MINERÍA DE DATOS

PROYECTO: CLUSTERING

17 de noviembre de 2019

Zuhaitz Martínez Jiménez Alex Ye Xu Alain Miranda López

Índice

1. Introducción **3**

2. Descripción y análisis de datos **3**

2.1. Pre-proceso 4

2.2. Problemas abordados y decisiones 5

2.3. Errores analizados 6

3. Algoritmo **10**

3.1. Problemas abordados y decisiones 11

4. Diseño **13**

5. Evaluación **16**

5.1. Índices de calidad interna 16

5.2. Índices de calidad externa 17

6. Planes futuros y opciones de mejora **18**

7. Conclusiones **19**

1. INTRODUCCIÓN

El objetivo de este proyecto es trabajar en profundidad el área de la minería de datos, agrupando datos en *clusters*, con la finalidad de analizar y detectar similitudes entre ellos. En este caso, lo que procesaremos serán artículos de noticias sobre temas variados, para por ejemplo encontrar o sugerir artículos similares, detectar artículos idénticos, agrupar temas... De esta forma, cuando se añadan nuevas noticias, veremos donde serán agrupadas y así podremos descubrir noticias que hablen de temas similares.

Para realizar esta tarea se hará uso de la clasificación no-supervisada que se encarga de hacer agrupaciones naturales. Por lo tanto no se utiliza ningún tipo de clase, ya que se trata de una técnica descriptiva.

Dispondremos de 20.000 instancias los cuales se trataran de noticias, para que de alguna manera podamos ver relaciones entre ellas y así poder utilizarlo más tarde como un sistema de recomendaciones.

Usaremos el *Clustering jerárquico* para agrupar las instancias, lo que puede ser interesante ya que al usar noticias, podremos ver que tipo de emparejamientos irá haciendo y ver de esta manera como de tarde se pueden llegar a juntar las instancias.

Debido a los problemas surgidos en el procesamiento de datos, no se ha podido ejecutar el proceso de *clustering* con datos reales, y han surgido algunos retraso en el proyecto. A lo largo de este documento se hablara de las decisiones tomadas y de la resolución de los problemas surgidos.

2. DESCRIPCIÓN Y ANÁLISIS DE DATOS

Los datos que se van a analizar provienen de *"UCI Machine learning repository"*. Estos documentos fueron reunidos e indexados en categorías por el personal de Reuters Ltd. (Sam Dobbins, Mike Topliss, Steve Weinstein) y Carnegie Group, Inc. (Peggy Andersen, Monica Cellio, Phil Hayes, Laura Knecht, Irene Nirenburg) en 1987

En 1990 fueron publicados para uso de investigación y avance de los departamentos de información y computación. Este paquete de documentos consisten en una colección de 22 ficheros de datos y un fichero *SGML* (Standard Generalized Markup Language) *DTD*

(Document type definition) que describen el formato de los datos y 6 ficheros que contienen las categorías utilizadas para crear el índice.

A pesar de que ya contiene etiquetas la clase no se puede utilizar como atributo para el *clustering* y cada documento puede tener múltiples "clases" asociadas, las cuales pueden ser usadas como evaluación externa. Por ejemplo, un *cluster* de las etiquetas de cada artículo con otro de las palabras de los documentos procesados.

2.1. Pre-proceso

Los documentos extraídos de la web se dice que están en un formato *Raw*, de manera que no están listos para ser analizados. Por lo tanto, lo primero que haremos será adecuarlo a las herramientas que vayamos a utilizar en nuestro caso como *Python* admite todo tipo de texto plano, introduciremos los archivos *SGML* directamente y lo separaremos por artículo para procesar.

Una vez separados los artículos se han filtrado para extraer la información más relevante o más interesante para nuestro uso. Para ello se ha seguido este procedimiento:

Se han cargado todos los textos de la carpeta seleccionada, por defecto 'datos', y se ha asegurado que el idioma de los artículos era Inglés. Seguidamente se ha asegurado *UTF-8* como *encoding* para no obtener caracteres raros. A continuación, se ha *"tokenizado"* el texto plano y separado por palabra. Todo el texto se ha guardado en minúsculas y se ha procedido a quitar los espacios, números y puntuaciones.

