Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MINERÍA DE DATOS
4º curso, grupo 46
Primer cuatrimestre

CLUSTERING

Aingeru Bellido, Nagore Gómez, Sergio Martín y Maitane Urruela.

Bilbao, 30 de octubre de 2023

Índice

Índice de figuras

Índice de tablas

1. Introducción

1.1. Definición de la tarea

El *Text Mining* es uno de los campos del *Data Science* en el que los conjuntos de datos para el proceso de extracción de información son textos de toda índole (opiniones, paginas web, comentarios, transcripciones, etc). En este proyecto se propone una tarea de *Text Mining* en el que el objetivo es agrupar documentos no clasificados, es decir, llevar a cabo un proceso de clasificación no supervisada. Trabajando sobre grandes conjuntos de datos se procede a descubrir las posibles agrupaciones naturales en los documentos tratados.

1.2. Objetivo

El objetivo de este proyecto es aplicar un proceso de *clustering* basado en densidad sobre un conjunto de datos, y adquirir las siguientes competencias:

- Capacidad para implementar una técnica de aprendizaje no supervisado.
- Capacidad para implementar un barrido de parámetros.
- Capacidad para conocer el coste computacional y las limitaciones de representación.
- Capacidad para caracterizar documentos en diferentes representaciones de atributos.
- Capacidad para hacer un análisis crítico y una discusión de los resultados.

1.3. Presentación del dataset

Para desarrollar este proyecto se ha empleado el *dataset* de la plataforma *"Kaggle"* llamado *"Suicide and Depression Detection"*[11] que presenta varios *tweets* de Twitter. Este *dataset* dispone de 232074 instancias con 3 atributos cada una: el *ID* de la instancia, el texto que contiene el *tweet* y la clase a la que pertenece la instancia.

- El *ID* es un atributo numérico que sirve para identificar cada una de las instancias.
- El texto es un atributo de tipo *string* con el contenido del *tweet* que representa la instancia (este es el atributo que se usará para el proceso de *clustering*). Todos son valores únicos (no se repite el mismo *tweet* dos o más veces).
- La clase puede tener uno de los siguientes valores nominales: *"suicide"*, si la persona que lo escribió acabó suicidándose o *"non-suicide"* en el caso contrario. Para cada uno de esos valores de clase se dispone del mismo número de instancias.

2. Aspectos teóricos

2.1. Vectorización de los datos

2.1.1. "Doc2Vec"

Los *embedding* son una técnica de procesamiento de lenguaje natural que convierte el lenguaje humano en vectores numéricos. Estos vectores son una representación del significado subyacente de las palabras, lo que permite que los ordenadores procesen el lenguaje de manera más efectiva. También permiten que los datos se manipulen matemáticamente.[6]

Este tipo de vectorización tiene un enfoque basado en redes neuronales que aprende la representación distribuida de documentos. Es una técnica de aprendizaje no supervisado que mapea cada documento a un vector (*embedding*), de tal manera que los documentos similares se mapean a puntos cercanos en el espacio vectorial. Esta técnica permite comparar documentos basados en su representación vectorial y realizar tareas como la de clasificación de documentos[12] [20] (la tarea planteada para este proyecto).

Cabe mencionar que también existe una técnica denominada *"Word2Vec"*. Esta técnica, consiste en asociar a cada uno de los *tokens* de cada texto un vector, capturando su significado, similitud semántica y la relación con el texto alrededor. Esto permite predecir una palabra dado el contexto y viceversa.[10]

Sin embargo, se considera más apropiado *"Doc2Vec"* para esta tarea, puesto que no solo captura los *embedding* de cada *token*, sino que crea un único vector con todo el contexto del texto, por lo que es más útil a la hora de llevar a cabo tareas de *clustering* en la clasificación de documentos.

En caso de emplear *"Word2Vec"*, como en este caso se está procediendo a la clasificación de textos, sería necesario sumar todos los vectores (componente a componente) de las palabras de un documento para obtener el vector correspondiente al mismo, con el coste que esto conlleva.

Previo a aplicar el *"Doc2Vec"* se requiere de un preprocesado en los datos para facilitar la tarea del modelo.

Limpieza del *dataset* y *tokenización* de los datos

Para llevar a cabo la limpieza de un *corpus*, hay varias técnicas que se le pueden aplicar sobre los datos, tales como:

- **Tokenización**: El proceso de tokenizar consiste en pasar cada texto a una cadena de palabras o caracteres sueltos, llamados *"tokens"*, que se manipularán individualmente en los siguientes procesos.
- **Lematización**: Este proceso relaciona una palabra derivada con su forma canónica o lema, donde no posee género, número ni conjugación. De esta forma, todas las diferentes representaciones de una misma palabra se juntan en un solo término, lo que ayuda al usar métodos basados en la frecuencia de las palabras, como es el caso [8].

- **Limpieza de datos**: En esta parte se eliminarán las letras mayúsculas, signos de puntuación, carácteres especiales, etc.
- **Limpieza de *stopwords***: Este tipo de palabras son las que el modelo considera redundantes o superficiales a la hora de agrupar los textos. Quitando estos *tokens*, se ayuda al modelo a obtener una mejor idea del tema que se aborda en cada documento.
- **Palabras demasiado cortas**: Se procede a eliminar aquellas palabras que contengan menos de 3 caracteres, como podrían ser: *'I'*, *'do'*, *'is'*... Pues tienden a ser palabras muy comunes sin demasiada relevancia a la hora de realizar el proceso de *clustering*.

2.1.2. *"Transformers"*

Los *"Transformers"* son una arquitectura de red neuronal que se utilizan en el ámbito del PLN (Procesamiento del Lenguaje Natural) para capturar las relaciones entre las palabras en un texto. Se basa en el uso de atención (*attention*) para permitir que las palabras interactúen entre sí y se comprendan en el contexto global del texto.[24]

Utilizan las capas de atención para procesar las palabras de manera simultánea y paralela. Esto permite que el modelo se beneficie de las relaciones de largo alcance y capture la información contextual de manera más efectiva [22].

Al utilizar *"Transformers"* para la vectorización de los textos, dependiendo del *dataset* se puede llegar a tener una mejor representación en los *embeddings*, que por ejemplo utilizando *"Doc2Vec"*, al tener en cuenta la *attention* antes mencionada. Es decir, se centran en la información más relevante del texto.

Además, no es necesaria una limpieza previa de los datos, puesto que los *"Transformers"* son capaces de lidiar con textos ruidosos, como *tweets* con errores ortográficos, transcripciones de voz imperfectas o textos con ambigüedades. Esto se da debido a que pueden aprender a partir de grandes conjuntos de datos que incluyen una variedad de textos, lo que les permite adaptarse y generalizar a diferentes condiciones.

2.1.3. Métodos alternativos de vectorización

En los anteriores apartados se hace especial hincapié en las técnicas de vectorización *"Doc2Vec"* y *"Transformers"*. No obstante, existen otras alternativas como son *"Bag of Words"* y *"TF-IDF"*.

Debido a que las dos técnicas de vectorización aquí mencionadas son bastante menos sofisticadas que *"Doc2Vec"* o *"Transformers"* se considera que no son del todo apropiadas para esta tarea, dada la complejidad de los datos sobre los que se trabaja.

"TF-IDF"

Term Frequency-Inverse Document Frequency es una técnica empleada en el Procesamiento del Lenguaje Natural para evaluar la relevancia de un término en un documento con respecto a un

conjunto de documentos.

La vectorización con este método se divide en dos procedimientos: *TF*, donde se observa la frecuencia con la que aparece cada término en los documentos e *IDF* donde se evalúa la importancia de cada término en el conjunto de documentos.

El resultado del cálculo de *"TF-IDF"* proporciona un valor numérico que refleja cuán importante es un término en un documento específico en comparación con un conjunto de documentos. Los términos que son más específicos de un documento tendrán puntuación más alta, mientras que los términos comunes tendrán una puntuación más baja.

"Bag of Words"

El modelo *"BoW"* es una técnica de representación de texto que se centra en la aparición de las palabras en un documento sin tener en cuenta su orden ni su contexto. Básicamente, considera un documento como una "bolsa" que contiene todas las palabras que aparecen en él, y luego cuenta cuántas veces aparece cada palabra en esa "bolsa". Cada palabra se trata como una característica y se crea un vector de características que representa el documento.

Sin embargo, esta técnica tiene limitaciones importantes, como la pérdida de información contextual y semántica, ya que no considera el orden de las palabras ni su significado.

2.2. Métricas de comparación vectorial

Si bien es cierto que la métrica por excelencia en situaciones cotidianas suele ser la distancia euclidiana, existen muchas otras métricas para evaluar la similitud entre vectores. Mediante los procesos de vectorización se consigue transformar texto a vectores numéricos, con el fin de poder compararlos entre si. A lo largo de esta investigación se han empleado dos métricas principalmente:

2.2.1. Distancia euclidiana

Esta métrica mide la distancia "ordinaria" entre dos puntos en un espacio euclídeo tomando como base el Teorema de Pitágoras. Aunque es muy común en espacios bidimensionales es ampliable a espacios euclídeos n-dimensionales. Se calcula de la siguiente forma:

$$D_E(P,Q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

2.2.2. Similitud coseno

Se trata de una medida de similitud existente entre dos vectores en un espacio que posee un producto interior con el que se evalúa el valor del coseno del ángulo comprendido entre ellos. Esta función trigonométrica proporciona un valor igual a 1 si el ángulo comprendido es cero, es decir si ambos vectores apuntan a un mismo lugar. Con cualquier ángulo diferente a cero existente entre los vectores, el coseno arrojaría un valor inferior a uno. Si los vectores fuesen ortogonales, el coseno

se anularía, y si apuntasen en sentido contrario, su valor sería -1. De esta forma, el valor de esta métrica se encuentra en el intervalo cerrado [-1,1]. (Ver figura 1)

