Inteligencia de Negocio

Práctica 2: Análisis Relacional mediante Segmentación

Autor
Mónica Calzado Granados, Grupo 2
monicacg1111@correo.ugr.es

Escuela Técnica Superior de Ingenierías
Informática y de Telecomunicación

Contents

1 Introducción

1.1 Conjunto de datos

En esta práctica, se aplicarán distintas técnicas de clustering sobre un conjunto de datos de un problema de aprendizaje no supervisado. El problema consiste en analizar una encuesta publicada el lunes 6 de noviembre de 2023 por el CIS, que recoge la intención de voto de la población. A partir de estos datos, se definirán tres casos de estudio distintos para analizar.

Los datos de la encuesta están formados por 2000 respuestas y 59 variables que recogen datos demográficos, sociales, económicos, ideología, religión, opinión sobre temas de actualidad e intención de voto. Podemos encontrar distintos tipos de variables:

- **Variables numéricas,** como la edad (años) o la posición ideológica (escala del 0 al 10).
- **Variables ordinales**, como el nivel de educación o la clase social. Estas variables pueden convertirse a numéricas y tratarse con las medidas de distancia tradicionales.
- **Variables nominales sin orden**, como la provincia y el sexo. Estas variables no sirven para aplicar clustering, pero las emplearemos para definir casos de uso y comparar resultados entre distintos grupos.

En cada caso de estudio definido, se elegirá un subconjunto de estas variables con el fin de obtener la mayor cantidad de información útil o relevante posible.

1.2 Algoritmos

En cada caso de estudio analizaremos 5 algoritmos distintos de clustering. Además, analizaremos el efecto que tienen algunos parámetros (por ejemplo, el valor de k) en 2 algoritmos distintos por cada caso de estudio. Hemos escogido los siguientes algoritmos:

- **K-means:** se trata de un método basado en particionamiento, es decir divide el conjunto de datos en k clusters. El algoritmo k-means necesita como parámetro de entrada el número de clusters deseado. Cada cluster se representa por el centro del cluster. Desde un principio se crean los k clusters deseados de forma aleatoria. Las instancias se van moviendo entre clusters (hacia el cluster más cercano) y se va desplazando el centro de cada cluster hasta que deja de haber movimiento entre clusters.

  Se trata de un algoritmo bastante eficiente y normalmente converge en un óptimo local. Sin embargo, es bastante débil ante datos ruidosos y outliers. Además, genera clusters convexos.

- **BIRCH:** método basado en particionamiento, realiza un clustering incremental, es decir, va agrupando conforme se van recibiendo objetos. Este método almacena las características de los clusters (CF=N, LS, SS) en un árbol para resumir los objetos que van llegando. Cada vez que llega un nuevo objeto, desciende por el árbol y se escoge el CF más cercano. Al llegar a una hoja, se decide si se puede agregar a un CF existente o si hay que crear uno nuevo. El método recibe el número de clusters que se quiere tener. En las ejecuciones se han fijado los parámetros a 15 como factor de ramificación y 0.1 como threshold.

- **Mean Shift:** es también un método basado en particionamiento. En lugar de fijar el número de clusters k, el algoritmo fija un radio y va desplazando los centros de los clusters hacia las regiones más densas. Mean Shift genera clusters convexos y no requiere la especificación previa del número de clusters. Su rendimiento puede depender de la elección del tamaño del radio, el cual se puede estimar con k-NN.

- **DBSCAN:** es un algoritmo basado en la densidad de los puntos en el espacio, por lo que no tiene por qué generar clusters convexos, sino que se pueden encontrar clusters de formas arbitrarias. Además, es robusto frente a ruido y outliers.

  La clave del método reside en los "puntos núcleo", que tienen al menos un número mínimo de vecinos ($minPts$) dentro de un radio especificado (eps). Estos puntos forman clusters alcanzando otros puntos directamente o a través de una cadena de conexiones. Además genera un cluster especial, etiquetado como -1, que contiene los puntos que no han sido agrupados.

  La elección de los parámetros eps y $minPts$ puede afectar su rendimiento.

- **Single Linkage:** el enlace simple es un método de clustering jerárquico aglomerativo que se basa en medir la distancia entre dos clusters utilizando la distancia más corta entre cualquier par de puntos, uno de cada cluster. En otras palabras, la distancia entre dos clusters se define como la distancia mínima entre un punto del primer cluster y un punto del segundo cluster. A continuación se fusionan los dos clusters más cercanos en uno nuevo y el nuevo cluster hereda el enlace simple (la distancia más corta entre los puntos de los clusters originales). Se repite el proceso una y otra vez hasta un criterio de parada (si no, acabaríamos con un único cluster).

  Aunque es computacionalmente eficiente, este algoritmo tiende a formar clusters alargados y es sensible a outliers y a ruido.

Para poder ejecutar los algoritmos he tenido que importar distintos métodos de sklearn:

```
from sklearn.cluster import KMeans, Birch, MeanShift, estimate_bandwidth, DBSCAN,
↪  AgglomerativeClustering
from sklearn import metrics
from sklearn.impute import KNNImputer
from sklearn.manifold import MDS
```

Luego he declarado las siguientes funciones para cada algoritmo:

```
### ALGORITMOS ###
def kmeans(X_normal, subset, n_clusters_arg=4, n_init_arg=5, random_state_arg=123456):
    print('----- Ejecutando k-Means',end='')
    k_means = KMeans(init='k-means++', n_clusters=n_clusters_arg, n_init=n_init_arg,
    ↪  random_state=random_state_arg)
    t = time.time()
    cluster_predict = k_means.fit_predict(X_normal, subset['ponde']) #se usa peso para
    ↪  cada objeto (factor de elevación)
```

```
7       tiempo = time.time() - t
8       print(": {:.2f} segundos, ".format(tiempo), end='')
9       return tiempo, k_means, cluster_predict
10
11  def birch(arg_branching_factor, arg_threshold, X_normal):
12      print('----- Ejecutando Birch, branching factor: ' + str(arg_branching_factor) + ',
        ↪  threshold: ' + str(arg_threshold), end='') # -----
13      #El branching factor (factor de ramificación) controla el número máximo de
        ↪  subclusters que un nodo interno puede tener antes de dividirse en dos.
14      #El threshold es un umbral que controla la compactación de los subclusters en el
        ↪  proceso de construcción del árbol.
15
16      #Medimos el tiempo
17      t = time.time()
18      # Ejecutamos el algoritmo, asignamos el num de clasters
19      birch = Birch(branching_factor=arg_branching_factor, threshold=arg_threshold,
        ↪  n_clusters=4)
20
21      cluster_predict = birch.fit_predict(X_normal)
22      tiempo = time.time() - t
23      #Mostramos resultados
24      print(": {:.2f} segundos, ".format(tiempo), end='')
25
26      return tiempo, birch, cluster_predict
27
28
29  def meanshift(arg_bandwidth, X_normal):
30      print('----- Ejecutando meanshift con radio=' + str(arg_bandwidth), end='')
31
32      #Tomamos tiempos
33      t = time.time()
34      # Ejecuto el algoritmo y asigno los clusters
35      mshift = MeanShift(bandwidth=arg_bandwidth)
36
37      cluster_predict = mshift.fit_predict(X_normal)
38      tiempo = time.time() - t
39      #Pinto resultados
40      print(": {:.2f} segundos, ".format(tiempo), end='')
41
42      return tiempo, mshift, cluster_predict
43
44
45  def dbscan(arg_eps, arg_min_samples, X_normal):
46      print('----- Ejecutando DBSCAN: minPts=' + str(arg_min_samples) + ', eps=' +
        ↪  str(arg_eps), end='')
47
```

```
    #Tomamos tiempos
    t = time.time()
    # Ejecuto el algoritmo y asigno los clusters
    dbscan = DBSCAN(eps=arg_eps, min_samples=arg_min_samples)

    cluster_predict = dbscan.fit_predict(X_normal)
    tiempo = time.time() - t
    #Pinto resultados
    print(": {:.2f} segundos, ".format(tiempo), end='')

    return tiempo, dbscan, cluster_predict

def single_linkage(X_normal, subset, n_clusters_arg=4):
    print('----- Ejecutando Single Linkage Hierarchical Clustering', end='')

    # Tomamos tiempos
    t = time.time()

    # Ejecutamos el algoritmo y asignamos los clusters
    single_linkage = AgglomerativeClustering(n_clusters=n_clusters_arg, linkage='single')
    cluster_predict = single_linkage.fit_predict(X_normal, subset['ponde'])  # se usa
    ↪  peso para cada objeto (factor de elevación)

    tiempo = time.time() - t

    # Pintamos resultados
    print(": {:.2f} segundos, ".format(tiempo), end='')

    return tiempo, single_linkage, cluster_predict
```

1.3 Medidas

Para cada uno de estos, obtendremos distintas medidas que nos informarán sobre el rendimiento y la eficiencia de cada algoritmo:

- **Tiempo de ejecución:** no refleja la eficiencia del algoritmo empleado, pero sí puede servir para decantarnos por uno u otro en un momento dado.
- **Coeficiente silhouette:** su objetivo es medir cómo de similares son los objetos de un mismo cluster, comparado con otros clusters. s(i) toma valores dentro de [-1,1], siendo mejor cuanto más cercano a 1 está (clusters más densos y separados entre sí). Si se acerca a cero, el objeto está al borde de dos clusters.
- **Índice Calinski-Harabasz:** es la razón entre la dispersión intra-clusters y la dispersión inter-clusters. Cuanto mayor es el valor, mejor es el agrupamiento, pues cada grupo será más denso y más distinto con respecto a los demás grupos.