Con la ayuda de la librería *NLTK (Natural Language Toolkit)* se han quitado todos las palabras que no aportan información llamadas *stopwords*, las cuales engloban conjunciones, artículos... Asimismo, cuando se añaden las palabras a la lista se lematiza y se comprueba que la palabra no está flexionada, es decir, que no es un plural, femenino o un verbo conjugado y a su vez se busca la raíz de la palabra para englobar más significados y obtener menos tokens para el aceleramiento del proceso del *clustering*.

2.2. Problemas abordados y decisiones

Durante este apartado han surgido múltiples problemas, principalmente problemas con el espacio de memoria. Desde un principio ya se había pensado que esto iba a dar lugar a muchas complicaciones y así ha sido. Se ha conseguido guardar la mayoría de archivos con una herramienta llamada *pickle* en memoria secundaria (disco duro). Y se han usado estructuras básicas como los *arrays*, en vez de estructuras de paginado como los diccionarios.

No obstante nuestro mayor obstáculo, que seguimos sin haber resuelto, ha sido el de cambiar la matriz dispersa en una lista de tuplas, ya que, ni las propias librerías de *scikit-learn*, ni *numpy*, que están diseñadas para grandes cantidades de información, consiguen transformar esta matriz en *arrays* o listas. Todas las ejecuciones con todos los datos han dado lugar a *memory error*, el error que se presenta a continuación.

2.3. Errores analizados

Aquí mostraremos algunos de los errores que han causado que no podamos seguir avanzando correctamente:

```
Espacio vectorial analizado y valores tf_idf calculados

Traceback (most recent call last):

  File "e:/Media/Documents/GIT/clustering/Clustering/main.py",
     line 94, in <module> runClusteringPruebas(args)
  File "e:/Media/Documents/GIT/clustering/Clustering/main.py",
     line 16, in runClusteringPruebas tfidf_vecs, documentos =
     preproceso.preprocesar_train(argumentos.preproceso)
  File "e:\Media\Documents\GIT\clustering\Clustering\preproceso.py
     ", line 302, in preprocesar_train    tfidf.
     generar_vector_tupla_pesos(train)
  File "e:\Media\Documents\GIT\clustering\Clustering\preproceso.py
     ", line 157, in generar_vector_tupla_pesos array_pesos =
     pesos.toarray()
  File "C:\ProgramData\Anaconda3\lib\site-packages\scipy\sparse\
     compressed.py", line 1024, in toarray out = self.
     _process_toarray_args(order, out)
  File "C:\ProgramData\Anaconda3\lib\site-packages\scipy\sparse\
     base.py", line 1186, in _process_toarray_args
  return np.zeros(self.shape, dtype=self.dtype, order=order)
MemoryError
```

El análisis de los resultados se ha visto dificultado por este error el cual indica que la memoria interna de la computadora no es suficiente para guardar las instancias. Esto se da porque la librería de *sklearn* usa como estructura una matriz dispersa *(sparse matrix)*, el cual omite los ceros y solo guarda los valores relevantes en dicha posición para ahorrar memoria. Sin embargo nuestro algoritmo como hemos visto en clase analiza un vector de tuplas y el problema surge a partir del relleno de atributos vacíos que no aportan nada a la tupla pero sin embargo se requieren para calcular las distancias. Se ha intentado varias soluciones pero ninguna ha solucionado el problema ya que la estructura final sigue siendo algo alternativo

a la matriz dispersa con lo cual la propia librería de *scipy* no consigue pasar dicha matriz a *array* como podemos ver en el ejemplo anterior. Donde nuestro código llama a *toarray()* y el proceso falla.

