Algoritmo de clustering jerárquico

Minería de Datos

Bosco Aranguren, Joel Bra, Borja Pecharroman
Alicia Pérez Ramírez

1 de noviembre de 2022

Índice

Índice de figuras

1. Introducción

1.1. La tarea

La tarea propuesta consiste en diseñar e implementar un algoritmo de clustering jerárquico. De forma resumida, el clustering jerárquico comienza considerando cada instancia como un cluster independiente. Por cada iteración une los dos clusters que comparten mayor similitud entre sus atributos y genera un nuevo cluster con la media de los valores de sus atributos.

Estas iteraciones continúan hasta llegar a k clusters. Parte de nuestra tarea consiste en obtener un valor k que nos genere un conjunto de clusters del que se puedan obtener conclusiones útiles. Además, se nos pide representar los resultados obtenidos de forma visual y ser capaces de añadir una nueva instancia a nuestro conjunto de clusters.

1.2. Retos

A primera vista el mayor reto que presenta este proyecto es desarrollar el algoritmo. Si bien es cierto que no es una tarea fácil, existe mucha información sobre el algoritmo y no contiene procesos matemáticos complicados. Sin embargo, lograr que la implementación sea eficiente puede llegar a ser complejo.

Un reto mayor puede ser como adaptar los textos a un modelo numérico que el algoritmo pueda utilizar. Como se explica más adelante nuestra idea inicial fue usar TF-IDF pero el coste computacional era demasiado alto, lo que nos hizo cambiar a LDA.

En cuanto a añadir una nueva instancia consideramos que el reto es más adaptarla al modelo numérico del LDA ya generado, que introducirla como nuevo cluster

ya que podemos reutilizar funciones y hemos buscado formas de almacenar los resultados de forma que se pueda continuar trabajando con ellos una vez ejecutado el algoritmo de clustering.

1.3. Antecedentes

Teniendo en cuenta los retos explicados en el apartado anterior, a continuación veremos como otros actores han abordado estos problemas de manera satisfactoria.

En cuanto a la implementación del algoritmo, existen diferentes librerías externas en python que lo realizan de manera rápida y eficiente. Sin embargo, y tal y como se nos presenta la tarea, es obligatorio que realicemos nuestra implementación aunque podríamos extraer ideas ya que el código es visible en GitHub en librerías como Scipy. Por otro lado, hemos recibido información suficiente en clase para crear un algoritmo matemático sencillo que nos permita abordar el problema sin mucha dificultad mediante matrices con distancias.

Al igual que el algoritmo de clústering jerárquico, el preproceso también se puede realizar con librerías ya implementadas. Sin embargo, podremos partir de implementaciones previas llevadas a cabo en otros trabajos o volver a programar el preprocesado. Además, y tal y como hemos comentado, se nos presentan diferentes opciones para preparar los datos para el clústering:

- Tf-Idf: Es una técnica de preprocesado sencilla que expresa la importancia de una palabra para un documento en una colección.
- Latent Dirichlet Allocation (LDA): es un modelo generativo que pretende otorgar un tópico a cada instancia.

En ambos casos se ha propuesto la aplicación del análisis de componentes prin-

cipales (PCA) para reducir el número de atributos de las instancias y tratar de no perder información.

Por otro lado, librerías externas también permiten añadir nuevas instancias a sus modelos ya implementados. Por tanto, podemos basarnos en ellas y no dejaremos de aplicar el preprocesado previo.

1.4. Propuesta

Para la realización del proyecto hemos pensado en basarnos en los apuntes dados en clase de la asignatura de Minería de Datos ya que su implementación sencilla mediante distancias entre instancias no debería ser complicada de implementar.

En cuanto al preprocesamiento, en un principio elegimos Tf-Idf por su sencilla implementación; sin embargo, decidimos decantarnos por LDA debido a la reducción de atributos que nos otorga.

Finalmente, para introducir una nueva instancia, aplicaremos el preprocesado y lo añadiremos al modelo LDA que hemos generado. Una vez realizado esto, mediante las funciones que implementaremos del algoritmo, lograremos añadir la instancia a un clúster e indicar cuales son las dos instancias más cercanas en esa agrupación.

2. Análisis de datos

2.1. Descripción cualitativa y fuente de los datos raw

Los datos que hemos usado para el proyecto han sido extraídos de una competición de kaggle sobre fake news, el objetivo de la competición es construir un

sistema que clasifique las noticias como verdaderas o falsas.