- **Número de clusters:** esta medida no tiene sentido en algoritmos donde se puede fijar de antemano el número de clústers, pero sí en los demás algoritmos pues podemos ver si se generan pocos o muchos clusters y a qué se debe.

1.4 Preprocesado

Con respecto al preprocesado de los datos proporcionados, en primer lugar hemos normalizado los datos utilizando la función `norm_to_zero_one`, para que queden en un rango del 0 al 1. La normalización es útil para asegurar que las variables tengan la misma escala y evitar que una variable domine sobre las demás debido a sus magnitudes.

Posteriormente hemos realizado imputación de datos por vecinos más cercanos, es decir, hemos aplicado el algoritmo KNN (con el imputador `KNNImputer` de scikit-learn).

Finalmente hemos eliminado outliers del conjunto de datos. Consideramos un outlier como aquellos casos fuera de 1.5 veces el rango intercuartil.

Para realizar el preprocesado he definido las siguientes funciones:

```
# Función de normalización
def norm_to_zero_one(df):
    return (df - df.min()) * 1.0 / (df.max() - df.min())

# Imputamos por vecinos más cercanos. También aplicamos normalización
def imputar_valores_perdidos(datos):
    imputer = KNNImputer(n_neighbors=3)
    datos_norm = datos.apply(norm_to_zero_one)
    datos_imputados_array = imputer.fit_transform(datos_norm)
    datos_norm2 = pd.DataFrame(datos_imputados_array, columns = datos.columns)
    datos = datos_norm2
    return datos

def eliminar_outliers(X):
    Q1 = X.quantile(0.25)
    Q3 = X.quantile(0.75)
    IQR = Q3 - Q1
    X = X[~((X < (Q1 - 1.5 * IQR)) |(X > (Q3 + 1.5 * IQR))).any(axis=1)]
    return X
```

2 Casos de Estudio

2.1 Caso de Estudio 1

2.1.1 Descripción del Caso de Estudio

En el primer caso de estudio vamos a segmentar a los encuestados por **nivel de estudios**. Dividiremos en conjunto de datos en dos: aquellas personas con niveles educativos hasta "Segundo grado, segundo ciclo", y aquellas con estudios a partir de "Tercer grado, primer ciclo". Según la variable `educacion_r`, el primer grupo correspondería a los `educacion_r<=2`, y el segundo grupo a `educacion_r>=3` . Una vez normalizados los datos, los dos casos serían `educacion_r<=0.5` y `educacion_r>=0.5`

Las variables de estudio que se pretenden estudiar van ser las relacionadas con la ideología y qué se conoce del conflicto de Israel y Palestina. La finalidad es comprobar si el nivel de estudios es influyente con tener una opinión u otra sobre el conflicto, y a qué se suele tender cuando el nivel de estudios es por ejemplo alto. Entonces las variables que vamos a seleccionar para el análisis son las siguientes cinco:

- **p7**: Ideología (de izquierdas o de derechas, en una escala del 0=izquierda al 10=derecha)
- **p10**: si ha oído hablar acerca de la guerra entre Israel y Hamás. Mientras el valor sea más cercano a 3 (a 1 una vez normalizado), más ha oído hablar de la guerra.
- **p11**: grado de acuerdo con varias afirmaciones sobre la guerra entre Israel y Hamás. Entre ellas he cogido las más polarizables, es decir, las que obligan al encuestado a "declarar" su apoyo a una parte o a la otra.
  - **p11_1**: Israel tiene derecho a defenderse.
  - **p11_2**: El uso de violencia por parte de Hamás está justificado.
- **p13**: valoración sobre la respuesta que han dado distintas instituciones y partidos.
  - **p13_2**: respuesta de la ONU

El conjunto de datos estudiado en el caso de estudio 1A está formado por **1093 individuos**. El conjunto de datos estudiado en el caso de estudio 1B está formado por **907 individuos**.

2.1.2 Caso 1A: Resultados de los Algoritmos de Clustering

Estos son los resultados de ejecutar en python (en jupyter notebook) los cinco algoritmos seleccionados. El valor de k (número de clusters) para kmeans y single_linkage es de k=4. En meanshift se estime el valor de bandwidth (radio) dentro de la función, aunque hemos fijado el valor de quantile=0.05, es decir, hemos tomado el 5% de los datos para estimar el radio. Si no hicieramos esto, sólo se produce un cluster pues sale un bandwith demasiado grande. Para el algoritmo de birch, hemos fijado branching_factor=15 y threshold=0.1. Por último, para DBSCAN, hemos tomado minPts=15 y eps=0.126. Cabe decir que he ajustado estos parámetros para que me de un número de clusters similar en todos los algoritmos.

Algoritmo	Tiempo (s)	Calinski-Harabasz	Silhouette	Número de clusters
kmeans	0.057	281.540	0.22860	4
birch	0.091	182.466	0.12856	4
meanshift	3.125	95.582	0.11052	5
dbscan	0.014	5.802	-0.31693	4
single_linkage	0.009	77.208	0.11784	4

Table 1: Resultados de los algoritmos de clustering

En primer lugar nos llama la atención que todos los algoritmos han tenido tiempos de ejecución despreciables, excepto el algoritmo meanshift, que ha tardado un tiempo razonablemente alto. Sin embargo, no ha obtenido mejores resultados que otros algoritmos. Esto lo achacamos a que tiene que calcular el bandwith, que puede resultar una operación costosa.

Observamos que hay algoritmos, el que más el DBSCAN, que tienen un índice de Calinski-Harabasz muy bajo. Esto sugiere que los puntos dentro de los clusters están dispersos y no forman grupos claramente distinguibles. Es fácil de comprender si vemos cuántos elementos forma cada uno de los 4 clusters, vemos que todos los datos están concentrados en el primer cluster:

```
----- Ejecutando DBSCAN: minPts=15, eps=0.126: 0.01 segundos,

Medidas


Calinski-Harabasz Index: 5.802, Silhouette Coefficient: -0.31693
Tamaño de cada cluster:
0:  1033 (94.51%)
1:    23 ( 2.10%)
2:    16 ( 1.46%)
3:    21 ( 1.92%)
```

Figure 1: Resultados DBSCAN

Podemos concluir por tanto que DBSCAN no ha hecho un buen agrupamiento. Se podría decir algo similar de Meanshift y Single_linkage, que tienen también valores bajos de Calinski-Harabasz. El algoritmo con mejores resultados en ese sentido es K-means, y podemos observar que los cuatro clusters tienen un número similar de elementos.

2.1.3 Caso 1A: Visualización

Para el análisis vamos a tomar los resultados del algoritmo K-means, pues ha obtenido unos resultados bastante aceptables. Vemos que este algoritmo genera 4 clusters con un número muy similar de elementos en cada cluster, como podemos ver en la Figura 2.

Hemos generado el gráfico de barras con el siguiente código:

```
# Crear un DataFrame a partir de la lista de diccionarios
df = pd.DataFrame(tabla_clusters[0])

# Creamos el gráfico de barras
```

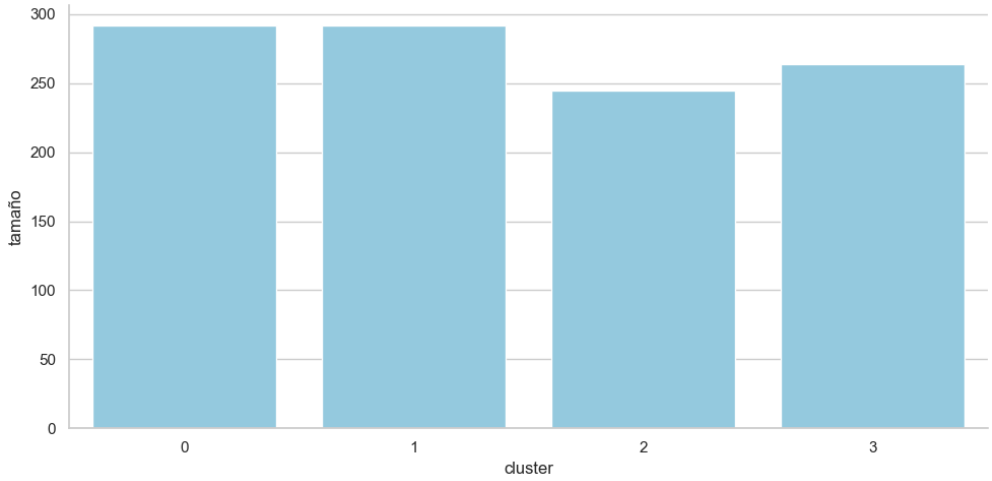

Figure 2: Gráfico de barras tamaño clusters

```
sns.set(style="whitegrid")
sns.catplot(y="tamaño", x="cluster", data=df, kind="bar", color="skyblue", aspect=2)
plt.show()
```

Seguimos con un Heatmap(3), que nos proporciona información sobre la relación entre diferentes clusters y sus similitudes. Es una herramienta útil para analizar la calidad y la separación de los clusters generados por el algoritmo KMeans.