Al principio fallaba al pasar los atributos y sus valores *tf-idf* de la estructura(matriz) a la nuestra(vector de tuplas), por lo que se ha cambiado el código para utilizar la estructura diccionario de *Python* y hacer una tabla de arrays (tabla[][]). Y después de reducir el número de atributos se ha vuelto a cambiar dicha matriz (tabla) a la estructura que utilizamos en el algoritmo, vector de tuplas.

```
2: Clustering
Se ha cargado los vectores tf-idf, del directorio: /preproceso/
   train_tfidf
Traceback (most recent call last):
  File "e:/Media/Documents/GIT/clustering/Clustering/main.py",
     line 94, in <module>
    runClusteringPruebas(args)
  File "e:/Media/Documents/GIT/clustering/Clustering/main.py",
     line 25, in runClusteringPruebas
    cl.clustering(argumentos.distancia)
  File "e:\Media\Documents\GIT\clustering\Clustering\clustering.py
     ", line 46, in clustering
    centroides = self.inicializarCentroides()
  File "e:\Media\Documents\GIT\clustering\Clustering\clustering.py
     ", line 94, in inicializarCentroides
    centroide = ut.calcularCentro(self.clust[i], self.vect)
  File "e:\Media\Documents\GIT\clustering\Clustering\util.py",
     line 22, in calcularCentro
    x+=vector[i]
IndexError: tuple index out of range
```

Este error ocurre después de haber intentado cambiar nuestra estructura para que no haya error como el anterior de *memory error*. Lo que se ha hecho es añadir una excepción para que siga el transcurso del cálculo, sin embargo, sabemos que no se debería pero no estamos seguros de qué es el causante, ya que la estructura es la misma pero con menos atributos. Este error era a causa de que los atributos que a los que se había rebajado el dataset

se duplicaba al añadirse dos veces, sin embargo, no debería importar ya que la tupla debería ser idéntica. Después de reflexionar sobre esto nos dimos cuenta de que era porque uno de los loops estaba hecho 'hard-coded' utilizando un *while* y la función *len* que da la longitud por ello al ser el doble se salía del número de tuplas.

```
(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.5578954100282402, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0), (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.051575046743429796, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.07076482035373939, 0.0, 0.0, 0.0774121964242252, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.05390683409459972, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0)]
centroides es de tipo:<class 'list'>
centroides[i] es de tipo:<class 'int'>
Cuando centroides[i] debia ser una tupla y esta parte del codigo
```

```
   no ha cambiado
Traceback (most recent call last):
  File "e:/Media/Documents/GIT/clustering/Clustering/main.py",
     line 94, in <module>
    runClusteringPruebas(args)
  File "e:/Media/Documents/GIT/clustering/Clustering/main.py",
     line 25, in runClusteringPruebas
    cl.clustering(argumentos.distancia)
  File "e:\Media\Documents\GIT\clustering\Clustering\clustering.py
     ", line 50, in clustering
    distancias.inicializarDist(centroides)
  File "e:\Media\Documents\GIT\clustering\Clustering\distancias.py
     ", line 36, in inicializarDist
    distAct = ut.calcularDistancia(centroides[i],centroides[j],
       self.coeficiente)
  File "e:\Media\Documents\GIT\clustering\Clustering\util.py",
     line 98, in calcularDistancia
    while i<len(centr1):
TypeError: object of type 'int' has no len()
```

En estos momentos el código está en una situación inesperada, los cambios de última hora (nos dimos cuenta de que el *memory error* lo estábamos provocando nosotros transformando la matriz tarde, el código hace lo que queremos y es lo que necesitamos pero no nos habíamos dado cuenta de esto y por ello se ha empleado mucho tiempo en cambiar y rediseñar el código sin pensar que lo que funciona para pocas instancias en realidad es lo que nos estaba haciendo cuello de botella, ya que rellenar un vector con ceros no es lo ideal). El problema reside en que conseguimos pasar una lista de tuplas, la que habíamos diseñado desde el principio, pero python detecta una lista de enteros, sin embargo, al imprimir por pantalla la lista (vector), se ve claramente que es una lista de tuplas y así es como se ha guardado. No obstante, ahora mismo python detecta la lista y coge el primer elemento de la tupla, saltándose la tupla centroides[0], debería ser la primera tupla y centroides[0][0] el primer elemento, pero por extrañas circunstancias no está ocurriendo esto.