Los datos están en formato csv y están formados por un train de 20.800 instancias y un test de 5.200, nosotros hemos usado solamente el test ya que lo considerábamos de tamaño suficiente para nuestro proyecto teniendo en cuenta los problemas de optimización del algoritmo. Este test esta compuesto por un identificador, el titulo de la noticia, el autor y el texto de la noticia.

Ya que nuestro trabajo consistía en programar un algoritmo de clustering jerárquico y no en diferenciar entre noticias verdaderas y falsas hemos optado por deshacernos de las columnas de título y autor y centrarnos en el texto de la noticia. El id se ha mantenido como identificador de las instancias pero no afecta a los resultados del algoritmo.

2.2. Pre-procesado

En cuanto al preprocesado hemos tenido que tratar el texto para hacerlo homogéneo y poder trabajar con él. Para empezar hemos pasado todas las letras a minúsculas, eliminado signos de puntuación, stopwords y caracteres no pertenecientes al abecedario. Tras ello hemos usado la librería nltk para lematizar y obtener la raíz de las palabras. De esta forma evitamos que palabras con el mismo origen pero distinto morfema sean consideradas como distintas, por ejemplo: gato y gata, gat; jugamos y jugasteis, juga...

Tras ello hemos aplicado LDA sobre estos datos de forma que obtenemos para cada instancia el porcentaje de pertenencia a cada tópico. En un primer momento probamos a usar TF-IDF pero eso suponía que cada palabra distinta que apareciera en el conjunto de textos sería un nuevo atributo, creando aproximadamente 50.000 atributos por instancia. Esto era muy costos computacionalmente. Tan solo usando 300 instancias el tiempo de ejecución era de 50 minutos. Calculando una función de coste aproximada y sabiendo el tiempo de ejecución para 300 instan-

cias estimamos que para las 5.200 instancias tardaría algo más de un año por lo que lo descartamos automáticamente y nos decantamos por el LDA.

El LDA (Latent Dirichlet Allocation) es un modelo probabilístico generativo que se usa principalmente para colecciones de textos. LDA es un modelo Bayesiano de tres niveles. Todos los niveles son relaciones entre documentos (los textos), palabras y tópicos. Para empezar la relación más obvia es que cada documento esta compuesto por un conjunto de palabras, esta relación por si sola no nos aporta nada nuevo pero sirve de nexo entre los tópicos y los documentos. Se considera que cada documento esta formado por todas las palabras de los documentos pero con un índice que determina su importancia dentro del tópico. Si este índice va desde 0 a 1, habrá palabra que no tengan relación con el tópico y por lo tanto un índice cercano o igual a 0, mientras que por otra parte palabras muy representativas del tópico tendrán un índice cercano a 1. Uniendo esta relación con la primera obtenemos la tercera relación que es la que más nos interesa. Cada documento tiene cierta probabilidad de probabilidad de pertenecer a cada tópico en función de las palabras que lo componen. Gracias a ello obtenemos por cada documento (instancia) una lista de valores numéricos (atributos) que podemos utilizar en el algoritmo.

2.3. Descripción cuantitativa de los datos pre-procesados

Lo que hacemos con proceso de LDA es un soft clustering (cada instancia puede pertenecer a varios clusters). Después utilizamos los resultados como atributos para nuestro algoritmo de clustering jerárquico. Con LDA podemos establecer el número de tópicos, y mediante pruebas y métricas podemos estimar cual es el mejor número de tópicos (atributos) para nuestro algoritmo.

Tras aplicar LDA obtenemos una matriz de alrededor de 20.000 instancias y 200 atributos. Esta compuesta por valores entre 0 y 1.

3. Clustering

3.1. Algoritmo

```
cargar_datos(cvs)
para cada instancia (i1):
    para cada instancia siguiente (i2):
        distancia = calcular_distancia(i1, i2)
        L_distancias_ordenadas.add(i1, i2, distancia)
mientras numero_clusters > k:
    nuevo_cluster = L_distancias_ordenadas[0]
    L_distancias_ordenadas.eliminar(new_cluster)
    new_centroide = average(new_cluster.i1, new_cluster.i2)
    para cada cluster c:
        distancia = calcular_distancia(c, new_cluster)
        L_distancias_ordenadas.add(c, new_cluster, distancia)
```

3.2. Métricas

Para el proyecto hemos usado métricas internas. Como se detalla en el apartado 4.1 hemos usado cohesión intracluster, disimilitud externa y separabilidad externa. Con estas métricas hemos obtenido información que nos ha ayudado a entender la distribución de los temas de las instancias. Adelantando a lo que se explica en futuros apartados la mayoría de instancias están agrupadas en un tema general y unas pocas se encuentra más dispersas con otro temas.