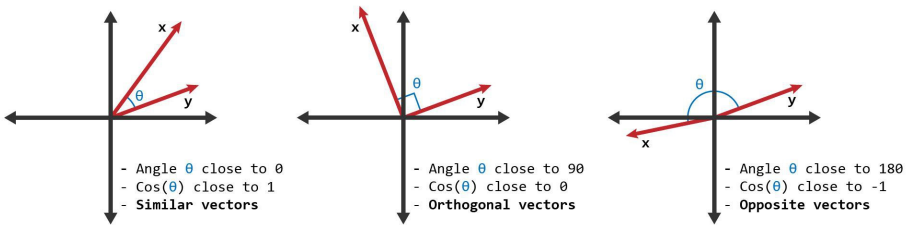

Figura 1: Similitud de coseno entre diferentes vectores
(Fuente: FineProxy)

Por ello, es especialmente adecuado para espacios de datos con dimensionalidades altas. El valor de la similitud coseno se mide de la siguiente manera:

$$Similarity = \cos(\theta) = \frac{\vec{A} * \vec{B}}{||\vec{A}|| * ||\vec{B}||} = \frac{\sum_{i=1}^{N} A_i * B_i}{\sqrt{\sum_{i=1}^{N} {A_i}^2} * \sqrt{\sum_{i=1}^{N} {B_i}^2}}$$

2.3. Definición del algoritmo

2.3.1. *Clustering* basado en densidad

Los algoritmos basados en densidad son algoritmos no supervisados que identifican agrupaciones en los datos en base a regiones de alta densidad de puntos (instancias). Estas agrupaciones forman *clusters* entre sí y se separan de otros *clusters* por zonas de baja densidad entre ellos. [18]

Este tipo de *clustering* se basa en dos parámetros. Por un lado, el concepto de ε vecindarios. La idea general de este valor, es que dado un punto del *dataset*, se pueda razonar sobre el resto de puntos que estén en el espacio de alrededor suyo. Por ejemplo, si definimos un valor ε de 0.5 (radio) para el punto p, crearíamos un vecindario de ese tamaño a su alrededor (ver figura 2).

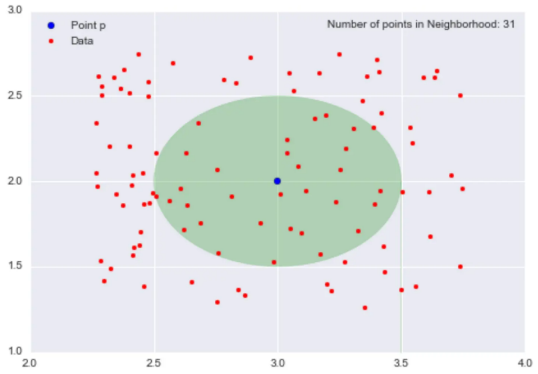

Figura 2: Vecinos de p recogidos por ε
(Fuente: [18])

El siguiente concepto es el de "densidad". Teniendo un punto p y el parámetro ε antes definido, podemos definir la masa como la cantidad de puntos dentro del vecindario. En este caso el valor de la masa sería 31 y al ser en un espacio de dos dimensiones, el volumen sería el área del círculo, es decir: $A = \pi r^2 = \pi \cdot 0{,}25$ (ver figura 2). Con el volumen calculado, solo faltaría calcular la densidad, que definiríamos de la siguiente manera:

$$densidad = \frac{masa}{volumen} = \frac{31}{\frac{\pi}{4}} = \frac{124}{\pi} \approx 39{,}5$$

Este valor no tiene ningún significado por sí solo, pero calculando la aproximación local de la densidad de todos los puntos del *dataset*, se podrían agrupar las instancias en base a su cercanía, haciendo de todos los puntos con parecida densidad local parte del mismo *cluster*. [18]

2.3.2. *"DBSCAN"*

El algoritmo *"DBSCAN"* [7] (*Densitiy-Based Spatial Clustering of Applications with Noise*) es el algoritmo de clustering por densidad más conocido, por lo que se ha escogido para llevar a cabo esta tarea. A diferencia de algoritmos como *"K-Means"*, los algoritmos basados en densidad no necesitan recoger el número de *clusters*, sino que lo infieren basándose en los datos. Como se ha explicado en el apartado anterior, para su aplicación se requiere de los siguientes parámetros [18]:

- ε: El radio de vecinos alrededor de un punto.
- ***minPts***: El mínimo de vecinos para definir un *cluster*. Es decir, durante el algoritmo no se establecerán *clusters* que tengan menos de $minPts$ vecinos.

Mediante estos parámetros, el algoritmo *"DBSCAN"* clasifica los puntos en las siguientes tres categorías:

- ***Core Points***: Serán de este tipo los puntos que formando un radio ε tengan $minPts$ vecinos o más. Los *Core Points* son los puntos sobre los cuales se formarán los *clusters*. Aunque el valor de ε es igual para todos los puntos, lo que varía es la cantidad de nodos vecinos que recogen. Por ello, ajustando el valor de $minPts$ se modificarán los *clusters* y su respectiva densidad.
- ***Border Points***: Serán los puntos que no cumplan la condición de ser *Core Point*, pero estén dentro del radio de un *Core Point*.
- ***Outlier***: Serán los puntos que no cumplan ninguna de las dos condiciones de ser *Core Point* ni *Border Point*. A los puntos *Outlier* se les considera como 'ruido' y no serán asignados a ningún *cluster*.

En el ejemplo de la figura 3 se puede observar como los puntos "A" son *Core Points*. Sin embargo, los puntos "B" y "C" no cumplen la condición para ser *Core Point* ya que solo tienen un vecino, pero sí que son considerados como *Border Point* al ser vecinos de al menos un *Core Point*. Finalmente, el punto "N" es un *Outlier* ya que no cumple las condiciones para ser ninguno de los anteriores.

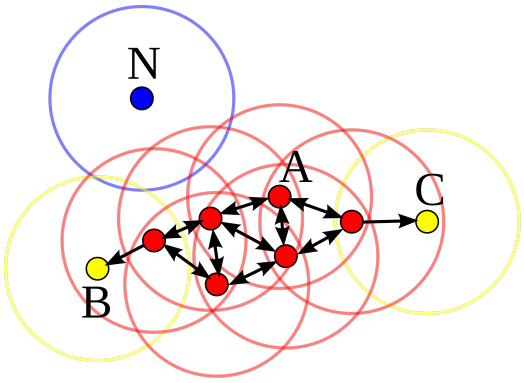

Figura 3: Ejemplo de *clustering* con *"DBSCAN"* donde $minPts = 3$.
(Fuente: [3])

2.3.3. *"DBSCAN"* - Pseudo-código

El algoritmo se ha diseñado en base a lo expuesto en el artículo [7].

Require: $Z_{train}, \varepsilon, MinPt$
  Let: $n = |Z_{train}| = |\{z_1, z_2, ..., z_j...., z_n\}|$
  Let: $\varepsilon, epsilon\ parameter$
  Let: $minPt, minimum\ points\ to\ consider\ a\ cluster\ parameter$
Ensure: $\hat{c}$
  $\hat{c} = (\hat{c}_1, ..., \hat{c}_n)$

  Begin:
  $\hat{c} \leftarrow [0] * n$
  $clusterId \leftarrow 0$
  for $z^j \in Z_{train}$ **do**
    $V_{[j]} \leftarrow \{l \mid z^l \in Z_{train} \wedge d(z^l, z^j) \leq \varepsilon \wedge \hat{c}_{[j]} = 0\}$
    if $|V_{[j]}| < minPt$ **then**
      $\hat{c}_{[j]} \leftarrow -1$
    else
      $clusterId \leftarrow clusterId + 1$
      $\hat{c}_{[j]} \leftarrow clusterId$
      for $k \in V_{[j]}$ **do**
        if $\hat{c}_{[k]} = -1$ **then**
          $\hat{c}_{[k]} \leftarrow clusterId$
        else if $\hat{c}_{[k]} = 0$ **then**
          $\hat{c}_{[k]} \leftarrow clusterId$
          for $z^x \in Z_{train}$ **do**
            $V2_{[x]} \leftarrow \{i | z^i \in Z_{train} \wedge d(z^i, z^x) \leq \varepsilon\}$
            if $|V2_{[x]}| \geq minPts$ **then**
              $V_{[j]} \leftarrow V_{[j]} + V2_{[x]}$
            end if
          end for
        end if
      end for
    end if
  end for

3. Algoritmo

El algoritmo de desarrollo propio, en cambio, es el siguiente (explicado en el apartado 5.1):

Require: $Z, \varepsilon, MinPt$
  Let: $n = |Z| = |\{z_1, z_2, ..., z_j ...., z_n\}|$
  Let: $\varepsilon, epsilon\ parameter$
  Let: $minPt, minimum\ points\ to\ consider\ a\ cluster\ parameter$
Ensure: $\hat{c}$
  $\hat{c} = (\hat{c}_1, ..., \hat{c}_n)$

  Begin:
  //Se recogen los vecinos de cada instancia
  for $z_j \in Z$ **do**
    $V_{[j]} \leftarrow \{z_i \mid z_i \in Z \wedge d(z_i, z_j) \leq \varepsilon \wedge i \neq j\}$
  end for
  //Se encuentran los Core Points del dataset
  $N \leftarrow \{z_j \mid z_j \in Z \wedge |V_{[j]}| \geq minPt\}$
  //Se preestablece el cluster de las instancias a ruido
  $\hat{c} \leftarrow [-1] * n$
  $nCluster \leftarrow -1$
  //Se recorren los Core Points asociándolos al siguiente número de cluster
  for $z_j \in N$ **do**
    if $\hat{c}\,[j] = -1$ **then**
      $nCluster \leftarrow nCluster + 1$
      $\hat{c}\,[j] \leftarrow nCluster$
      $S \leftarrow \{z_j\}$
      for $z_i \in S$ **do**
        $S \leftarrow S \setminus \{z_i\}$
        //Se asocian todos los vecinos del Core Point a su mismo cluster
        for $z_v \in V_{[j]}$ **do**
          //Se comprueba que no estuviesen ya asociados a otro cluster
          if $\hat{c}\,[v] = -1$ **then**
            $\hat{c}\,[v] \leftarrow nCluster$
          end if
        end for
      end for
    end if
  end for