Para generar el Heatmap hemos empleado el siguiente código:

```
### VISUALIZACIÓN
def heatmap(k_means):
    print("---------- Heatmap de centroides...")
    centers = pd.DataFrame(k_means.cluster_centers_,columns=list(X))
    centers_desnormal = centers.copy()

    # se convierten los centros a los rangos originales antes de normalizar
    for var in list(centers):
        centers_desnormal[var] = X[var].min() + centers[var] * (X[var].max() -
        ↪  X[var].min())

    plt.figure()
    centers.index += 1
    plt.figure()
    hm = sns.heatmap(centers, cmap="YlGnBu", annot=centers_desnormal,
    ↪  annot_kws={"fontsize":18}, fmt='.3f')
    hm.set_ylim(len(centers),0)
    hm.figure.set_size_inches(15,15)
    hm.figure.savefig("centroides.pdf")
    centers.index -= 1
```

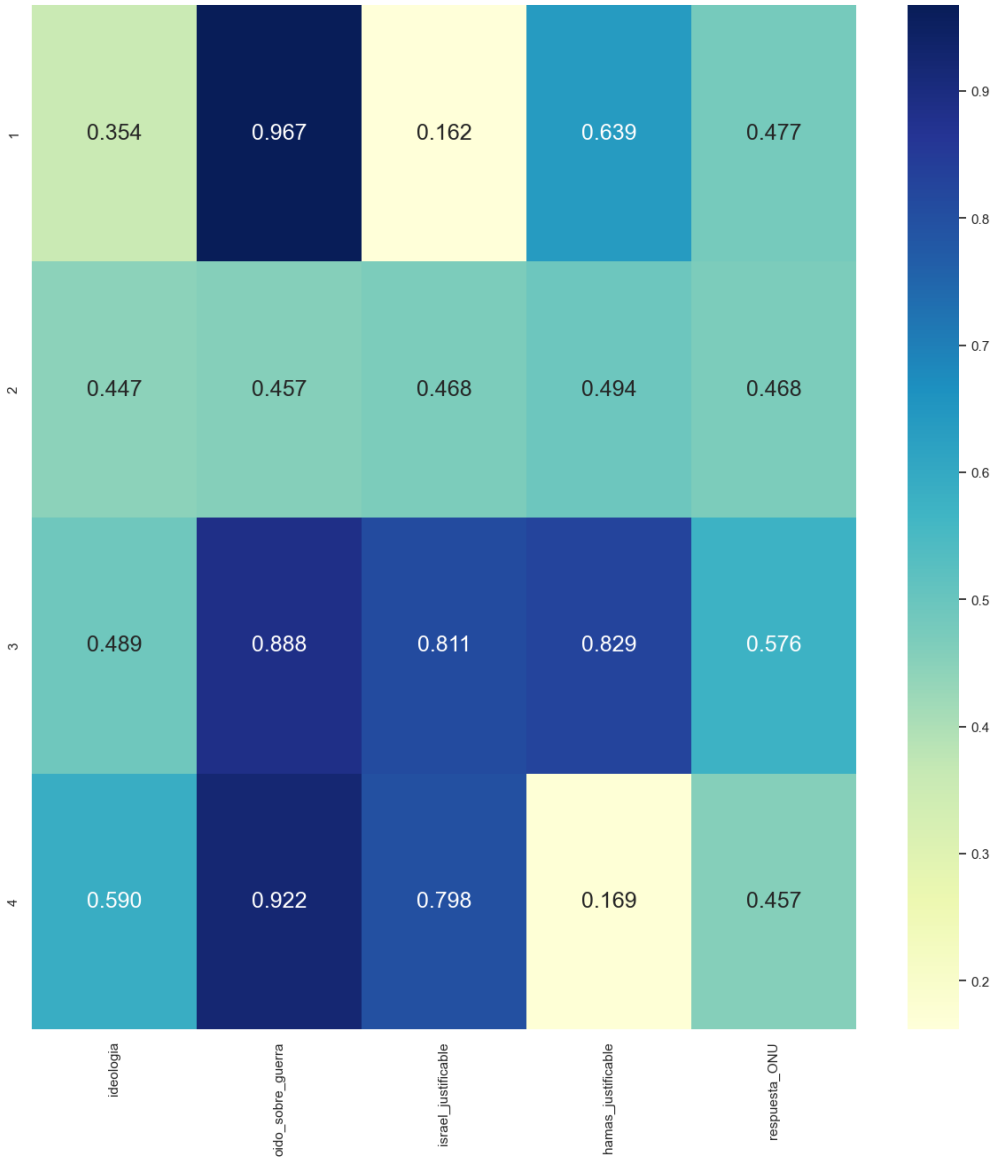

Figure 3: Heatmap para K-means

Observamos que en los cuatro clusters el atributo ideología (p7) toma unos valores similares, por lo que es poco discriminante. Aun así, podríamos decir que el primer cluster representa a gente un poco más de izquierdas y avanza progresivamente hacia el cuarto cluster a gente un poco más de derechas. Por otro lado, vemos que el segundo atributo oido_sobre_guerra (p10) es bastante discriminante pues en todos los clusters toma valores muy altos excepto en el segundo cluster. Parece que los clusters 1, 3 y 4 representan a la gente que está bastante informada sobre la guerra. En cuanto al tercer y cuarto atributos, que corresponden a opiniones "polarizadas" entre sí, vemos que queda cluster representa una situación distinta. El primer cluster representa a la gente que no está nada de acuerdo con las acciones que está llevando a cabo Israel, mientras que está bastante de acuerdo con la violencia ejercida por parte de Hamás. El segundo cluster parece representar a la gente que mantiene una opinión neutra tanto con respecto una parte del conflicto como con la otra. El tercer cluster parece agrupar a la gente que está de acuerdo tanto con las acciones de Israel como con las de Hamás.

El cuarto cluster representa a la gente de acuerdo con Israel y en contra de Hamás. Por último, el quinto atributo (p13_2), es poco discriminante pues todos sus valores son similares.

Podemos deducir entonces que el primer cluster representa a gente ligeramente de izquierdas que ha seguido el conflicto con bastante detalle, que está indignada con las acciones de Israel mientras que apoya un poco más a Hamás. El segundo cluster representaría a gente de centro, que no está muy informada y que tiene una opinión bastante neutra sobre el conflicto. El tercer cluster representa a gente de centro que está bien informada y apoya las acciones de ambas partes del conflicto. El cuarto cluster representa a gente ligeramente de derechas, bien informada y que apoya únicamente las acciones de Israel.