La cohesión intracluster de cada cluster nos muestra por lo tanto que los pequeños clusters tienen una gran cohesión y el cluster ”grande”mantiene una cohesión alta aunque menor que las de los otros clusters.

La disimilitud es alta en todos los clusters al estar la gran mayoría separados y

repartidos. De la separbilidad no hemos podido obtener información relevante.

3.3. Almacenamiento del modelo

Guardar el modelo es fundamental para su utilización en caso de que queramos añadir nuevas instancias. Para realizar esto, guardaremos el dataframe generado que nos da nuestro algoritmo de clustering jerárquico y que está formado por lo siguiente:

- Cluster: se trata de una cadena de texto que determina si es un cluster con varias instancias (c-k) o con una única instancia (c-i).
- Instancias: es una lista compuesta por los números que identifican a los documentos en la colección original.
- Centroide: es una lista de valores que forman el centroide del cluster.

Cluster	Instancias	Centroide
i-3.0	[3.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0311207414071978
i-13.0	[13.0]	[0.0, 0.0, 0.0, 0.0489236755543272, 0.0, 0.0308
c-16	[17, 1, 9, 16, 4, 15, 8, 18, 20, 6, 2, 19, 12, 5, 14, 7, 10, 11]	[0.009768263750846223, 0.000105691827593(

Figura 1: Ejemplo extraído del archivo .csv del modelo

Además, para realizar el preproceso de manera correcta a una nueva instancia, deberemos guardar el vocabulario y el modelo del LDA.

3.4. Asignación de un cluster a una nueva instancia

Como requisito de esta práctica se nos pedía que una vez entrenado el modelo encontrásemos una forma de introducir una nueva instancia a un cluster ya

existente. Además, dentro de ese cluster, debíamos buscar las dos instancias más cercanas a la nueva introducida.

Para llevar a cabo esta tarea, hemos guardado los modelos tal y como se ha explicado en el aparatado anterior. A partir del diccionario y el modelo del LDA hemos logrado el mismo preproceso que realizamos a las instancias ya agrupadas. Después, para buscar el cluster más cercano, hemos utilizado parte del código ya implementado en la creación del modelo. Finalmente, recalculamos el centroide teniendo en cuenta el peso de las demás instancias en la agrupación.

3.5. Número óptimo de clusters

Uno de los problemas mas claros que nos encontramos a la hora de utilizar algún método de clústering es la elección del número de clusters que queremos crear. Si elegimos un número erróneo de grupos puede dar como resultado conjuntos con datos muy variados, en el caso de que escojamos pocos clusters, o agrupaciones diferentes con datos muy parecidos entre sí, en el caso que escojamos demasiados clusters.

Existen diferentes métodos para lograr un número óptimo de clusters, entre ellos destacan:

- Método del codo.
- Criterio de Calinsky.
- Gap.
- Dendogramas.

En nuestro caso utilizaremos el método del codo, ya que no es muy complicada de implementar debido a su sencilla fórmula. En este caso, para cada k (número

de clusters), calcularemos la inercia. Esta medida es la suma de las distancias al cuadrado de cada instancia al cluster de su centroide.

$$\sum_{i=1}^{N}(x_i - C_k)^2$$

Figura 2: Fórmula de la inercia

En la fórmula anterior, x_i es un instancia perteneciente al cluster C_k. N será el número de instancias que tiene la muestra.

El codo ha de encontrarse de manera gráfica, para ello, tomaremos diferentes valores de k y veremos su inercia. Una vez realizado este proceso, obtendremos una gráfica en la que tendremos en el eje x el número de agrupaciones y en el y, la inercia.