$A \leftarrow \{\}$
//Se recorren todos los clusters
for $\{c_j \mid c_j \in \hat{c} \wedge \forall c_i \in \hat{c} : c_j \neq c_i\}$ **do**
  //Se comprueba que cumplan con los requisitos del minPoints
  if $|\{c_z \mid c_z \in \hat{c} \wedge c_z = c_j\}| < minPt$ **then**
    //Si no lo cumplen se añaden todas las instancias de ese cluster a la lista de alcanzables
    $A \leftarrow A \cup \{[V_{[i]}] \mid c_i \in \hat{c} \wedge c_j = c_i\}$
    //Si lo cumplen se los asocia como cluster válido
  else
    $C \leftarrow C \cup c_j$
  end if
end for
//Se recorren todos los vecinos de los alcanzables
for $[V_{[j]}] \in A$ **do**
  $numCluster \leftarrow \hat{c}_{[j]}$
  //Se asocia a ese alcanzable al primer cluster cercano que se encuentre entre los clusters de sus vecinos
  for $i \in [V_{[j]}]$ **do**
    if $\hat{c}_{[i]} \in C$ **then**
      $c_{[j]} \leftarrow c_{[i]}$
    end if
  end for
  //Si sus vecinos no tienen cluster se les asocia como ruido
  if $numCluster = c_{[j]}$ **then**
    $c_{[j]} \leftarrow -1$
  end if
end for
Return $\hat{c}$

4. Diseño

En la figura 4 se expresa de una manera gráfica el proceso principal por el que pasan los datos para ser agrupados y evaluados. El proceso ha sido diseñado de una manera modular de modo que los módulos sean reutilizables para diferentes tareas del proyecto, como el proceso principal, el apartado de optimización de parámetros o el módulo de inferencia de nuevas instancias *test*.

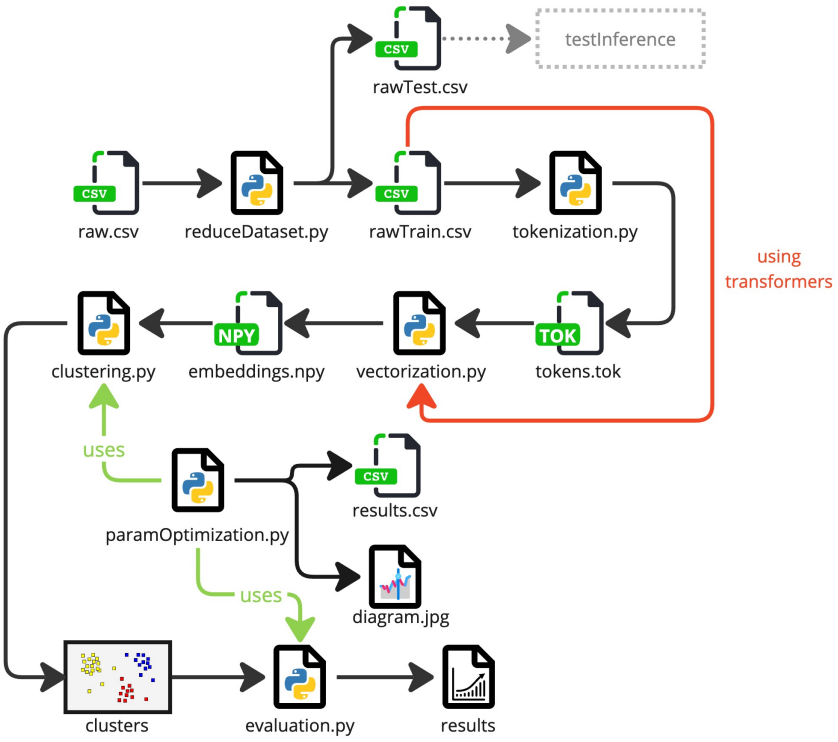

Figura 4: Proceso principal

Además de la linea de ejecución principal el proyecto cuenta con un módulo de inferencia. En este se expresa de manera gráfica el proceso por el que pasan los datos para ser asignados a *clusters* creados previamente.

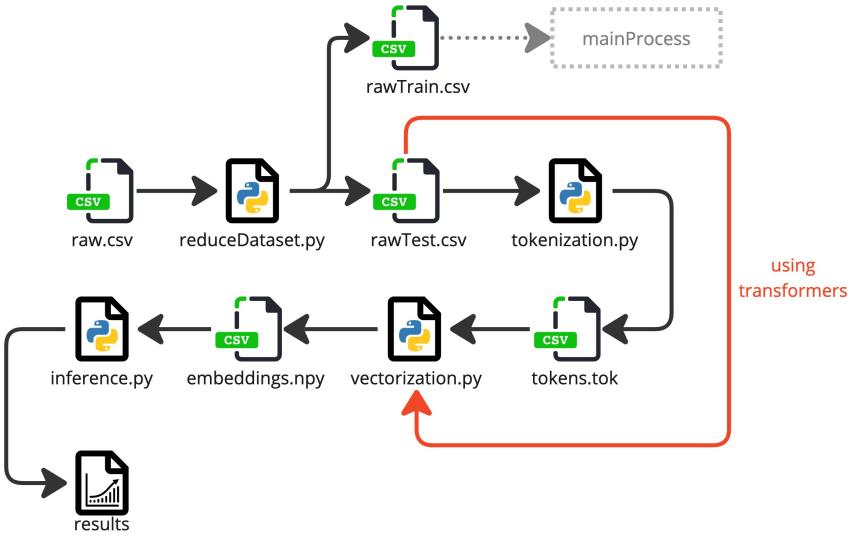

Figura 5: Proceso de inferencia

4.1. Reducción del dataset

El *dataset* al completo consta de más de doscientas mil instancias. Por un lado, se ha hecho una separación entre *train* y *test* para el proceso de *clustering* y más adelante el proceso de clasificación de nuevos textos. Por otro, cada uno de estos grupos se han reducido considerablemente, puesto que por tiempo y coste computacional, sería inviable procesar más instancias.

Finalmente se han mantenido diez mil instancias para el proceso de *clustering*. A pesar de no tener ningún significado relevante a la hora de hacer *clustering*, se ha mantenido una proporción bastante igualitaria entre instancias de las clases *"suicide"* y *"non-suicide"* como se puede ver en la figura 6. De igual forma, para la obtención del subconjunto *test* se ha seguido el mismo criterio.

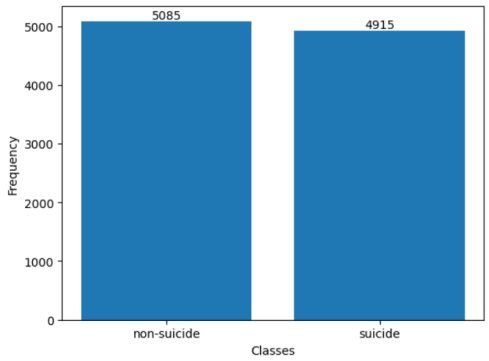

Figura 6: Distribución de las instancias en el conjunto *train* en las clases.

4.2. Vectorización

A la hora de vectorizar los datos, en una primera instancia se planteó directamente llevar a cabo el *"Doc2Vec"*, descartando *"BoW"*, *"TF-IDF"* y *"Word2Vec"*. Como se ha explicado en el apartado teórico son técnicas de menor complejidad, y por ello se han mantenido en un segundo plano.

4.2.1. Limpieza para la vectorización *"Doc2Vec"*

Tratándose de una tarea de *clustering*, solo se han de tener en cuenta los textos de cada instancia. El *ID* en este caso es irrelevante, ya que, antes del proceso *"Doc2Vec"* es necesario re-etiquetar las instancias. Por otro lado, la clase de cada instancia (*"suicide"* o *"non-suicide"*), sólo será necesaria en caso de realizar una evaluación externa utilizando por ejemplo una matriz de tipo *ClassToCluster*.

Por ello, una vez cargado el *dataframe* haciendo uso de la librería *"pandas"*, se han conservado únicamente los textos de cada instancia, es decir, el segundo atributo. Para el preprocesado de estos textos se ha hecho uso de las librerías *"spacy"* y *"emoji"* de *Python*.

Más concretamente, dentro de *"spacy"* se ha hecho uso del modelo previamente entrenado de procesamiento de lenguaje natural *"en_core_web_sm"*, ya que es uno de los más pequeños y ligeros disponibles para el inglés. Además, incluye varios componentes de tokenización, análisis gramatical

y etiquetado de partes de la oración, entre otros, que han resultado de gran utilidad[23]. Para ser exactos, se han aplicado los siguientes filtros (explicados en section 2):

- Tokenización
- Lematización
- Limpieza de datos
- Limpieza de *stopwords*
- Supresión de palabras demasiado cortas

Además de estos procesos propios del modelo, usando el *pipe* del paquete (*"emoji"*), los iconos se han transformado en cadenas de palabras con su significado. Por ejemplo, una cara feliz se traduciría a *"smiley::face"* que más adelante se transformaría en *"smiley face"* gracias a un *"replace"*.

4.2.2. *"Transformers"*

Tras realizar el proceso de *clustering* con la vectorización *"Doc2Vec"* no se han obtenido resultados concluyentes. Habiendo descartado que no se obtengan debido a un error de implementación en el algoritmo, se ha procedido a emplear una vectorización basada en *"Transformers"*. Como se mencionaba en la sección 2.1.2, al utilizar un mecanismo de atención, se prevé que los *embeddings* generados por los *"Transformers"* serán más representativos del tema principal de los textos.

Para llevar todo esto a cabo, se ha utilizado el modelo *"Twitter-XLM-Roberta-base"*[2]. Este modelo preentrenado tiene las siguientes características:

- Es un modelo XLM (*Cross-lingual Language Model Pretraining*) es decir, está entrenado con múltiples idiomas.
- Esta basado en el modelo optimizado *"RoBERTa"*[13] del LLM (*Large Language Model*) *BERT*[4] creado por Google en 2019. Estos modelos tipo *"transformer"*, han sido entrenados con una cantidad masiva de datos y son de gran utilidad a la hora de mejorar nuestras representaciones vectoriales.
- El modelo está entrenado con alrededor de ciento noventa y ocho millones de *tweets* en diferentes idiomas. Esta es una de las características por las cuales se ha decidido usar este modelo, teniendo en cuenta que el *dataset* de la tarea también está formado por *tweets*.