A continuación, en la Figura 4 mostramos la Scatter Matrix, donde en cada celda encontramos un gráfico que muestra la dispersión entre dos variables. Aquí se aprecian cómo se van separando los clusters y las variables necesarias para separar cada uno.
Para generar el Scatter Matrix hemos empleado el siguiente código:

```
def scatter(X_kmeans, colors):
    print("---------- Scatter matrix...")
    plt.figure()
    sns.set()
    variables = list(X_kmeans)
    variables.remove('cluster')
    X_kmeans['cluster'] += 1
    sns_plot = sns.pairplot(X_kmeans, vars=variables, hue="cluster", palette=colors,
    ↪  plot_kws={"s": 25}, diag_kind="hist") #en hue indicamos que la columna 'cluster'
    ↪  define los colores
    X_kmeans['cluster'] -= 1
    sns_plot.fig.subplots_adjust(wspace=.03, hspace=.03)
    sns_plot.fig.set_size_inches(15,15)
    sns_plot.savefig("C:/Users/monic/Documents/DGIIM5/IN/practicas/p2/scatter.pdf")
    plt.show()
```

Por último, mostramos la distancia relativa entre centroides de los clusters, en la Figura 5. El radio del círculo (y por tanto su tamaño) es proporcional al número de objetos en cada cluster. En este caso, como el número de elementos en cada cluster está muy equilibrado, los tamaños de los círculos serán similares. Observamos que los clusters se encuentran bastante separados entre sí, por lo que podríamos concluir que se ha realizado bien el agrupamiento. Podemos sacar poca más información pues el gráfico no muestra nada relacionado con los diferentes atributos. Para generar la figura hemos empleado el siguiente código:

```
def MDS_plot(centers, size):
    print("---------- MDS...")
    X_kmeans = pd.concat([X, clusters], axis=1)
    k = len(size)
    colors = sns.color_palette(palette='Paired', n_colors=k, desat=None)
```

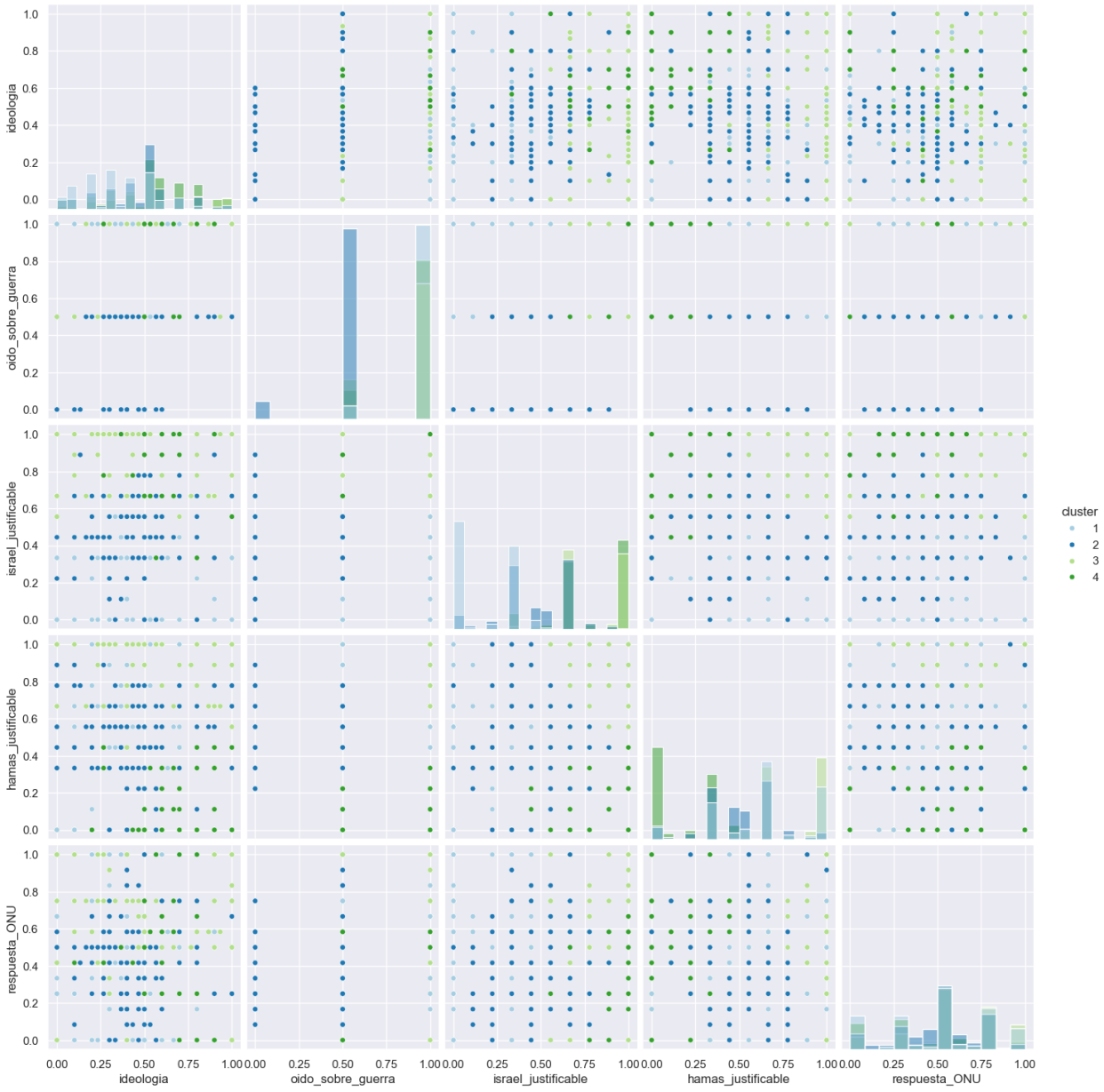

Figure 4: Scatter Matrix para K-means

```
6
7       mds = MDS()
8       centers_mds = mds.fit_transform(centers)
9       fig=plt.figure(4)
10
11      plt.scatter(centers_mds[:,0], centers_mds[:,1], s=size*10, alpha=0.75, c=colors)
12      for i in range(k):
13          plt.annotate(str(i+1),xy=centers_mds[i],fontsize=18,va='center',ha='center')
14      plt.xticks(fontsize=18)
15      plt.yticks(fontsize=18)
```

```
16      fig.set_size_inches(15,15)
17
18      nombre = "mds.png"
19      plt.savefig(nombre)
```

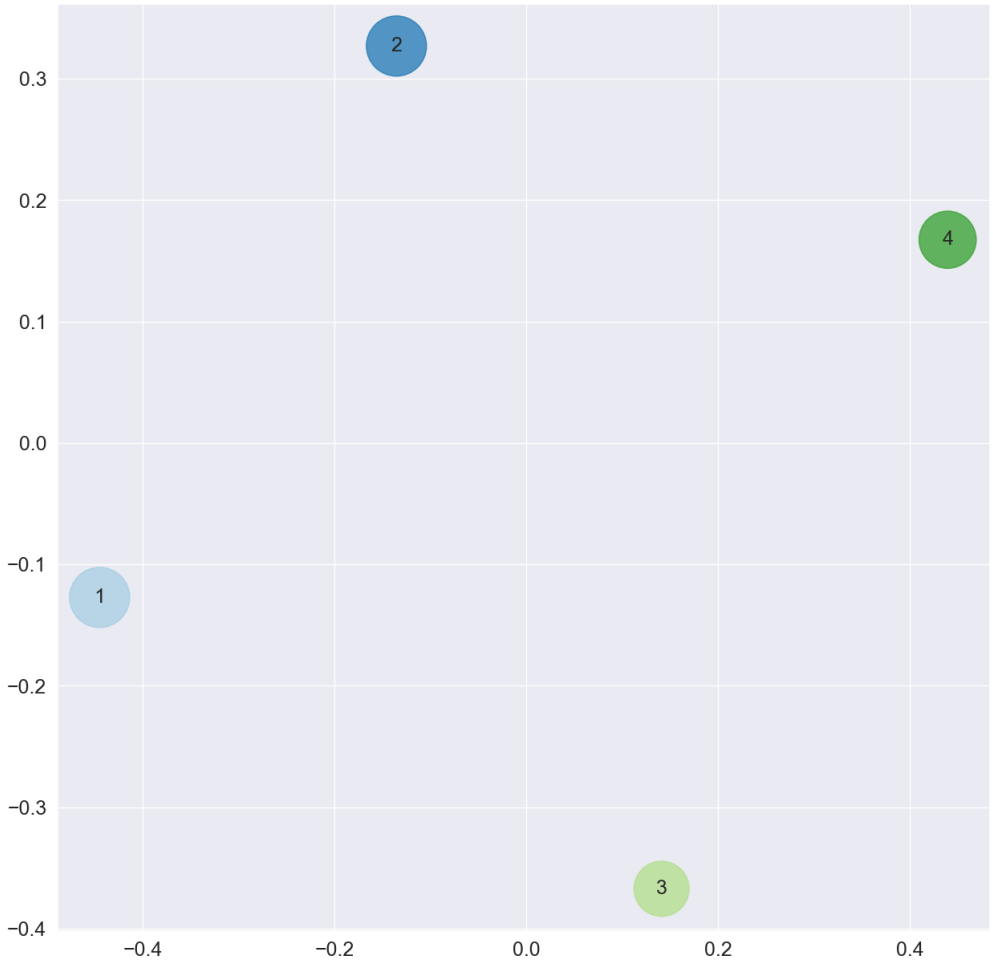

Figure 5: Distancia relativa entre centroides de clusters

En la tabla 2.1.7 se muestran qué variables son necesarias para identificar cada cluster:

Cluster	ideologia	oido sobre guerra	Israel justificable	Hamas justificable	respuesta ONU
1			Bajo	Alto	
2			Medio	Medio	
3			Alto	Alto	
4			Alto	Bajo	

Table 2: Variables necesarias para identificar cada cluster

Con solo los atributos israel_justificable y hamas_justificable conseguimos identificar los cuatro clusters.