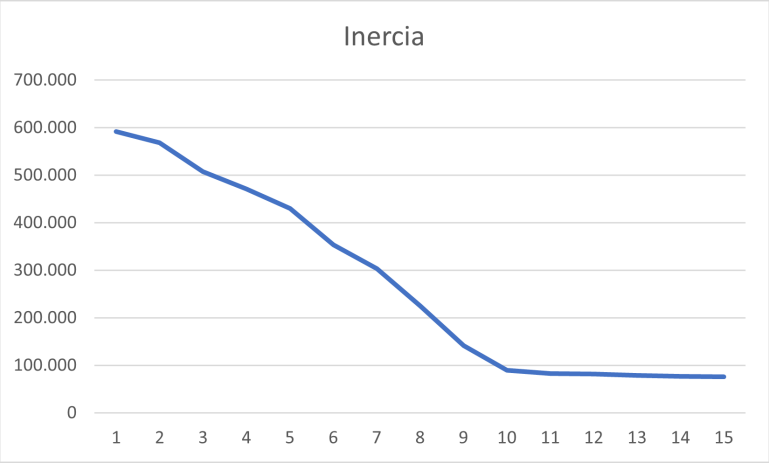

Figura 3: Gráfico de la inercia

Tal y como se puede observar en el gráfico generado que tiene en el eje de abscisas el número de clusters y en el de ordenadas el valor de la inercia, existe un cambio brusco cuando llegamos a 10 agrupaciones. Por ello, vamos a tomar este

valor como el número de clusters óptimo ya que a partir de ahí la distancia de una instancia al cluster se va reduciendo de manera paulatina.

4. Evaluación

4.1. Técnicas de evaluación

Para el algoritmo hemos implementado tres técnicas de evaluación. La primera de ellas es la cohesión interna. Esta medida representa como de relacionados están las instancias de un cluster entre si. Para ello suma los errores cuadráticos de cada instancia a su centroide.

$$SSE(G_i) = \sum_{x \in G_i} d^2(x, c_i) \tag{1}$$

Esto nos da la cohesión interna de cada cluster, para la de la partición tendremos que sumar las de todos los clusters.

$$SSE(G) = \sum_{i=1}^{m} \sum_{x \in G_i} d^2(x, c_i) \tag{2}$$

Cuanto menor sea el valor la cohesión interna, mayor será la cohesión interna.

Otra métrica que hemos usado ha sido la disimilitud externa. Esta métrica mide la separación de los clusters entre si calculando la distancia cuadratica de cada instancia a los centroides a los que no pertenece. Al igual que la anterior calculamos la de cada cluster y la de la partición.

$$extSSE(G_i) = \sum_{x \notin G_i} d^2(x, c_i) \tag{3}$$

$$extSSE(G) = \sum_{i=1}^{m} \sum_{x \notin G_i} d^2(x, c_i) \tag{4}$$

Por último la separibilidad mide la distancia desde cada centroide al centroide del conjunto completo de datos.

$$BSS(G) = \sum_{i=1}^{m} |G_i d^2(c, c_i) \tag{5}$$

5. Resultados experimentales

5.1. Objetivos de los experimentos y conclusiones

Durante las pruebas del algoritmo hemos tenido muchos problemas intentando entender los resultados del proceso. Nuestras dudas surgían porque el algoritmo creaba un gran cluster con prácticamente todas las instancias y dejaba unas pocas instancias sueltas para llegar al numero deseado de clusters. Es decir para k=5 y 1.000 instancias tendríamos un cluster de 996 instancias y 4 instancias sueltas que serían los otros 4 clusters. Tras muchos experimentos tratando de buscar el error descubrimos que realmente no había error y el algoritmo estaba ejecutándose de forma correcta. Lo que sucede es que la gran mayoría de nuestros datos tienen un tema en común y unos pocos textos difieren de este tema común. Esto lo vimos al principio con PCA pero no entendimos porque daba ese resultado y creyendo que había algún error en el PCA lo abandonamos. Como se puede ver en la imagen 4, la gran mayoría de instancias se concentran en la parte baja del gráfico y unas pocas instancias se reparten sueltas por el resto del gráfico.