3. ALGORITMO

Como se mencionó antes, el algoritmo que se llevará a cabo en el proyecto es el de *Clustering jerárquico*, más concretamente el método aglomerativo (Bottom-up). Consiste en asociar cada instancia en un *cluster* diferente e ir iterando la agrupación de los *clusters* más cercanos entre ellos según su distancia *intergrupal*, cual se tratará en este caso del *Average-link*. Se decidió utilizar esta distancia porque al tratarse de la distancia entre los *centroides* de cada *cluster*, podría reducir bastante el coste de calcular las distancias, ya que cuando vaya juntándose todos los *clusters*, el número de *centroides* irá reduciéndose.

Algoritmo en pseudocódigo:

For all $\{x^t\}_{t=0}^{N}$ in X
  $G^i = \{x^t\}$

`C` <- `G`

For i = 1,...,N-1
  For j = i+1,...,N
    d_{ij}<-d(C_i, C_j)

A la hora hora de iniciar todo el proceso se encargará de inicializar los *centroides* `C` que se resumirá a colocar las instancias en un nuevo *ArrayList*. Ya que los *clusters* solo disponen de una instancia por el momento, el *centroide* será igual que su instancia. Después de esto daremos paso a cargar las matriz de distancias d_{ij} entre los *centroides*, y para evitar cálculos innecesarios se a calculado sólo la mitad de la matriz, porque la diagonal son ceros y la parte inferior de la matriz son simétricas a la parte superior.

Repeat
  x, y <- minimaDistancia(d_{ij})
  $C^x = (C^x + C^y)/2$
  delete C^y from `C`
  update d_{ij}
  combine G^x with G^y

Until $\mathbb{G} = \{x^0, x^1, x^2, ..., x^N\}$

Ya hecho todo este proceso pasamos a la parte iterativa de programa donde iremos llamando a un método que se encargará de buscar la distancia mínima de la matriz, y así conseguir el par de *clusters* que tengan menor distancia entre ellos. Por último, a la hora de actualizar los *centroides* tan solo se sumarán y se hará un a media entre ellos y luego también actualizaremos la matriz de distancias con el nuevo *centroide* que hemos creado y borra uno de los *clusters*, para que no nos ocupe espacio.

3.1. Problemas abordados y decisiones

Uno de los principales problemas que podían surgir con el algoritmo fue lo que podía llegar a tardar en ejecutarse el proceso de *clustering*, porque al tratarse del *Clustering jerárquico*, hay que hacer tantas iteraciones como instancias haya en el conjunto de entrenamiento, esto provoca que el coste se alce mucho si el proceso en cada iteración no está bien optimizado. Esto nos lleva al porqué nos fijamos en no calcular distancias innecesariamente o en actualizar las tablas y *centroides* de una manera más rápida, pero el coste de búsqueda sigue siendo muy alto porque tenemos que ir buscando por la matriz una a una para encontrar el menor. Para ello se pensaron dos posibles soluciones:

- La primera fue utilizar un árbol binario, de manera que fuera repartiendo los costes por las diferentes ramas y así fuera más rápido encontrar el mínimo, ya que el coste de la búsqueda es *logarítmico*, pero hay que tener especial cuidado con que el árbol estuviese balanceado, además de que al desaparecer ciertos *centroides* muchos de las distancias calculadas dejarían de existir, y sería demasiado engorroso actualizarlas o borrarlas
- La otra opción era usar una cola de prioridad, que nos devolvería siempre el valor más pequeño y además se puede actualizar el coste más fácilmente, pero seguiría existiendo el problema de los *centroides* que dejan de existir tendrían sus distancias calculadas, ocupando espacio, además de ser muy difícil actualizar los costes.