2.1.4 Caso 1A: Estudio de Parámetros - Algoritmo DBSCAN

Vamos a analizar el comportamiento de DBSCAN según sus dos parámetros. Hemos variado el valor de los parámetros comenzando con minPts=15, y eps=0.12. A continuación se muestran los resultados:

Algoritmo	eps	minPts	Tiempo (s)	Silhouette	Calinski-Harabasz	Número de clusters
dbscan	0.120	15.000	0.029	-0.317	5.80219	4
dbscan	0.130	15.000	0.029	-0.315	6.04675	4
dbscan	0.150	15.000	0.035	-0.309	7.07495	5
dbscan	0.170	15.000	0.032	-0.286	8.76053	5
dbscan	0.200	15.000	0.044	-0.185	21.86313	6
dbscan	0.120	20.000	0.044	-0.213	3.69383	2
dbscan	0.130	20.000	0.043	-0.213	3.69383	2
dbscan	0.150	20.000	0.030	-0.203	4.65679	2
dbscan	0.170	20.000	0.031	-0.285	7.01732	4
dbscan	0.200	20.000	0.034	-0.241	10.77351	5
dbscan	0.120	25.000	0.030	-0.241	10.77351	1
dbscan	0.130	25.000	0.031	-0.241	10.77351	1
dbscan	0.150	25.000	0.030	-0.241	10.77351	1
dbscan	0.170	25.000	0.030	-0.241	10.77351	1
dbscan	0.200	25.000	0.032	-0.279	8.46059	4
dbscan	0.120	30.000	0.029	-0.279	8.46059	1
dbscan	0.130	30.000	0.030	-0.279	8.46059	1
dbscan	0.150	30.000	0.028	-0.279	8.46059	1
dbscan	0.170	30.000	0.029	-0.279	8.46059	1
dbscan	0.200	30.000	0.028	-0.279	8.46059	1
dbscan	0.120	35.000	0.022	-0.279	8.46059	1
dbscan	0.130	35.000	0.022	-0.279	8.46059	1
dbscan	0.150	35.000	0.022	-0.279	8.46059	1
dbscan	0.170	35.000	0.021	-0.279	8.46059	1
dbscan	0.200	35.000	0.025	-0.279	8.46059	1

Table 3: Resultados de DBSCAN por parámetros

Vemos que el tiempo de ejecución se mantiene constante todo el rato, por lo que no nos vamos a fijar en esto. Podemos apreciar que DBSCAN va generando entre 2 y 6 clusters hasta que el valor de minPts supera el 25, a partir de ahí sólo genera un cluster. Además podemos observar que para todos los casos el índice de Calinski-Harabasz es muy bajo, al igual que el coeficiente de Silhouette, por lo que podemos concluir que DBSCAN no es apropiado para el caso de estudio definido.

2.1.5 Caso 1A: Estudio de Parámetros - Algoritmo BIRCH

Vamos a analizar el comportamiento de BIRCH según sus dos parámetros. Hemos variado el valor de los parámetros comenzando con branding factor=15, y threshold=0.1. A continuación se muestran los resultados:

Algorit	brand fact	threshold	Tiempo (s)	Silhouette	Calinski-Harabasz	nº clusters
birch	15.000	0.100	0.039	0.129	182.46621	4
birch	15.000	0.120	0.060	0.168	215.46119	4
birch	15.000	0.150	0.057	0.163	205.23887	4
birch	15.000	0.200	0.053	0.128	171.94368	4
birch	15.000	0.250	0.022	0.143	192.48083	4
birch	20.000	0.100	0.070	0.171	219.97098	4
birch	20.000	0.120	0.052	0.172	205.50396	4
birch	20.000	0.150	0.061	0.164	213.54106	4
birch	20.000	0.200	0.051	0.155	208.13983	4
birch	20.000	0.250	0.039	0.129	179.55461	4
birch	25.000	0.100	0.058	0.120	181.96023	4
birch	25.000	0.120	0.063	0.146	192.36345	4
birch	25.000	0.150	0.056	0.106	156.97155	4
birch	25.000	0.200	0.037	0.151	187.19656	4
birch	25.000	0.250	0.037	0.153	198.57197	4
birch	30.000	0.100	0.049	0.167	207.29454	4
birch	30.000	0.120	0.051	0.144	190.58724	4
birch	30.000	0.150	0.037	0.159	202.14362	4
birch	30.000	0.200	0.026	0.143	185.08682	4
birch	30.000	0.250	0.030	0.116	165.16270	4
birch	35.000	0.100	0.048	0.122	181.10394	4
birch	35.000	0.120	0.054	0.181	218.52828	4
birch	35.000	0.150	0.047	0.166	209.39206	4
birch	35.000	0.200	0.052	0.156	203.88134	4
birch	35.000	0.250	0.045	0.118	173.49033	4

Table 4: Resultados de DBSCAN por parámetros

Observamos que el algoritmo agrupa siempre en 4 clusters y obtiene bastante buenos resultados, similares a los de K-means. Podemos afirmar que ha hecho buenas agrupaciones a juzgar por el índice de Calinski-Harabasz. El tiempo de ejecución es similar aunque vayamos variando los parámetros.

2.1.6 Caso 1B: Resultados de los Algoritmos de Clustering

Estos son los resultados de ejecutar en python (en jupyter notebook) los cinco algoritmos seleccionados. Los valores de los parámetros son los mismos que en el caso 1A.

Algoritmo	Tiempo (s)	Calinski-Harabasz	Silhouette	Número de clusters
kmeans	0.058	251.908	0.21843	4
birch	0.085	208.682	0.20020	4
meanshift	1.772	69.249	0.07590	9
dbscan	0.006	3.904	-0.26760	3
single_linkage	0.003	3.385	0.12683	4

Table 5: Resultados de los algoritmos de clustering

De nuevo, todos los algoritmos han tenido tiempos de ejecución despreciables, excepto el algoritmo Meanshift, que ha tardado un tiempo razonablemente alto, por las mismas razones que antes.

Observamos que kmeans y birch han obtenido buenos resultados, mientras que los otros tres algoritmos han tenido un índice de Calinski-Harabasz muy bajo, al igual que antes.

Nos llama la atención que Meanshift ha obtenido 9 clusters. Veamos si están bien distribuido el conjunto de datos(2):

```
Calinski-Harabasz Index: 69.249, Silhouette Coefficient: 0.07590
Tamaño de cada cluster:
0:   499 (55.02%)
1:   183 (20.18%)
2:   116 (12.79%)
3:    49 ( 5.40%)
4:     6 ( 0.66%)
5:    25 ( 2.76%)
6:    17 ( 1.87%)
7:     6 ( 0.66%)
8:     6 ( 0.66%)
```

Figure 6: Resultados Meanshift

Podemos concluir por tanto que Meanshift no ha hecho un buen agrupamiento, ya que el primer cluster contiene el 55% del conjunto de datos y luego hay clusters con apenas 6 elementos.

2.1.7 Caso 1B: Visualización

Para el análisis vamos a tomar otra vez los resultados del algoritmo K-means, para que esto no influya en la interpretación. Vemos que este algoritmo genera 4 clusters con un número muy similar de elementos en cada cluster, como podemos ver en la Figura 7.

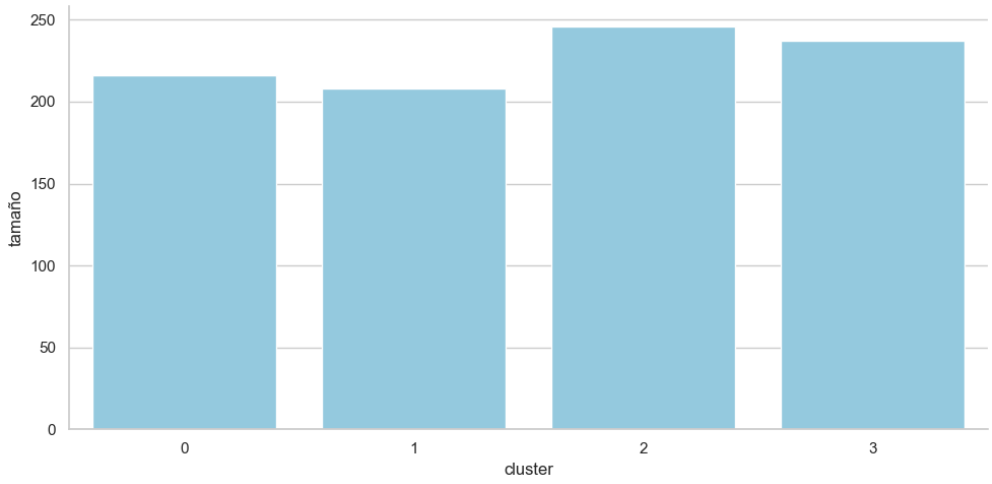

Figure 7: Gráfico de barras tamaño clusters

Seguimos con el Heatmap(8):