Para confirmar esta hipótesis probamos a ejecutar el algoritmo con un número de clusters cercano a al número de instancias. De esta forma deberíamos ver muchos

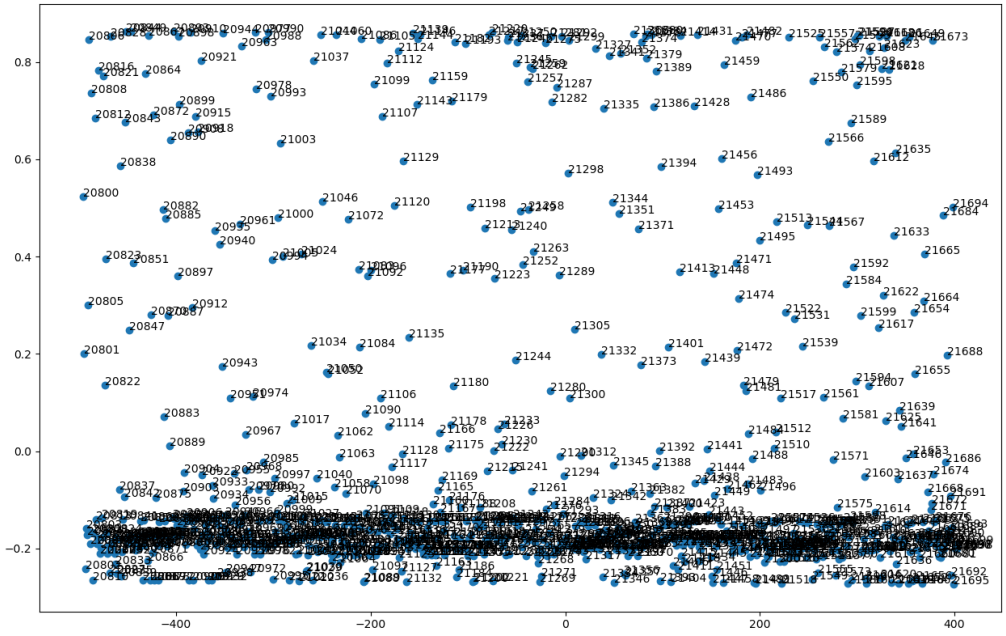

Figura 4: PCA para 900 instancias

clusters de pocas instancias. El resultado fue el esperado y esto nos confirmo las sospechas. Por último hicimos una comprobación menos empírica, entramos en el archivo csv que contenía los datos y revisamos los textos. Como era de esperar todos hablaban de temas similares, principalmente conflictos bélicos modernas de Estados Unidos.

6. Conclusiones y trabajo futuro

La primera conclusión que sacamos es la necesidad de gestionar la eficiencia del código en este tipo de algoritmo. Hemos tratado de evitar operaciones innecesarias pero tampoco podemos decir que haya sido nuestra prioridad. Como trabajo futuro deberíamos optimizar el código y, entre otras cosas, buscar como aprovechar las opciones multihilo para usar todo el potencial de la CPU.

Otra conclusión que sacamos es la importancia de los procesos previos a aplicar el algoritmo. Por un lado, un buen preprocesado puede ahorrar tiempo y mejorar los resultados. Apoyarse en la librería nltk para ello nos ha ayudado ya que la conocíamos de anteriores proyectos y nos ha llevado tiempo adaptarla. Por otro lado, el LDA nos ha solucionado el problema del exceso de atributos del TF-IDF aunque nos haya causado retrasos por confusiones con el número de tópicos que deberíamos usar. Por último la elección de los datos también nos ha afectado en gran medida. Si bien los resultados que hemos obtenido no han sido el mejor ejemplo posible de clustering, varios clusters medianos a distancias significativas, si son unos resultados interesantes que nos han permitido descubrir que había un tema común en los textos.

Como trabajo futuro se podría adaptar el código para poder usarlo con todo tipo de datos. El algoritmo siempre y cuando los datos se introduzcan en el formato correcto funciona para textos en inglés. Sería interesante adaptarlo para imágenes y otro tipo de contenido aparte de texto.

Referencias

[1] Journal of Machine Learning Research 3 (2003) 993-1022. *Latent Dirichlet Allocation.*

[2] Librería Scikit learn *sklearn.decomposition.PCA*

[3] Librería NLTK *nltk.stem.wordnet*

[4] Librería NLTK *stem.porter*

[5] Librería Gensim *models.ldamodel – Latent Dirichlet Allocation*

[6] Medium *Principal Component Analysis(PCA)*

[7] SciPy *scipy.cluster.hierarchy.linkage*

[8] StackAbuse *Definitive Guide to Hierarchical Clustering with Python and Scikit-Learn*

[9] Jarroba *Selección del número óptimo de Clusters*

[10] Medium *Clustering: How to Find Hyperparameters using Inertia*