Este modelo se ha cargado desde *huggingface.co* y se ha utilizado en el código Python mediante la librería *"transformers"* que presenta la misma web.

El proceso de generar los *embedding* mediante los *"Transformers"* sigue los siguientes pasos:

1. **Tokenización**: La tokenización se lleva a cabo mediante un tokenizador del propio modelo. Este tokenizador está preparado para tokenizar de una manera especial y diferente a la que se ha estado viendo hasta ahora, la cual es preferible antes de crear los *embedding*. La figura

7 muestra el proceso de tokenización de una frase de ejemplo que como se puede apreciar, no recibe ningún tratamiento antes de pasar por el tokenizador.

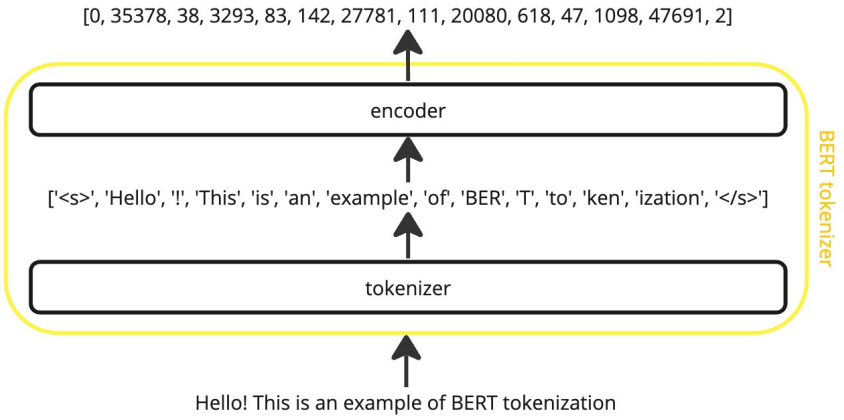

Figura 7: Ejemplo de tokenización con *BERT*

Como se verá más adelante, el modelo BERT admite *inputs* de 512 *tokens* por lo que si se generan más de ese número de *tokens*, se ha optado por truncar la salida, es decir, utilizar solamente los 512 primeros *tokens*. Existen otros métodos para tratar textos que generan más de 512 *tokens* pero truncar la salida es una opción sencilla y que suele tener buen rendimiento.

2. **Vectorización**: Una vez generados los *tokens* y codificados con su *ID*, estos *embeddings* se le han dado como *input* al modelo *"Twitter-XLM-Roberta-base"*. Este modelo, al ser un modelo *BERT*, tiene un *input* de 512 *tokens*, por lo que si el texto tiene menos de 512 *tokens*, se le añadirá un *padding* hasta completar el formato de *input*. Esto se puede ver en la figura 8 que a diferencia de la salida del mismo ejemplo de la figura 7, tiene más *tokens*.

   Como salida, el modelo genera un *embedding* de 768 dimensiones.

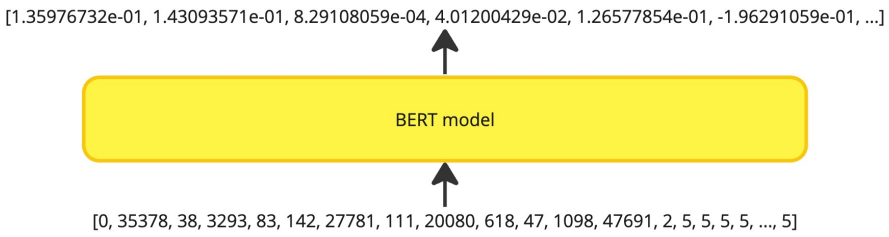

Figura 8: Ejemplo de vectorización con *BERT*

4.3. Ajuste de parámetros

4.3.1. *"Doc2Vec"*

Para utilizar esta vectorización, primero se han etiquetado los documentos con el formato *"TaggedDocument"*, puesto que es el único formato que acepta el *"Doc2Vec"*. A la hora de generar los *embedding* de cada texto se han fijado varios hiper-parámetros[5]:

- ***DM***: A este parámetro se le ha establecido en todos los casos el valor numérico 1 (*distributed memory*), de forma que se tenga en cuenta el contexto de cada *token* en el texto.[20]
- ***Epochs***: Determina al número de iteraciones a las que es sometido el *dataset*. El valor de los *embeddings* se recalcula varias veces durante el proceso de vectorización. Partiendo de un vector inicial se itera sobre el *dataset* el número de veces que el parámetro *epochs* indique recalculando los vectores. Se han establecido 100 *epochs*.
- ***Negative***: Este parámetro se ha establecido en todos los casos a 0, de forma que no tenga en cuenta el ruido generado (palabras que no aparecen en el modelo).
- ***Window***: El parámetro *window* especifica cuántos *token* anteriores y posteriores al actual se tienen en cuenta para el contexto. Se le ha establecido el valor numérico 2, para evitar un coste computacional demasiado alto.
- ***Workers***: Este parámetro es de ayuda a nivel computacional, ya que cuanto mayor sea el parámetro, más núcleos de la *CPU* utilizará para procesar en paralelo los textos. En este caso se ha establecido a 4, ya que es el máximo de núcleos de los que se disponen.

Adicionalmente, se ha optimizado el siguiente hiper-parámetro de *"Doc2Vec"*:

- ***Vector size***: Define las dimensiones de los *embedding*, esto es, el número de componentes que contiene cada *embedding*. La optimización de este parámetro depende del *dataset* sobre el que se aplique, y aunque la intuición tiende a dar una idea de que a mayor número de dimensiones mejor representados quedarán los datos, cabe la posibilidad de que a partir de un número de dimensiones, aumentarlas resulte en fenómenos como "la maldición de la dimensionalidad", donde la dispersión de los datos dificulta su gestión y agrupación [9].

Como se menciona en [12], las pruebas experimentales que se llevaron a cabo para demostrar la efectividad de *"Doc2Vec"*, fueron ejecutadas con una dimensionalidad de 400 sobre un *dataset* de 100.000 instancias. Se puede comprobar cómo en diferentes artículos de T.Mikolov [15] [14] no se obtiene una mejoría notable al aumentar este número. Estas pruebas siempre se hacen en un espacio de dimensionalidad 100-400, por lo que en esta tarea, se han probado con diferentes dimensiones en ese rango: 100, 150, 200 y 250, para ser exactos.

4.3.2. *"DBSCAN"*

Una vez obtenidos los *embeddings* (tanto con el *"Doc2Vec"* como con los *"Transformers"*), es necesario realizar un ajuste de parámetros final en el modelo de agrupación, de manera que interprete la representación vectorial de manera adecuada y sea capaz de crear agrupaciones con ellos. Para ello se han tenido en cuenta tres métricas: ε, *minPoints* y la métrica para calcular las distancias.

Inicialmente, se pensó en utilizar la distancia euclidiana como métrica de distancia, ya que es la utilizada por el algoritmo *"DBSCAN"* por defecto. Sin embargo, esto resultó en varios problemas a la hora de agrupar, puesto que al estar tan juntas las instancias entre sí, no era tan representativa esta distancia como la similitud del coseno. Este cálculo se realiza en espacios positivos de alta dimensión, por lo que es de gran utilidad en la recuperación de información valiosa y minería de textos [1]. Tiene en cuenta los siguientes puntos:

1. **Invariación de la magnitud**: La medida de similitud del coseno es invariante a la magnitud de los vectores, es decir, no importa su dimensión, ya que la similitud del coseno solo se ve afectada por la dirección relativa de los vectores.
2. **Sensibilidad a la dirección**: En muchas aplicaciones, la dirección de los datos es más importante que su distancia absoluta. En procesamiento de texto, los documentos pueden ser representados como vectores de términos, y la similitud del coseno mide cuán similares son dos documentos en función de la frecuencia de palabras. La distancia euclidiana puede ser sensible a la longitud de los documentos, mientras que la similitud del coseno se enfoca en la orientación de los vectores, lo que es más apropiado en este contexto.
3. **Mejor manejo de datos dispersos**: La similitud del coseno tiende a funcionar mejor que la distancia euclidiana cuando se trabaja con datos de alta dimensionalidad. Aquí es común que los puntos estén espaciados lejos unos de otros en términos de distancia euclidiana, pero aún pueden ser similares en términos de similitud del coseno si comparten una dirección similar en el espacio de características.
4. **Enfocado en la densidad**: *"DBSCAN"* se basa en la densidad de los puntos para encontrar grupos, y la similitud del coseno se alinea bien con este enfoque, centrándose en la orientación y la similitud de direcciones.

Para el algoritmo planteado en esta tarea se ha decidido mantener la distancia euclidiana, ya que, el coste de calcular las distancias con la similitud del coseno de la librería *"numpy"* es considerablemente mayor en los aspectos de tiempo y recursos.

En el ajuste de los otros dos parámetros, se ha hecho uso de la librería *"Optuna"* [16], que permite hacer pruebas proponiendo un rango de ε y *minPts* de manera automática tras establecer los saltos a dar entre valores (similar al *learning rate* utilizado en redes neuronales).

A la hora de establecer los rangos, el mínimo de puntos se ha puesto de forma que abarque un rango bastante grande (de 2 a 30 puntos), ya que no había forma de prever cuál sería el parámetro óptimo. Sin embargo, para el valor ε, del *"DBSCAN"* se han registrado cuáles eran las distancias

coseno entre instancias más comunes, concluyendo un rango óptimo para ε de [0.9, 0.1] (figura 9).

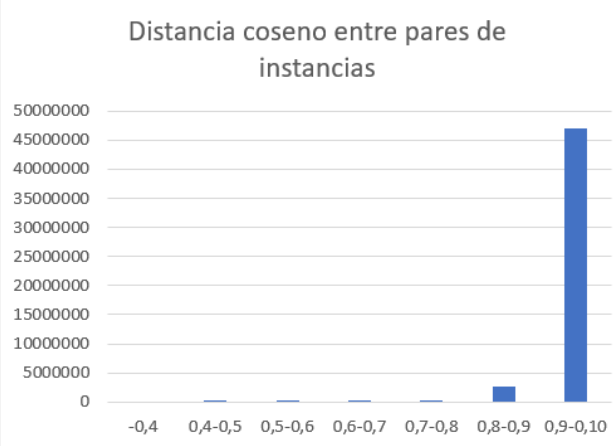

Figura 9: Distancia coseno entre pares de instancias.