Observamos que de nuevo en los cuatro clusters el atributo ideología (p7) toma unos valores similares, por lo que es poco discriminante. Aun así, podríamos decir que el cuarto cluster representa a gente un poco más de izquierdas y el tercer cluster a gente un poco más de derechas. Por otro lado, vemos que el segundo atributo oido_sobre_guerra (p10) es bastante discriminante pues en todos los clusters toma valores muy altos. Esto parece indicar que las personas con estudios altos están bastante bien informadas sobre los temas de actualidad como el de la guerra de Israel y Palestina. En cuanto al tercer y cuarto atributos, que corresponden a opiniones "polarizadas" entre sí, vemos que cada cluster representa una situación distinta. El primer cluster representa a la gente que no está nada de acuerdo con las acciones que no están nada de acuerdo con las acciones que están llevando ambas partes del conflicto. El segundo cluster parece contener a la gente que está de acuerdo tanto con las acciones de Israel como con las de Hamás. El tercer cluster parece agrupar a gente a favor de Israel y en contra de Hamás. El cuarto cluster representa a la gente en contra de Israel y de acuerdo con Hamás. Por último, el quinto atributo (p13_2), es poco discriminante pues todos sus valores son similares, así que no lo vamos a tener muy en cuenta.

Podríamos deducir que tanto el primer como el segundo cluster representa a gente de ideología de centro que ha seguido el conflicto con detalle, sólo que en el primer cluster están en contra de ambas partes del conflicto, y en el segundo cluster están a favor de ambas. Nos encontramos entonces ante dos grupos que son idénticos en cuanto a condiciones ideológicas y de información, pero tienen opiniones totalmente contrarias. El tercer cluster representaría a gente que tiende ligeramente a la ideología de derechas, que estaría a favor de Israel y en contra de Hamás. El cuarto cluster contiene a la gente que tiende ligeramente a la ideología de izquierdas, que estaría a favor de Hamás y en contra de Israel.

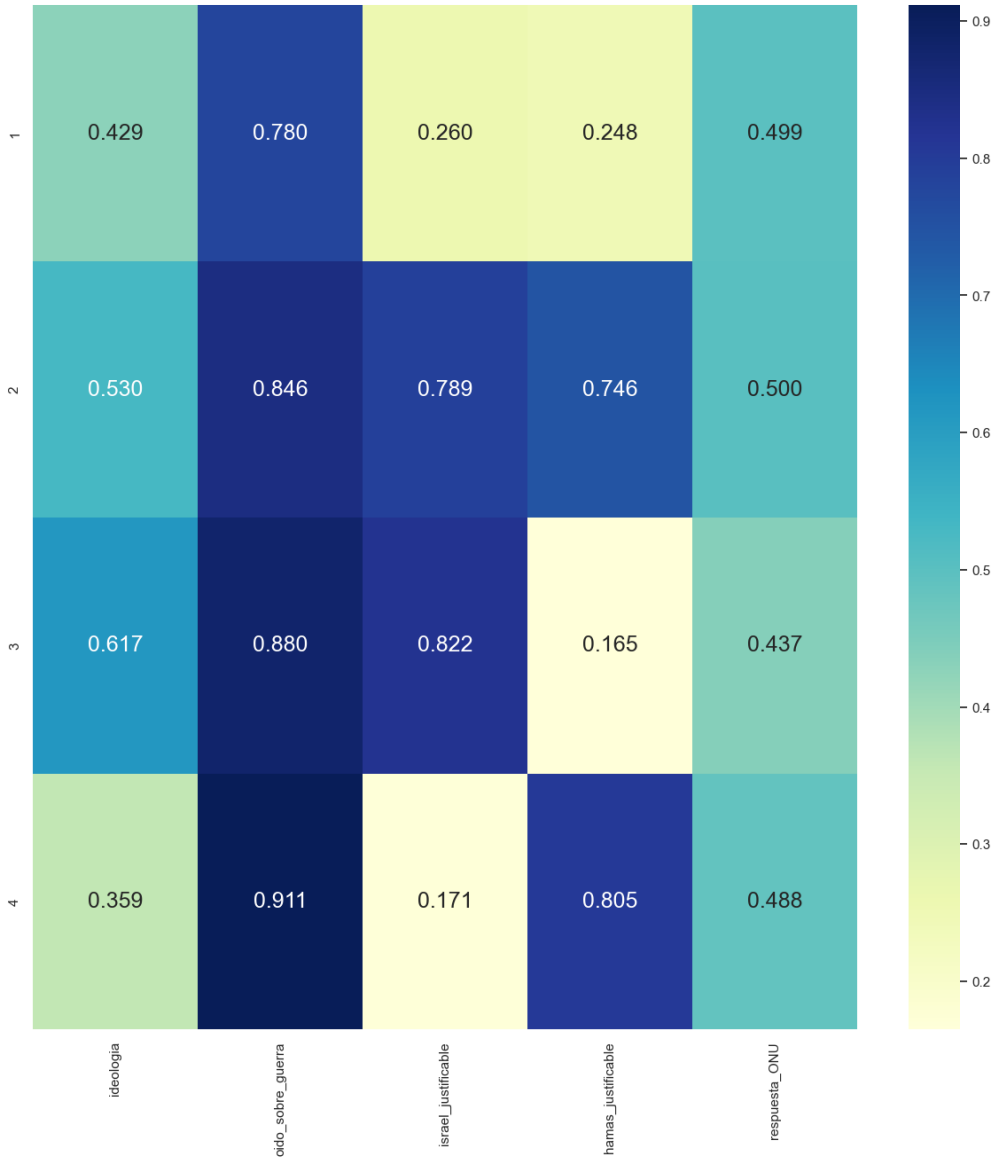

Figure 8: Heatmap para K-means

A continuación, en la Figura 9 mostramos la Scatter Matrix.

Por último, mostramos la distancia relativa entre centroides de los clusters, en la Figura 10. Observamos que los clusters se encuentran de nuevo bastante separados entre sí, por lo que podríamos concluir que se ha realizado bien el agrupamiento. Podemos sacar poca más información pues el gráfico no muestra nada relacionado con los diferentes atributos.

En la tabla 2.1.7 se muestran qué variables son necesarias para identificar cada cluster:

