4589b5](http://www.interior.gob.es/documents/642012/13622471/Informe+de+la+encuesta+sobre+delitos+de+odio_2021.pdf/0e6ffacb-195e-4b7b-924e-bf0b9c4589b5)
- *Plotly*. (n.d.). Wikipedia. <https://en.wikipedia.org/wiki/Plotly>
- *Python*. (n.d.). Wikipedia. <https://es.wikipedia.org/wiki/Python>



- 
- *Qué es un delito de odio.* (n.d.). Ministerio del Interior.  
<http://www.interior.gob.es/web/servicios-al-ciudadano/delitos-de-odio/que-es-un-delito-de-odio>
  - Richards, T. (n.d.). Streamlit • The fastest way to build and share data apps.  
<https://streamlit.io/>
  - *Support Vector Machine — Introduction to Machine Learning Algorithms | by Rohith Gandhi.* (n.d.). Towards Data Science. Retrieved December 11, 2021, from  
<https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>