Y lo mismo con la distancia euclidiana para el algoritmo implementado, sacando como conclusión que el ε óptimo se encuentra en el rango (0, 5] (figura 10).

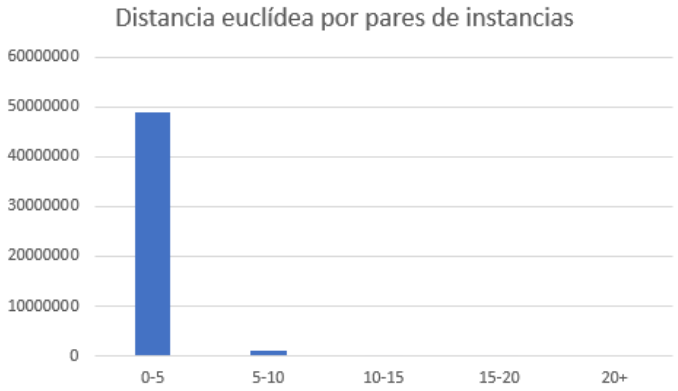

Figura 10: Distancia euclidiana entre pares de instancias.

Una vez obtenidos los rangos más relevantes para cada métrica, se le pasarán a *"Optuna"*, junto con el algoritmo para el que se quieren optimizar. Para saber en base a qué debería optimizar la pareja de valores ε y *minPoints*, se ha fijado que haga lo siguiente:

- Maximizar el valor de *Silhouette Score*. Esta métrica oscila en un rango [-1, 1] donde un valor cercano a -1 implicaría *clusters* con instancias muy dispersas entre sí y *clusters* solapados entre ellos. Mientras que un valor cercano a 1, implicaría una mejor cohesión intra-cluster y mayor separación inter-cluster.
- Minimizar el número de *clusters*. De esta forma se evita que cree un *cluster* por instancia, ya que, haciendo eso el valor de la métrica anterior también sería alta, pero la agrupación sería incorrecta.

- Maximizar la media de puntos por *cluster*. De esa forma será más probable que se agrupen un mayor número de instancias por *cluster*, disminuyendo el ruido.
- Maximizar el mínimo de instancias en un *cluster*. Haciendo esto, evitamos que se creen *clusters* con el número de *minPoints* o al menos reducimos la probabilidad, creando *clusters* de tamaños más balanceados.

Para la implementación, ha servido de ayuda los ejemplos que se presentan en los artículos [17] [19] [21].

4.4. Evaluación

Una vez finalizado el proceso de *clustering* se procede a evaluar los resultados obtenidos. Para ello existen diferentes métricas y procedimientos. Por una parte se ha optado por realizar una evaluación externa utilizando una matriz de confusión *ClassToCluster* sumado a la creación de *WordClouds*.

Complementario a los *WordClouds* se analizará una muestra de instancias por cada uno de los *clusters*. De tal forma que si el "factor comun" de los textos no se ve reflejado en las palabras contenidas en él pueda detectarse mediante la visualización en crudo de las instancias.

A partir de la matriz de confusión se pretende identificar si la agrupación natural de los datos que se haya podido dar durante el proceso tiene algún tipo de relación con la etiqueta de clase. En cambio, se emplean *WordClouds* para comprobar la naturaleza de las agrupaciones que realiza el algoritmo de *clustering*. Visualizando las *keywords* de cada uno de los *clusters* es posible deducir el factor común de los textos pertenecientes a esa agrupación.

Finalmente, para evaluar la eficacia del algoritmo propio, se pretende realizar una comparación entre este y el *"DBSCAN"* de la librería *"Sklearn"* basada en la métrica de evaluación interna conocida como *Silhouette Score*.

4.5. Inferencia

Una vez finalizado el proceso de *clustering* y habiendo inferido una agrupación, se procede a aplicar esa agrupación para extraer información de un nuevo conjunto de datos.

Para la aplicación del modelo inferido se han utilizado dos conjuntos de *clusters* diferentes, por una parte, los *clusters* creados por el propio *"DBSCAN"*, y por otra parte, los creados por el algoritmo implementado.

Para ello, se ha utilizado un conjunto de datos que consta de dos mil instancias, al cual se le agregan varias instancias del conjunto de datos de entrenamiento. El objetivo de incorporar estas instancias previamente analizadas es comparar las nuevas asignaciones con lo inferido previamente.

También se han añadido varias instancias en otros idiomas, instancias que se sabe a que *cluster* deberían pertenecer (por ejemplo, URLs, dibujos...) e instancias que contienen palabras que no están en el vocabulario, instancias OOV (*Out Of Vocabulary*).

Para asignar una nueva instancia a un *cluster*, se ha aplicado la siguiente estrategia: a cada instancia del conjunto de prueba se le ha asignado el *cluster* de la instancia de entrenamiento más cercana. Para ello, se han calculado todas las distancias entre la instancia de prueba y las instancias de entrenamiento.

Además, se ha probado a seleccionar las k instancias mas cercanas (algoritmo *"k-NN"*) a las instancias de prueba y asignar a estas el *cluster* más frecuente entre los vecinos. Pero se ha observado que con los datos utilizados el rendimiento no ha sido satisfactorio, debido a que hay un *cluster* que contiene la gran mayoría de instancias, por lo que tiende a asignar las nuevas instancias a éste, lo que en algunos casos, es incorrecto.

5. Resultados experimentales

5.1. Implementación del algoritmo

En primera instancia se ha procedido a replicar el algoritmo presentado en el artículo de publicación de *"DBSCAN"* [7]. El algoritmo planteado funciona correctamente y ofrece los mismos resultados que la implementación contenida en la librería *"Sklearn"*. No obstante, la implementación propia obtiene un rendimiento peor cuestiones de tiempo y memoria que la original.

Sin embargo, durante el proceso de implementación del algoritmo se ha desarrollado otra versión de un algoritmo basado en densidad que no sigue el mismo pseudocódigo que *"DBSCAN"* (section 3). El algoritmo de *"DBSCAN"* itera todas las instancias hasta dar con un *Core Point* (explicado en subsubsection 2.3.2), que se agrupa en un *cluster* junto con todos los *Border Points* asociados a este. Dentro de esos *Border Point*, busca las instancias que también coincidan con los criterios de un *Core Point*, expandiéndolos a su vez y añadiendo todos sus vecinos al mismo grupo, así hasta no detectar mas *Core Points*.

En cambio, el algoritmo planteado en esta tarea es una versión simplificada que limita la agrupación de una manera sencilla. Desde un primer momento, se recogen todas las instancias que cumplan los requisitos para ser *Core Point*. Por ejemplo, en la figura 11, se ve cómo se han considerado *Core Point* las instancias azules, que tienen sus *Border Point* (puntos verdes) dentro de la circunferencia ε y los *Outliers*, en este caso los amarillos.

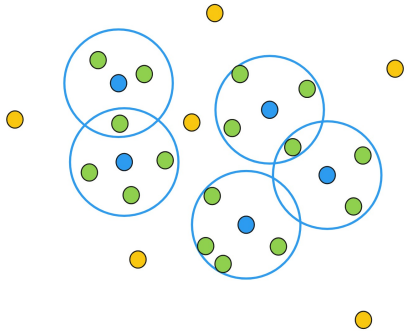

Figura 11: Instancias con ε representada con el círculo de minPoints=3.

Más tarde, estos se iteran de forma que recojan cada *Core Point* con sus *Border Point* en el mismo *cluster*, pero no hace comprobaciones para ver si se podría expandir ese *cluster*. Como se ve en la figura 12, se iteran esos *Core Point* hasta crear un total de 5 *clusters* (del 0 al 4).

Sin embargo, puede suceder que después de varias iteraciones, haya *Core Points* que se hayan quedado solos en un *cluster* o con menos instancias que lo que marque el *minPts* (esto pasa porque una vez una instancia se asocia a un *cluster* no se comprueba si también pertenecía a otro). Para evitar *clusters* que no cumplan las características mínimas, estas instancias se reagrupan en el *cluster* válido más cercano. Un ejemplo de esto sería el *cluster* 4 de la figura 12, que reagrupa sus instancias a la agrupación más cercana.

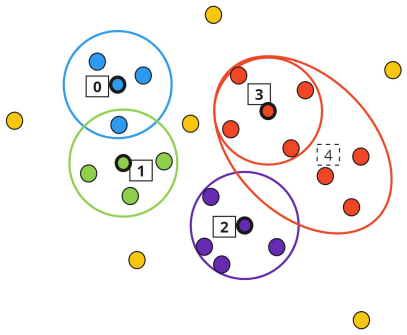

Figura 12: Instancias con ε representada con el círculo de minPoints=3.

De esta forma, se han obtenido resultados que podrían considerarse en cierto modo concluyentes. Al analizar los resultados mediante *WordClouds* y las muestras de instancias por cluster, se pueden apreciar agrupaciones naturales de diferentes tópicos. Esto se debe a que se fuerza la creación de varios grupos más pequeños que con el *"DBSCAN"* original, serían parte de un solo *cluster* más grande.

Pese a conseguir más *clusters*, en cualquiera de los dos algoritmos (el implementado en esta tarea y el oficial) los resultados obtenidos con vectorización *"Doc2Vec"* no han sido demasiado representativos. Ambos algoritmos ofrecen un *cluster* con gran parte de las instancias, y alguna agrupación mucho más pequeña (varias en caso del implementado), además de haber una gran cantidad de instancias consideradas ruido.

En vista de la escasez de resultados se plantea la posibilidad de usar otro método de vectorización como los *"Transformers"*. Con el uso de esta nueva técnica de vectorización los resultados obtenidos distan bastante de los que se obtienen empleando *"Doc2Vec"* (ver figura 13). En este caso se consiguen agrupaciones naturales variadas tanto con el algoritmo original de *"Sklearn"* como con la aproximación a *"DBSCAN"* propia.