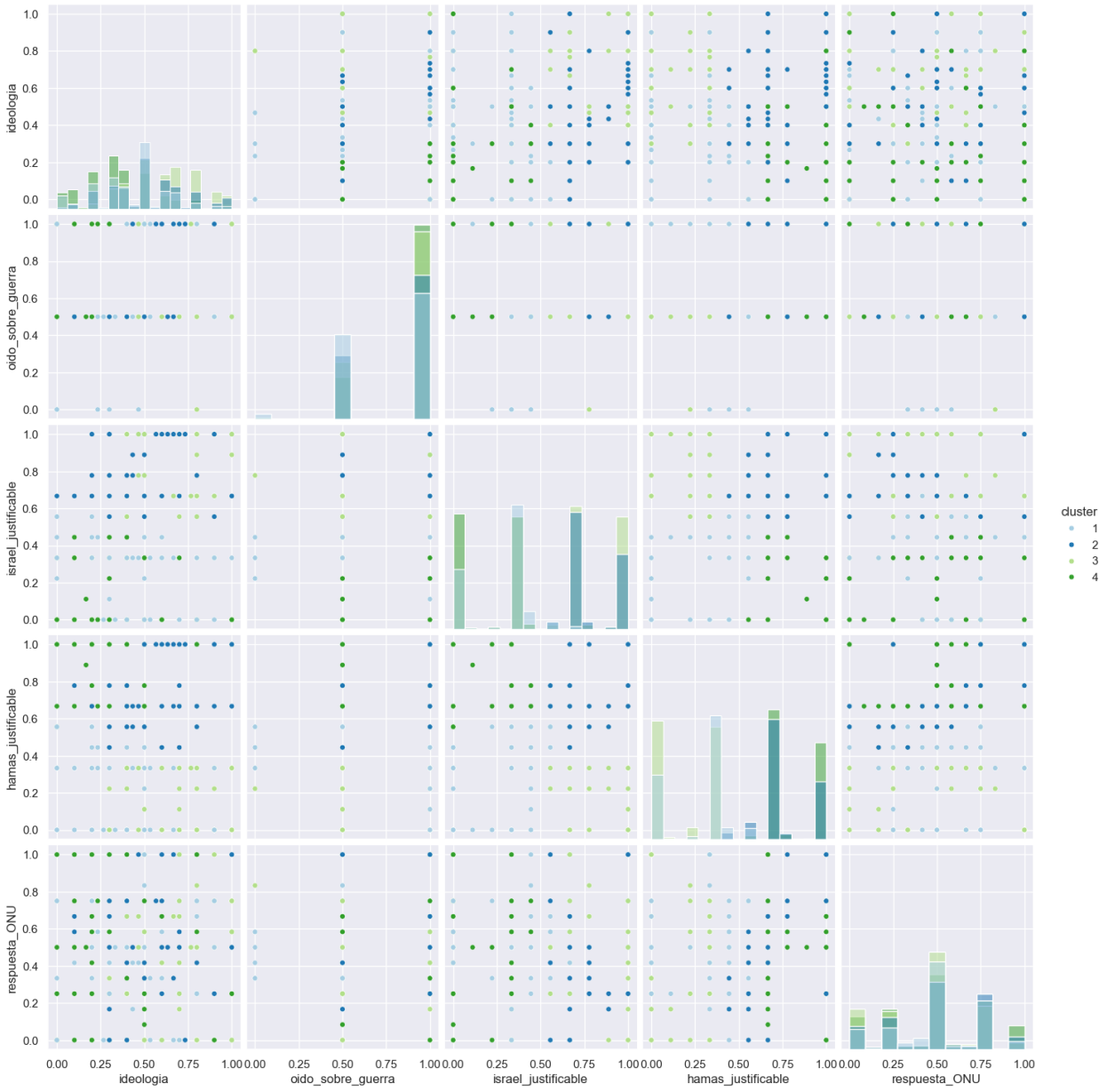

Figure 9: Scatter Matrix para K-means

Cluster	ideologia	oido sobre guerra	Israel justificable	Hamas justificable	respuesta ONU
1			Bajo	Bajo	
2			Alto	Alto	
3			Alto	Bajo	
4			Bajo	Alto	

Table 6: Variables necesarias para identificar cada cluster

Con solo los atributos israel_justificable y hamas_justificable conseguimos identificar los cuatro clusters.

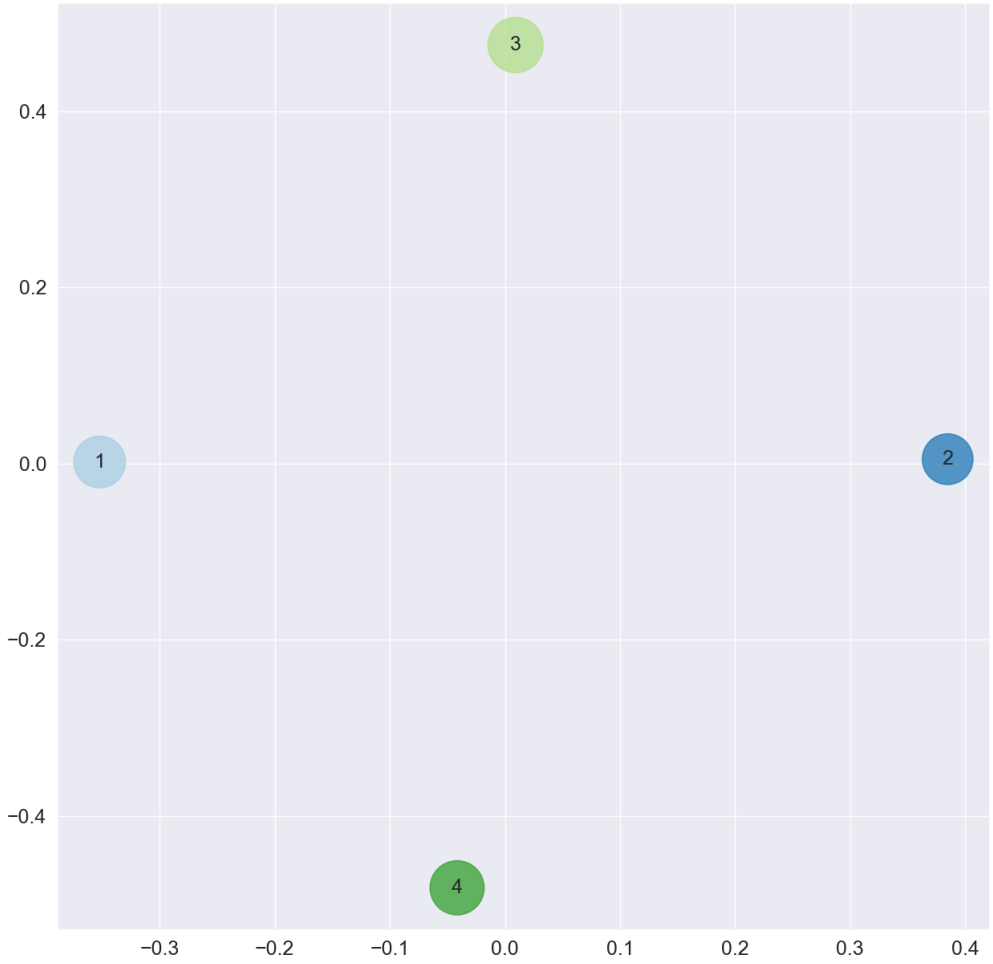

Figure 10: Distancia relativa entre centroides de clusters

2.1.8 Interpretación de la Segmentación

Como conclusión general del caso de estudio 1, podemos afirmar que el nivel de estudios no es determinante a la hora de tener una opinión u otra sobre el conflicto de Hamás. De hecho, hemos notado que tanto en el caso 1A como en el 1B, se han formado los mismos 4 grupos diferentes de opinión. Únicamente se podría señalar que la gente con más estudios tiende a estar un poco más informada sobre la guerra. Asímismo, hemos notado una pequeña tendencia de la gente de izquierdas a estar poco de acuerdo con las acciones de Israel y un poco más de acuerdo las de Hamás. Por el contrario, la gente ligeramente de derechas tiende a pensar al revés. De todas formas, no podemos sacar conclusiones de aquí ya que la ideología no ha quedado bien discriminada en los gráficos generados para el análisis, ya que no se han obtenido valores extremos en los mapas de calor.

2.2 Caso de Estudio 2

2.2.1 Descripción del Caso de Estudio

En este segundo caso de estudio vamos a segmentar a los encuestados por **rango de edad**. En particular, vamos a tomar a las personas con un rango de edad entre 18 y 34 años (rangos 1 y 2 dentro de edad_r.csv).

Las variables que se pretenden estudiar están relacionadas con la intención de voto y la preocupación con varios problemas de actualidad. La finalidad es comprobar si los jóvenes están concienciados con los problemas que atañen el mundo y qué partido creen que es la mejor opción para resolver dichos problemas. Las variables seleccionadas para el análisis son las siguientes:

- **p5**: Probabilidad de votar a los cuatro principales partidos (del 0 al 10).
  - **p5_1**: Probabilidad de votar al PSOE
  - **p5_2**: Probabilidad de votar al PP
  - **p5_3**: Probabilidad de votar a Vox
  - **p5_4**: Probabilidad de votar a Sumar
- **p8**: ¿En qué medida sientes como una amenaza cada uno de los siguientes asuntos de carácter global?
  - **p8_1**: El cambio climático y los desastres naturales
  - **p8_3**: Los flujos migratorios
  - **p8_5**: Las crisis de inflación que encarecen el coste de la vida
  - **p8_9**: Las pandemias

El conjunto de datos estudiado en el caso de estudio 1B está formado por **379 individuos**.

2.2.2 Resultados de los Algoritmos de Clustering

Estos son los resultados de ejecutar en python (en jupyter notebook) los cinco algoritmos seleccionados. El valor de k (número de clusters) para kmeans y single_linkage es de k=4. En meanshift se estime el valor de bandwidth (radio) dentro de la función, aunque hemos fijado el valor de quantile=0.05, es decir, hemos tomado el 5% de los datos para estimar el radio. Si no hicieramos esto, sólo se produce un cluster pues sale un bandwith demasiado grande. Para el algoritmo de birch, hemos fijado branching_factor=15 y threshold=0.1. Por último, para DBSCAN, hemos tomado minPts=15 y eps=0.126.

Algoritmo	Tiempo (s)	Calinski-Harabasz	Silhouette	Número de clusters
kmeans	0.045	83.399	0.17788	4
birch	0.042	66.952	0.15666	4
meanshift	1.472	26.202	0.09435	11
dbscan	0.002			1
single_linkage	0.002	1.706	0.02411	4

Table 7: Resultados de los algoritmos de clustering

En primer lugar, al igual que en el caso de estudio 1, todos los algoritmos han tenido tiempos de ejecución despreciables, excepto Meanshift. Observamos que Meanshift ha producido 11 clusters, un número muy elevado. Veamos qué ocurre: Vemos que casi todos los

```
Calinski-Harabasz Index: 26.202, Silhouette Coefficient: 0.09435
Tamaño de cada cluster:
0:   137 (36.15%)
1:    68 (17.94%)
2:   129 (34.04%)
3:     8 ( 2.11%)
4:    11 ( 2.90%)
5:     9 ( 2.37%)
6:     3 ( 0.79%)
7:     2 ( 0.53%)
8:     5 ( 1.32%)
9:     4 ( 1.06%)
10:    3 ( 0.79%)
```

Figure 11: Resultados Meanshift

objetos están concentrados en los clusters 0 y 2. Además, el índice de Calinski-Harabasz es muy bajo, por lo que Meanshift parece no haber hecho un buen agrupamiento.