Otro problema que hubo fue que al principio se utilizaban diccionarios donde se definían los *clusters*, de esta manera se podía asignar un nombre propia a cada uno de estos. Esto podría hacer que se pudiesen distinguir después de que incluso se juntasen dos, ya que se

quedaba el nombre de uno de los dos, y así hacer que la tarea de actualizar las distancias fuera más fácil. Pero luego las *keys* las pasamos a números enteros porque nos eran útiles para implementar una parte de una funcionalidad, y al ser números enteros ya no tenía mucho sentido utilizar un diccionario, porque un *ArrayList* era mucho mejor. Esto implicaba que había que realizar algunos cuantos cambios en el código para poder ajustarse a un *ArrayList*, y esto provocó que la actualización de las distancias en los árboles binarios y colas de prioridad fueran más complicadas, porque los *clusters* dejaban de tener el mismo nombre, así que se decidió usar una matriz.

4. DISEÑO

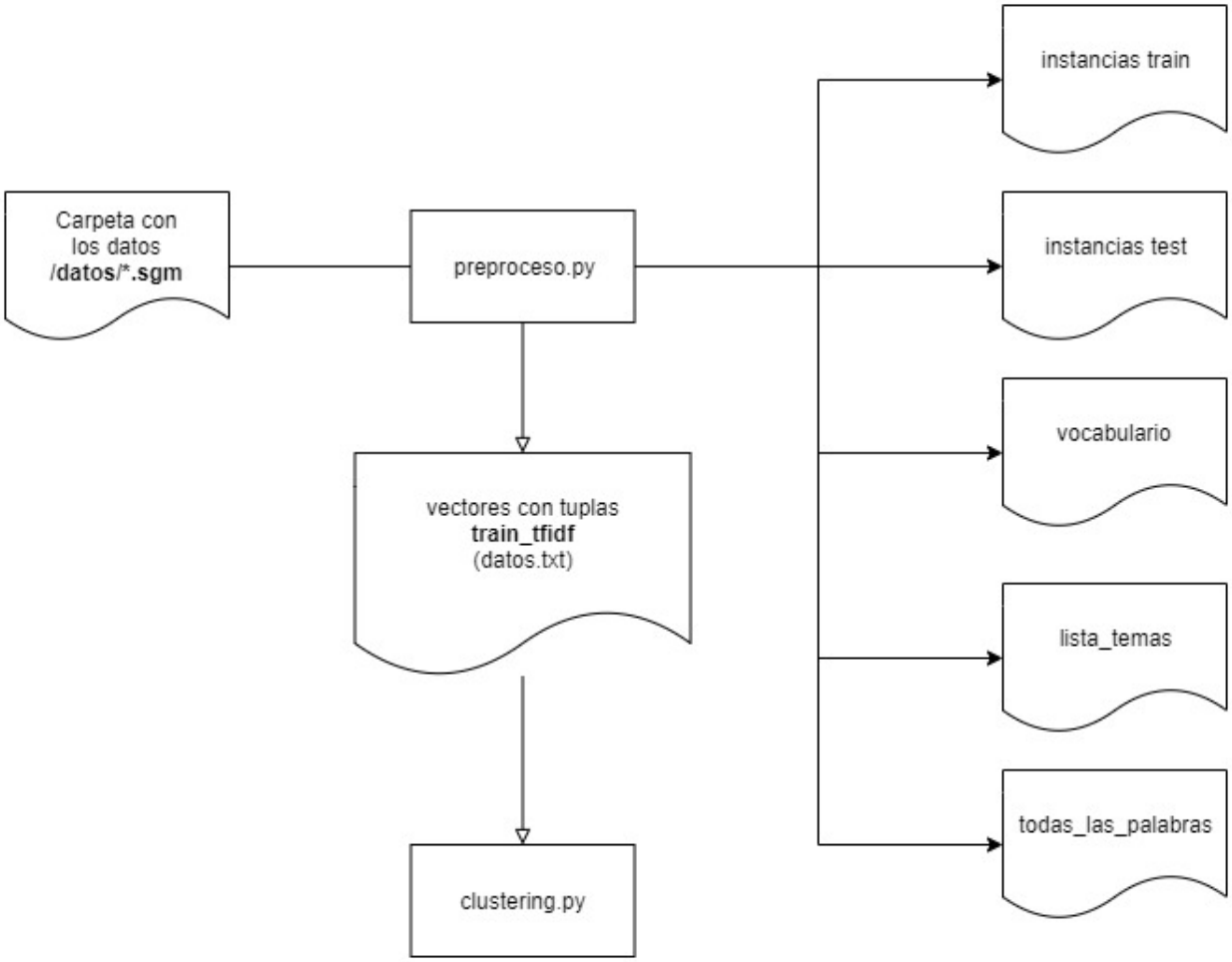

Figura 1: Flujo de ejecución del preproceso

- ***.sgm** Son los datos que se van a analizar, las noticias, aunque el programa filtra todos los archivos que estén en la carpeta datos y que el articulo esté etiquetado con los tags xml <reuters></reuters>.
- **instancias train** Lista de clase datos con las instancias con las que se ha creado el *cluster*.
- **instancias test** Lista de clase datos con las instancias que se añadirán al *cluster* y se analizarán más adelante.

- **vocabulario** Es el *dataset* utilizado, en ella están todos los tokens usados en el *cluster*.
- **lista_temas** Son todos los temas que hay en el *dataset* analizado, etiqueta que viene con el artículo que se utilizará como clase para clasificar un *cluster* en el análisis del índice de evaluación *Jaccard*.
- **vocabulario** Es el *dataset* que no se ha utilizado, en ella están todos los tokens que se extraen de los datos *Raw*.
- **train_tfidf** Es el archivo que se utilizará en *clustering* para calcular las distancias.
- **datos.txt** En el caso de que se añadan nuevas instancias lo que se generará será un archivo *tfidf* con las nuevas instancias (Este se llamará como desee el usuario no se llamará *train_tfidf*).