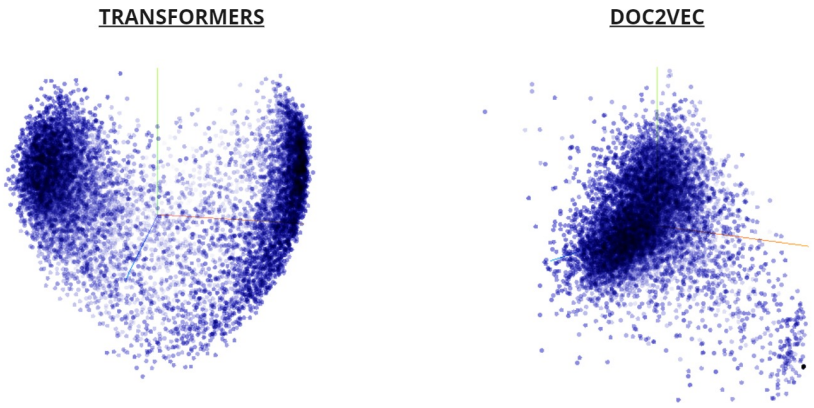

Figura 13: Visualización de las instancias vectorizadas con *"Doc2Vec"* y con *"Transformers"*.
(Fuente: *Embedding Projector*)

Incluso utilizando los *"Transformers"*, las agrupaciones siguen siendo demasiado dispares en tamaño para el *"DBSCAN"* original, por lo que también se plantea la posibilidad de cambiar la métrica para calcular la distancia entre vectores, habiendo usado hasta ese momento la euclidiana. La razón para cambiarla es que, en un espacio de altas dimensiones la distancia euclidiana pierde su relevancia, siendo mucho más adecuado usar la similitud o distancia de coseno.

Si bien es cierto que la métrica basada en el coseno es mas adecuada para esta situación, como se ha explicado anteriormente, emplearla sobre el algoritmo de desarrollo propio resulta inabordable dados los recursos de los que se dispone. Por ello para la ejecución del algoritmo de densidad planteado en section 3 se ha seguido utilizado la distancia euclidiana.

5.2. Análisis critico y discusión de resultados

5.2.1. Resultados de clustering

A la hora de elegir finalmente los parámetros de ε y *MinPts* tanto para el algoritmo *"DBSCAN"* como para el implementado, se han contemplado varias posibilidades gracias a las pruebas con diferentes parámetros obtenidos por la librería *"Optuna"*. Mediante estas pruebas, se ha intentado obtener el par de parámetros que mantengan en balance entre el número de *clusters* y el *Silhouette* del modelo. Algunas de las pruebas realizadas son las que aparecen en la tabla 1.

Clusters	***DBSCAN Sklearn***			**Algoritmo implementado**		
	Epsilon	***Min Points***	***Silhouette Score***	***Epsilon***	***Min Points***	***Silhouette Score***
2	0.1252	5	**0.7368**	4.807	11	**0.4819**
3	0.0611	5	**0.6033**	4.761	6	**0.4106**
7	0.00838	5	**-0.0213**	4.242	7	**0.3044**
8	0.00485	5	**-0.0862**	3.324	12	**0.1482**
9	0.0071	5	**0.0016**	3.338	10	**0.1482**
12	-	-	**-**	2.567	12	**0.1306**

Tabla 1: Pruebas con los parámetros de ε y *MinPts* y su respectivo *Silhouette Score*.

Con estos resultados, se ha concluido que cuantas más agrupaciones se hagan, menor será el valor de la métrica, y por ende, tanto la separación inter-cluster como la intra-cluster de las instancias se verá encarecida (figura 14).

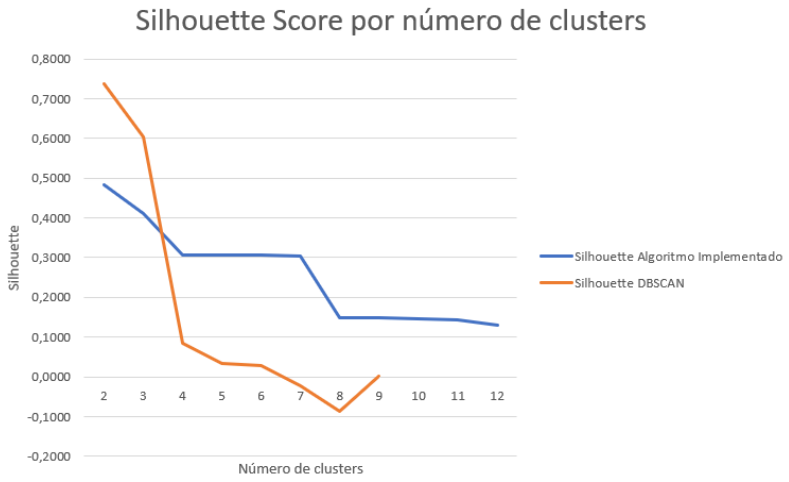

Figura 14: *Silhouette Score* por número de *clusters*.

Sin embargo, también se puede concluir mediante el muestreo de instancias y los *WordClouds*, cómo cuantos más *clusters* se obtienen, más concretas y definidas son las agrupaciones hechas en cuestión de temática. Es por eso que después de analizar los posibles resultados para ambos algoritmos dependiendo de los valores de ε y *MinPts*, se ha tomado la decisión de escoger los siguientes parámetros para cada uno de ellos (tabla 2):

DBSCAN original		**Algoritmo Implementado**	
Min Points	***Épsilon***	***Min Points***	***Épsilon***
5	0.0071	12	2.567

Tabla 2: Parámetros de ε y *MinPoints* definitivos para cada algoritmo.

Con éstos parámetros se han conseguido un total de 9 *clusters* para el algoritmo de *"Sklearn"* y 12 para el de desarrollo propio. Se ha escogido este número de agrupaciones para cada uno, puesto que menos *clusters* implican agrupaciones incoherentes y poco evidentes entre instancias. Tampoco se han planteado más *clusters*, ya que los tipos de grupo comienzan a ser redundantes en el algoritmo implementado y en el caso del *"DBSCAN"* no parece encontrar más agrupaciones relevantes entre instancias.

Además, se han elegido estos valores en concreto para el *"DBSCAN"* también en parte porque como aparece en la figura 14, pese a que el valor de la métrica cae a medida que el número de *clusters* sube, en este punto el valor de la métrica ha vuelto a ser positiva.

Para medir la similitud entre los dos algoritmos se han usado diferentes métricas de evaluación externas. Los resultado obtenidos se muestran en la tabla 3:

Métrica	*Jaccard*	*ARI*	*Fowlkes-Mallows*
Valor	0.7692	0.4429	0.7092

Tabla 3: Métricas de evaluación para comparar ambos algoritmos.

1. ***Rand Index* (ARI)**. Esta métrica mide la similitud entre las particiones de los dos resultados de *clustering*. El valor varía entre 0 y 1, y cuanto más similares sean las particiones más cercano a 1 será el valor. Un valor de 0.4429 indica una similitud moderada entre las particiones generadas por los dos algoritmos.
2. ***Jaccard Score***. Esta evalúa la similitud entre dos conjuntos. Es decir, la similitud entre los conjuntos de elementos que se agrupan de la misma manera por dos algoritmos diferentes. Un valor de 0.7692 indica una similitud relativamente alta entre las asignaciones de *clusters* de los dos algoritmos.
3. ***Fowlkes Mallows Score***. Mide la similitud entre dos conjuntos de agrupamientos al considerar la relación entre los pares de elementos. Un valor de 0.7092 indica una similitud bastante alta entre las particiones generadas por los dos algoritmos.

Es decir, ambos algoritmos tienen una similitud bastante alta en cuanto a agrupaciones se refiere. Esto también se puede apreciar mediante una matriz *"pairwise"*, viendo de qué manera agrupan los pares de instancias cada algoritmo. Cuanto mayor sea la cantidad de instancias que se encuentren en la diagonal principal, más similares serán, ya que esto implica que esos pares de instancias siempre se clasifican en el mismo *cluster*, o siempre en *clusters* diferentes. La matriz para ambos algoritmos es la de la tabla 4.

		Algoritmo implementado	
		$=$	$\neq$
DBSCAN	$=$	16.938.110	7.638.481
	$\neq$	6.271.712	19.146.697

Tabla 4: *"Pairwise matrix"* de ambos algoritmos.

Como se puede ver claramente, más del 72 % de los pares de instancias se han agrupado de la misma manera en ambos algoritmos.

Además de con métricas objetivas, gracias a los *WordClouds* creados y al muestreo de instancias por *cluster*, se pueden comparar de forma más subjetiva ambos algoritmos.

Por un lado, vemos que en el algoritmo implementado se han creado los siguientes *clusters* (tabla 5):

Algoritmo implementado || Epsilon: 2.567 || Min Points: 12		
Cluster	**Instancias**	**Tema**
-1	162	Ruido
0	5505	Frases largas + Desesperación + Pérdida
1	4010	Frases cortas + Depresión + Malestar + Suicidio
2	64	Familia rota + Suicidio + Mensajes terminales
3	45	Eventos de clase o del trabajo
4	100	Links o cadenas de números
5	12	Textos cortos amenazantes
6	15	Fiesta + Alcohol + Depresivos
7	32	Links a memes o juegos
8	14	Saludos o felicitaciones
9	15	Muerte + Pidiendo ayuda + Edades
10	12	Letras de canciones o poemas depresivos
11	14	Estudios + Trabajo + Política

Tabla 5: *Clusters* del algoritmo implementado.

Los temas que aparecen se han deducido primero observando los *WordClouds* y obteniendo muestras de instancias por cada *cluster* y leyéndolas más a fondo, sobre todo en los casos que los *WordClouds* no eran demasiado representativos. Cabe mencionar que para realizar los *WordClouds* ha sido necesario mantener la limpieza de tokens pese a usar *"Transformers"*, puesto que no afecta al proceso de *clustering*, pero sí ayuda a una mejor interpretación de los temas.

Por ejemplo, tres imágenes muy representativas serían las de los *clusters* tres, siete y ocho en este caso (figura 15), donde se ven a la perfección las palabras relacionadas a los temas dados.

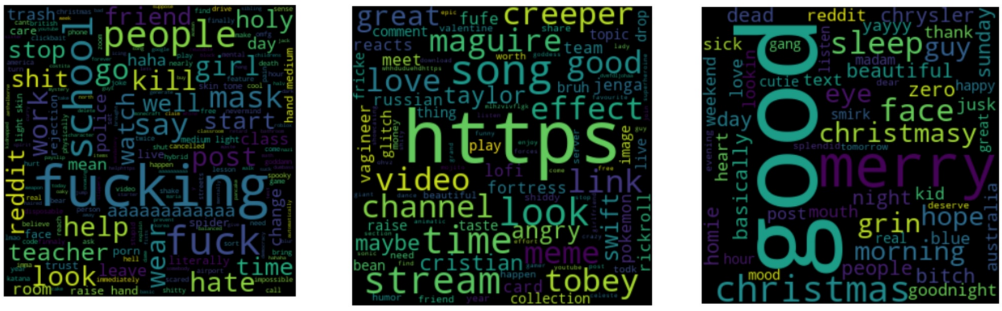

Figura 15: *WordClouds* de los *clusters* 3, 7 y 8 del algoritmo implementado.

En cambio, para otras agrupaciones donde estas palabras son menos representativas, ha sido necesario conseguir muestras de los clusters, como en el caso de los *clusters* 0 y 1, donde se agrupan instancias de este tipo:

- ***Cluster* 0:** *"Ex Wife Threatening SuicideRecently I left my wife for good because she has cheated on me twice and lied to me so much that I have decided to refuse to go back to her. As of a few days ago, she began threatening suicide. I have tirelessly spent these paat few days talking her out of it and she keeps hesitating because she wants to believe I'll come back. I know a lot of people will threaten this in order to get their way, but what happens if she really does? What do I do and how am I supposed to handle her death on my hands? I still love my wife but I cannot deal with getting cheated on again and constantly feeling insecure. I'm worried today may be the day she does it and I hope so much it doesn't happen."*
- ***Cluster* 1:** *"It ends tonight.I can't do it anymore. I quit."*

Mientras que en el algoritmo de *"DBSCAN"* se pueden ver los siguientes resultados (tabla 6):

DBSCAN || Epsilon: 0.0071 || Min Points: 5		
Cluster	***Instancias***	***Tema***
-1	5110	Ruido
0	4801	Frases largas + Desesperación + Pérdida
1	11	Juegos en línea + Discord
2	28	Palabra "filler"
3	17	Emoji + Link
4	14	Palabra "award" (premio)
5	5	Palabra "Reddit"
6	4	Cómo relacionarse o ligar
7	5	Palabra "comment" (comentario)
8	5	Temas controversiales

Tabla 6: *Clusters* del algoritmo *"DBSCAN"*.

Aquí se ve claramente cómo a diferencia del algoritmo implementado, muchas instancias se han considerado ruido. Además, analizando estas instancias, se puede ver cómo muchas de ellas eran parte de en lo que el algoritmo implementado se consideraba el ***Cluster* 1**, tales como:

- *"Am I weird I don't get affected by compliments if it's coming from someone I know irl but I feel really good when internet strangers do it"*
- *"Finally 2020 is almost over... So I can never hear "2020 has been a bad year.^{ev}er again. I swear to fucking God it's so annoying"*
- *"i need helpjust help me im crying so hard"*
- *"It ends tonight.I can't do it anymore. I quit."*
- *"Been arrested - feeling suicidalEdit".*

Sin embargo, y a diferencia de en algoritmo implementado, se ven *clusters* con temas más concretos, como agrupaciones por palabras concretas, como "*award*", "*comment*" y "Reddit", entre otras; además de temas concretos como formas para ligar o temas controversiales (figura 16). Ambos algoritmos comparten las instancias del ***Cluster* 0** y el *cluster* de las URL.

Figura 16: *WordClouds* de los *clusters* 4, 5 y 8 de *"DBSCAN"*.

5.2.2. Inferencia de nuevas instancias

Para inferir el nuevo conjunto de datos de prueba se han utilizado los *clusters* creados por los dos algoritmos (el implementado en esta tarea y el incluido en la libreria *"Sklearn"*).

Primero, se han añadido al conjunto de prueba las siguientes instancias:

- Test[2000]: *"348109, https://egela.ehu.eus/login/index.php, non-suicide"*
- Test[2001]: *"348110, no estoy bien y estoy pensando en suicidarme, suicide"*
- Test[2002]: *"348111, nire buruaz beste egin nahi dut, suicide"*
- Test[2003]: *"348112, ajahsdkdk akdjjddjkdk kadsjhdyrwuronu, non-suicide"*

Además, como ya se ha mencionado, se han añadido varias instancias del conjunto de entrenamiento (las instancias Test[2004-2007]).

El algoritmo *"DBSCAN"* ha agrupado las instancias del conjunto de datos de entrenamiento en 9 *clusters*, de 0 a 8. Estas son las distancias calculadas y *clusters* asignados para algunas de las instancias (tabla 7):

Tipo e índice de la instancia	Cluster original	Cluster inferido	Distancias	Distancia e índice a instancia mas cercana
Test 0	-	0	[0.00692236 0.04815924 ...0.05279809 0.02892232]	Train [4861]: 0.0052016973
Test 1253	-	4	[0.02566284 0.01962507 ...0.02873564 0.04986429]	Train [908]: 0.0052666664
Test 2000	-	-1	[0.08638364 0.0482446 ...0.05199772 0.11761403]	Train [5047]: 0.015028298
Test 2001	-	-1	[0.09149384 0.03749883 ...0.03732944 0.12812108]	Train [2246]: 0.018589318
Test 2002	-	-1	[0.09424865 0.04569519 ...0.04886806 0.13516021]	Train [2246]: 0.026731074
Test 2003	-	-1	[0.1201542 0.09969419 ...0.09958231 0.1415909]	Train [927]: 0.026137471
Train 908 (Test 2004)	4	4	[0.02454859 0.02200723 ...0.03301752 0.04744589]	Train [908]: -1.1920929e-07
Train 1894 (Test 2006)	3	3	[0.09468049 0.0742119 ...0.07123268 0.11437774]	Train [1894]: 0.0

Tabla 7: Inferencia del algoritmo *"DBSCAN"*.

El algoritmo implementado ha agrupado las instancias del conjunto de datos de entrenamiento en 12 *clusters*, de 0 a 11. Estas son las distancias calculadas y *clusters* asignados para algunas de las instancias (tabla 8):

Tipo e índice de la instancia	**Cluster original**	**Cluster inferido**	**Distancias**	**Distancia e índice a instancia mas cercana**
Test 0	-	0	[0.00692236 0.04815924 ...0.05279809 0.02892232]	Train [4861]: 0.0052016973
Test 1253	-	0	[0.02566284 0.01962507 ...0.02873564 0.04986429]	Train [908]: 0.0052666664
Test 2000	-	7	[0.08638364 0.0482446 ...0.05199772 0.11761403]	Train [5047]: 0.015028298
Test 2001	-	1	[0.09149384 0.03749883 ...0.03732944 0.12812108]	Train [2246]: 0.018589318
Test 2002	-	1	[0.09424865 0.04569519 ...0.04886806 0.13516021]	Train [2246]: 0.026731074
Test 2003	-	1	[0.1201542 0.09969419 ...0.09958231 0.1415909]	Train [927]: 0.026137471
Train 360 (Test 2004)	2	2	[0.037745 0.10276937 ...0.10033143 0.01027799]	Train [360]: 0.0
Train 603 (Test 2006)	4	4	[0.0594492 0.07950979 ...0.08742547 0.06305248]	Train [603]: -1.1920929e-07

Tabla 8: Inferencia del algoritmo propio.

Observando los resultados obtenidos, se puede ver que las instancias del conjunto de entrenamiento han sido clasificadas correctamente en ambos casos. La distancia a la instancia más cercana debería ser 0.0 puesto que es la misma instancia, pero debido a la representación finita de números y a errores de redondeo, esto no es así en algunas de las instancias.

Las instancias añadidas han sido clasificadas de manera diferente. En el caso del *"DBSCAN"*, las cuatro instancias han sido clasificadas como ruido, pese a hablar sobre suicidio o ser un URL, que deberían estar en su *cluster* correspondiente. Al contrario, el algoritmo implementado a agrupado las cuatro instancias en los *clusters* correctos.

5.3. Rendimiento del software

Independiente al correcto funcionamiento del software desarrollado, una cuestión importante a tener en cuenta es el rendimiento que ofrece. Se considera que en situaciones como la gestionada en este proyecto, donde se ha de trabajar con grandes volúmenes de datos, el coste computacional toma una gran relevancia.

En el software desarrollado se da la posibilidad de ejecutar el algoritmo de desarrollo propio y el implementado en la librería *"Sklearn"*. Como ya se ha mencionado, con el algoritmo propio la eficiencia y el rendimiento se ven bastante reducidos, por una gestión no demasiado eficiente del uso de memoria.

Sin embargo, en cuanto a los procesos auxiliares, como la limpieza de textos, vectorización de los datos, calculo de distancias, etc. se ha establecido la política de guardar los resultados para poder cargarlos en futuras ocasiones. De modo que el programa pueda ser ejecutado de manera modular cargando los *outputs* del resto del módulos de manera independiente, evitando así la necesidad de ejecutar el proceso en su totalidad cada vez que se requiere de la ejecución de un módulo.

De no guardar estos datos, el tiempo que tarda cada proceso aumenta considerablemente con respecto a la cantidad de instancias utilizadas (ver figura 17). Esto se aprecia especialmente en el algoritmo implementado en la tarea, donde al pasar de diez mil instancias al doble (veinte mil), le lleva unas veinte veces más de tiempo entrenar el modelo. Esto se debe a la cantidad de distancias a calcular que existen, ya que se calcula la distancia por cada pareja de instancias una vez, es decir, al haber veinte mil instancias, las distancias a calcular ascienden a casi doscientos millones de distancias.