Proceso de clustering

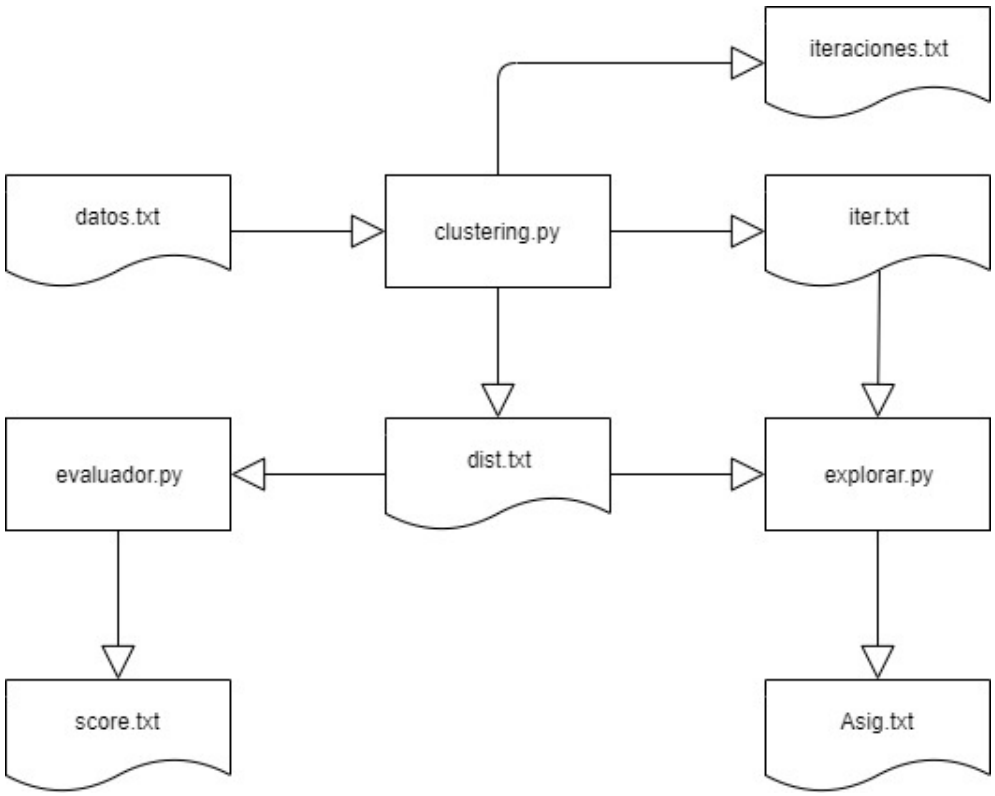

Figura 2: Flujo de ejecución del *clustering*

- **datos.txt**: Es donde guardamos la estructura de nuestro *dataset*.

- **iteraciones.txt**: Está compuesto por los *clusters* en cada iteración, su distancia y el estado de los *clusters* en esta.
- **iter.txt**: Está compuesto por los *clusters* en cada iteración, lo cual nos puede servir para sacar mas rápido el número de agrupaciones que se quiere utilizar.
- **dist.txt**: Es donde guardamos la estructura de todas las distancias calculadas, con sus respectivos *clusters*, para acceder cuando sea necesario.
- **score.txt**: Tras realizar las evaluaciones se guardarán en este documento los resultados obtenidos.
- **Asig.txt**: Te muestra el *cluster* al que pertenece cada instancia, dado el número de *clusters*.

5. EVALUACIÓN

Para la evaluación de nuestro proyecto se harán uso tanto de índices de calidad internos como externos. Estos se encargan de validar la cohesión interna y externa de nuestro *clustering*, y darnos a conocer que nivel del jerárquico es el más adecuado para agrupar las instancia nuevas.

5.1. Índices de calidad interna

Para la evaluación se ha decidido utilizar como índice de calidad interna el *Davies Bouldin score*, el cual define su *score* la distancia media en el *cluster* y a los demás *clusters*. Por lo tanto los *clusters* que se alejen más de los demás y no sean muy dispersos recibirán mejor valoración.

Davies Bouldin score

Siendo s_i y s_j la distancia media de cada *instancia* del *cluster* a su *centroide* y $d_i j$ la distancia entre los *centroides* de los *clusters* i y j. De esta manera podremos calcular la similitud entre *clusters* (1). [1]

$$R_{ij} = \frac{s_i + s_j}{d_{ij}} \tag{1}$$

Calculando la fórmula 1 para cada par de *clusters*, calculará un promedio entre el numero de *clusters k*, de esta manera sacaremos el índice *Davies Bouldin*. Cuanto más se acerque este índice a 0 mejor será el *clustering*. [1]

$$DB = \frac{1}{k}\sum_{i=1}^{k} \underset{i\neq j}{max} R_{ij} \tag{2}$$

Así pues, elegimos este índice de evaluación porque, en comparación a otros índices, su coste computacional es menor y aunque su valoración es menos precisa, sigue siendo bastante buena ya que buscábamos un índice que fuese rápido pero diese un buen resultado.

Otros índices de calidad

- Silhouette Coefficient: Por su amplio coste computacional se decidió descartar este coeficiente.
- Calinski Harabasz Score: Al principio se consideró usar este índice para evaluar el *clustering*, ya que el coste computacional a la hora de calcularlo se muy bajo, y además devolverá un mejor valor si los *clusters* están bien separados entre ellos y su densidad es alta. Pero los valores que nos devuelve este índice no están acotados, o se que nunca sabemos si el *clustering* es bueno o no porque solo devuelve valores altos.

5.2. Índices de calidad externa

Los índices de calidad externa comparan los resultados entre los datos resultantes del *clustering* y un conjunto de datos etiquetado. También es conocido como *cluster vs class evaluation*.