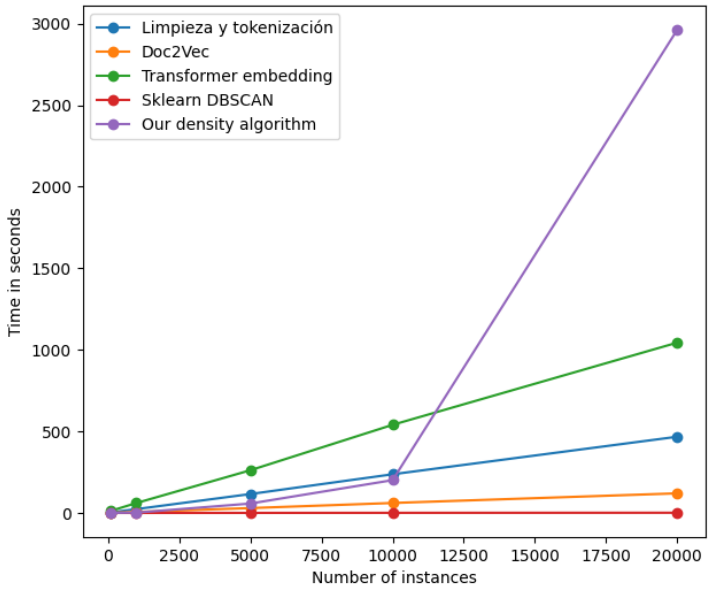

Figura 17: Rendimiento del software en diferentes tareas

6. Conclusiones

6.1. Conclusiones generales

Tras el desarrollo de esta tarea de investigación, se han determinado diferentes conclusiones relativas a *clustering* basado en densidad, el propio algoritmo *"DBSCAN"* de *"Sklearn"*, la evaluación de los resultados obtenidos de un proceso de clustering y los procedimientos a seguir para inferir nuevas instancias.

En primer lugar se ha observado que los algoritmos basados en densidad requieren de unos datos que sean favorecedores a esta técnica para encontrar resultados. A diferencia de técnicas como *"K-means"*, donde se establece el numero de *clusters* de manera anticipada, en los algoritmos basados en densidad cabe la posibilidad de que si los datos están demasiado agrupados, formando un único bloque de alta densidad estos, se agrupen en un solo *cluster*. Para evitarlo hay que parametrizar minuciosamente los argumentos del algoritmo de *clustering*.

Adicionalmente, en cuanto a la evaluación de los resultados obtenidos, se entiende que no hay una forma normalizada de evaluarlos. Existen métricas que sí están normalizadas, pero precisamente durante esta investigación ha sido de mayor ayuda hacer pequeños muestreos sobre los datos que observar los valores de las métricas estandarizadas.

Respecto a la introducción de nuevas instancias en el sistema, se ha concluido que en lo que respecta a este conjunto de datos lo mas adecuado es clasificar la instancia a inferir en el *cluster* correspondiente a la instancia de *train* mas cercana. Viene a ser equivalente a calcular un *"K-NN"* donde *'n'* es uno.

Finalmente, como conclusión general se puede destacar que esta tarea de investigación ha sido de gran utilidad para conocer y familiarizarse con los procesos a seguir en tareas de *clustering*, así como para enfrentar los problemas y dificultades que surgen durante un proceso de investigación en un área que se desconoce.

6.2. Propuestas de mejoras y trabajo a futuro

El tiempo del que se ha dispuesto para la realización de esta investigación es un factor determinante para calidad de los resultados obtenidos. Aunque el objetivo de la tarea se ha cumplido habiendo conseguido encontrar agrupaciones naturales en los datos sobre los que se ha trabajado, se considera que el trabajo puede ser extendido en algunos puntos de la implementación.

En primer lugar, debido a una gestión no demasiado eficiente de la memoria en el algoritmo de desarrollo propio, ha resultado inabordable computacionalmente el uso de la similitud coseno como métrica para la comparación vectorial. Siendo la distancia euclidiana incorrecta para el contexto de esta investigación se han conseguido resultados ciertamente concluyentes, sin embargo, seria conveniente optimizar el algoritmo para que sea computable empleando la similitud coseno.

Además, con los parámetros para el algoritmo de *clustering* que se han considerado óptimos en esta tarea, se puede observar que se forma siempre una agrupación muy grande respecto al resto.

Esto es, aparecerán múltiples *clusters* con un numero de instancias relativamente pequeño y un *cluster* grande donde se agrupen muchas instancias. En vista de esto, se considera la opción de aplicar el algoritmo de *clustering* nuevamente sobre los datos del *cluster* grande. De esta forma es posible que se encuentren sub-agrupaciones naturales dentro de los datos agrupados en el *cluster* de gran tamaño.

Respecto a el uso de *"Transformers"*, se ha explicado que el modelo empleado solo acepta un máximo de 512 *tokens*. En los casos donde el texto a tokenizar supera esa cantidad, en vez de truncar el texto perdiendo la información de los *tokens* que se descartan, sería conveniente aplicar alguna otra medida que fuese menos invasiva.

Por otro lado, existe la posibilidad de que la proximidad de los datos una vez vectorizados haya sido demasiado elevada debido a una limpieza no lo suficientemente exhaustiva. De modo que, sería conveniente explorar nuevos filtros para la limpieza del *corpus*. De esa forma, habría que cerciorarse de cual de las dos vectorizaciones sería más adecuada, si la realizada por los *"Transformers"* o la vectorización *"Doc2Vec"*.

Adicionalmente, en caso de precisar de un análisis más completo sobre los documentos y las relaciones entre sí, sería de gran valor desarrollar diagramas de cuerda o alguna estructura de datos similar que muestre las relaciones entre las palabras contenidas en el vocabulario de los documentos.

Para concluir, se considera posible la aplicación del *software* desarrollado sobre otros *datasets*, tal vez con alguna mínima modificación, por lo tanto, sería enriquecedor tratar otros datos con este mismo *software* para ver su comportamiento y rendimiento.

Referencias

[1] *Algoritmo de similitud de coseno*. URL: https://www.grapheverywhere.com/algoritmo-de-similitud-de-coseno/#:~:text=La%20similitud%20del%20coseno%20es,(0%2C%20%CF%80%5D%20radianes..

[2] Francesco Barbieri, Luis Espinosa Anke y Jose Camacho-Collados. "XLM-T: Multilingual Language Models in Twitter for Sentiment Analysis and Beyond". En: *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, jun. de 2022, págs. 258-266. URL: https://aclanthology.org/2022.lrec-1.27.

[3] *DBSCAN*. URL: https://es.wikipedia.org/wiki/DBSCAN#/media/Archivo:DBSCAN-Illustration.svg.

[4] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[5] *Doc2vec paragraph embeddings*. URL: https://radimrehurek.com/gensim/models/doc2vec.html.

[6] Gustavo Espíndola. *¿Qué son los embeddings y cómo se utilizan en la inteligencia artificial con python?* URL: https://gustavo-espindola.medium.com/qu%C3%A9-son-los-embeddings-y-c%C3%B3mo-se-utilizan-en-la-inteligencia-artificial-con-python-45b751ed86a5#:~:text=Los%20embeddings%20son%20una%20t%C3%A9cnica,lenguaje%20de%20manera%20m%C3%A1s%20efectiva..

[7] Martin Ester et al. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". En: *AAAI* (1996), págs. 226-231. URL: https://cdn.aaai.org/KDD/1996/KDD96-037.pdf.

[8] Lino Alberto Urdaneta Fernández. *Reducir el número de palabras de un texto: lematización y radicalización (stemming) con Python*. 4 de mayo de 2019. URL: https://medium.com/qu4nt/reducir-el-n%C3%BAmero-de-palabras-de-un-texto-lematizaci%C3%B3n-y-radicalizaci%C3%B3n-stemming-con-python-965bfd0c69fa.

[9] Piotr Indyk y Rajeev Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality". En: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 1998, págs. 604-613.

[10] Pratyush Khare. *Deep Learning for NLP: Word2Vec, Doc2Vec, and Top2Vec Demystified*. 1 de abr. de 2023. URL: https://medium.com/mlearning-ai/deep-learning-for-nlp-word2vec-doc2vec-and-top2vec-demystified-3842b4fad5c9.

[11] NIKHILESWAR KOMATI. *Suicide and Depression Detection*. URL: https://www.kaggle.com/datasets/nikhileswarkomati/suicide-watch.

[12] Quoc Le y Tomas Mikolov. "Distributed representations of sentences and documents". En: *International conference on machine learning*. PMLR. 2014, págs. 1188-1196. URL: https://proceedings.mlr.press/v32/le14.pdf.

[13] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[14] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". En: *Advances in neural information processing systems* 26 (2013). URL: https://arxiv.org/pdf/1310.4546.pdf.

[15] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". En: *arXiv preprint arXiv:1301.3781* (2013). URL: https://arxiv.org/pdf/1301.3781.pdf.

[16] Mohantysandip. *A Step by Step approach to Solve DBSCAN Algorithms by tuning its hyper parameters*. URL: https://medium.com/@mohantysandip/a-step-by-step-approach-to-solve-dbscan-algorithms-by-tuning-its-hyper-parameters-93e693a91289.

[17] Kamil Mysiak. *Explaining DBSCAN Clustering*. URL: https://towardsdatascience.com/explaining-dbscan-clustering-18eaf5c83b31.

[18] Manojit Nandi. *Density-Based Clustering*. 2 de dic. de 2020. URL: https://domino.ai/blog/topology-and-density-based-clustering. (accessed: 16.10.2023).

[19] HASSAN SAYED RAMADAN et al. "A HEURISTIC NOVEL APPROACH FOR DETERMINATION OF OPTIMAL EPSILON FOR DBSCAN CLUSTERING ALGORITHM". En: *Journal of Theoretical and Applied Information Technology* 100.7 (2022).

[20] ramyarshet123. *Doc2Vec in NLP*. URL: https://www.geeksforgeeks.org/doc2vec-in-nlp/.

[21] Jörg Sander et al. "Density-based clustering in spatial databases: The algorithm gdbscan and its applications". En: *Data mining and knowledge discovery* 2 (1998), págs. 169-194.

[22] Arjun Sarkar. *All you need to know about 'Attention' and 'Transformers' — In-depth Understanding — Part 1*. URL: https://towardsdatascience.com/all-you-need-to-know-about-attention-and-transformers-in-depth-understanding-part-1-552f0b41d021.

[23] *SpaCy models*. URL: https://spacy.io/models/en.

[24] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].