Nuestros datos al no contar con una clase predefinida no era posible medir los índices de calidad externa. Para ello, aprovechando que las noticias contaban con un tema asignado lo hemos utilizado como nuestra clase de manera que podamos medir los índices externos.

Jaccard index

El índice *Jaccard* se utiliza para medir la similitud entre dos *datasets*. Tomando un valor entre el 0 y 1, significando 0 no tener elementos en común y el 1 una completa similitud entre los *datasets*.

Se calcula por lo tanto mediante la intersección de los *datasets* entre su unión. (3).

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{TP}{TP + FP + FN} \tag{3}$$

6. PLANES FUTUROS Y OPCIONES DE MEJORA

Al no haber logrado unos resultados reales no hemos llegado a plasmar los *clusters* en un gráfico o construir el *dendograma* que obtendríamos tras nuestro *clustering jerárquico*. Esto habría podido ayudar a visualizar bien los emparejamientos y distinguir como diferentes noticias tardarían mucho en juntarse en el mismo *cluster*.

Quedan cosas por mejorar, entre las cosas que más nos gustaría implementar con tiempo, sería el análisis completo de todas las instancias dadas y una mejor coordinación en la ejecución del código. Cabe destacar, que las opciones del menú son ampliable y el fallo más destacable es de memoria. Se quería abarcar muchas cosas y al final nos hemos visto muy limitados por la falta de distribución del tiempo de forma eficiente y principalmente por este fallo pensábamos que era de crear demasiadas clases o estructuras, y aunque no se aleja mucho, no era el principal problema. Además, no debería ocupar demasiado un *array* de tuplas, ya que, según nuestros hallazgos en la web es lo que menos ocupa y aún así sigue dando errores.

Así mismo, nos habría gustado comparar lo teórico con lo práctico, sin embargo, la falta de resultados prácticos por errores de ejecución no nos ha permitido plasmar de forma concisa en tablas cómo debería comportarse el *cluster* jerárquico que hemos generado. En un principio teníamos pensado probar con *bag of words, tf-idf y word2vec*, sin embargo, finalmente nos decantamos por *word2vec*, además nos habría gustado ver qué diferencias obtendríamos con las diferentes fórmulas para las distancias y finalmente ejecutar diferentes tipos de evaluación interna y externa a expensas de el tiempo que lleguen a tardar.

7. CONCLUSIONES

Aunque no hemos podido hacer pruebas y conseguir resultados reales, hemos intentado diseñar bien el código y asegurarnos de que se entendiese bien, tanto comentando el código además de explicando las decisiones tomadas al respecto como haciéndolo modular, de manera que se pudiese reutilizar los métodos en diferentes partes del código. También se realizaron pruebas con datos generados aleatoriamente, con objetivo de probar cuánto tardaba en hacer el *clustering*, y para asegurarnos de presentar algo que funciona.

Como se mencionó anteriormente, puede que le falte un poco de optimización, sobre todo en la parte de buscar el par de *clusters* con la distancia mínima. Pero como se explicó antes no se encontró muchas opciones que fuesen fáciles de implementar y no llevasen mucho tiempo. Aproximadamente, con 1000 instancias con 800 atributos, los cuales estaban generados aleatoriamente, tardaba unos 5 minutos en realizar el *clustering*, en los cuales aproximadamente 2 minutos se dedicaban a calcular la matriz de distancias.

Puede que fuera una equivocación haber realizado este proyecto con datos totalmente nuevos que desconocemos, además de estar en un formato diferente, pero queríamos hacer algo interesante con las palabras. Queríamos haber podido trastear con las diferentes posibilidades y ver que tipo de resultados nos daría, además de poder ver que tipo de agrupamientos haría el *clustering*.

REFERENCIAS

[1] *Scikit Learn Davies Bouldin Score*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.davies_bouldin_score.html#sklearn.metrics.davies_bouldin_score.