Por otro lado, DBSCAN sólo forma un cluster. He probado con distintos valores de los parámetros y con todos me sigue saliendo un solo cluster por lo que parece ser que este algoritmo no sirve para el problema definido.

Por último, single_linkage tiene un índice de Calinski-Harabasz muy bajo, y esto se debe a que aunque tiene 4 clusters, está agrupando todos los objetos en el primero, dejando los demás clusters con sólo un elemento.

2.2.3 Visualización

Para el análisis vamos a tomar los resultados del algoritmo K-means, con k=4, pues ha obtenido unos resultados bastante aceptables. Este algoritmo genera 5 clusters bastante compensado entre sí, como podemos ver en la Figura 12.

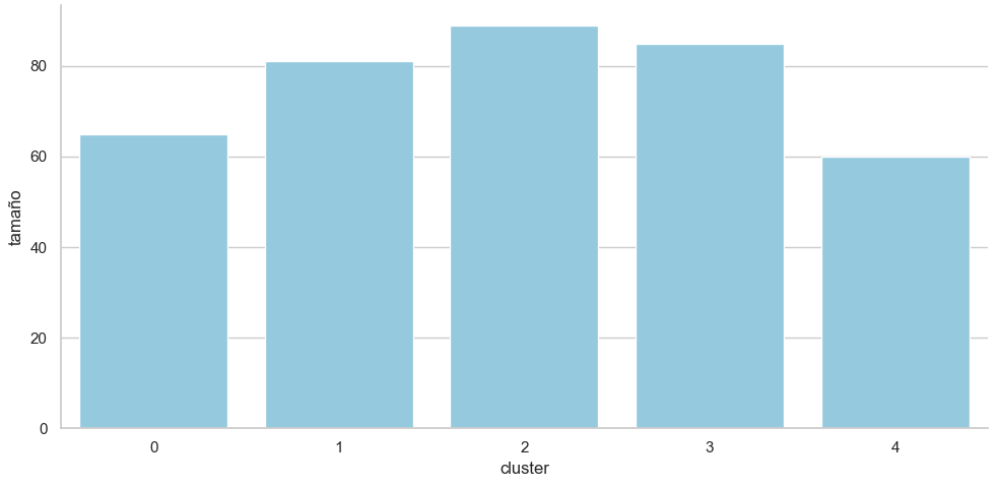

Figure 12: Gráfico de barras tamaño clusters

Seguimos con un Heatmap(13):

Observamos que el primer cluster representa a los votantes del PSOE y a los PP (bipartidismo). En el segundo cluster tenemos a los votantes del PSOE y Sumar ('considerados' de izquierdas), mientras que en el tercer cluster tenemos a los votantes del PSOE y VOX ('considerados' de derechas). El cuarto cluster está representado por los votantes del PSOE de nuevo. Por último, el cluster cinco representaría a las personas 'apolíticas' o que no están de acuerdo con ninguno de los partidos hegemónicos.

Con respecto a los atributos sobre las preguntas de actualidad, vemos que el cambio climático no deja indiferente a ningún grupo, aunque los más preocupados son los clusters 1 y 2. Lo mismo podemos decir sobre los flujos migratorios, aunque el grupo más preocupado es el del cluster 3, que casualmente representa a los partidos de derecha. Con respecto a la inflación, todos los grupos parecen muy preocupados, por lo que no es un atributo muy discriminante, aunque destaca con un valor muy alto el cluster 2. Por último, la pandemia preocupa un poco menos en general, siendo el cluster más preocupado el primero.

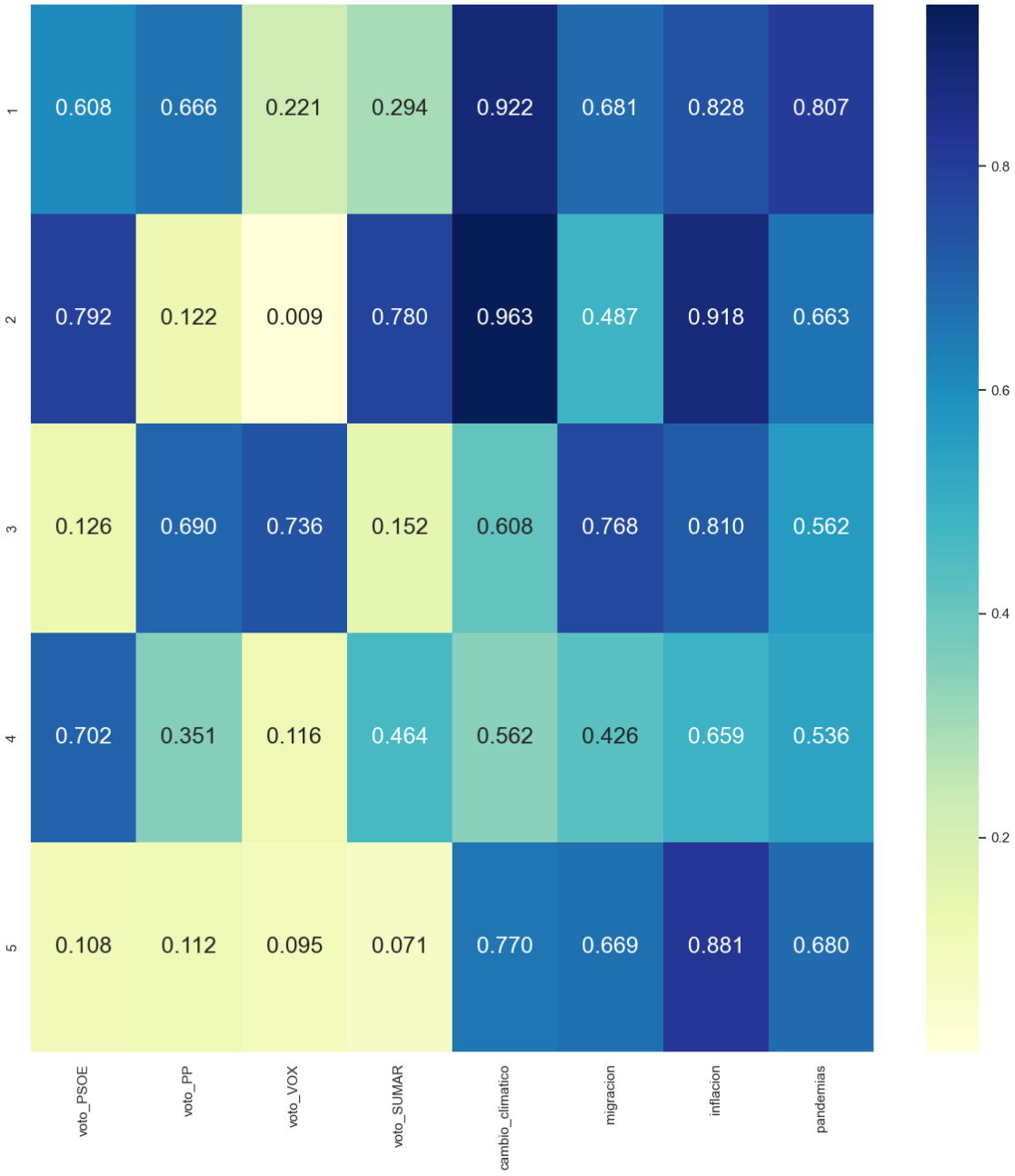

Figure 13: Heatmap para K-means

A continuación, en la Figura 14 mostramos la Scatter Matrix:

Vemos que la dispersión es demasiado grande, por lo que no podemos obtener una dispersión lo suficientemente clara de los clusters. Llama la atención las graficas unidimensionales pues las variables son discretas y los datos se encuentras apilados en un diagrama de barras.

Figure 14: Scatter Matrix para K-means

Por último, mostramos la distancia relativa entre centroides de los clusters, en la Figura 15. Podemos sacar poca información pues el gráfico no muestra nada relacionado con los diferentes atributos. Se aprecia cómo los clusters están muy separados y con tamaños (radios) similares, por lo que podríamos concluir que se ha realizado bien el agrupamiento.

En la tabla 2.2.3 se muestran qué variables son necesarias para identificar cada cluster:

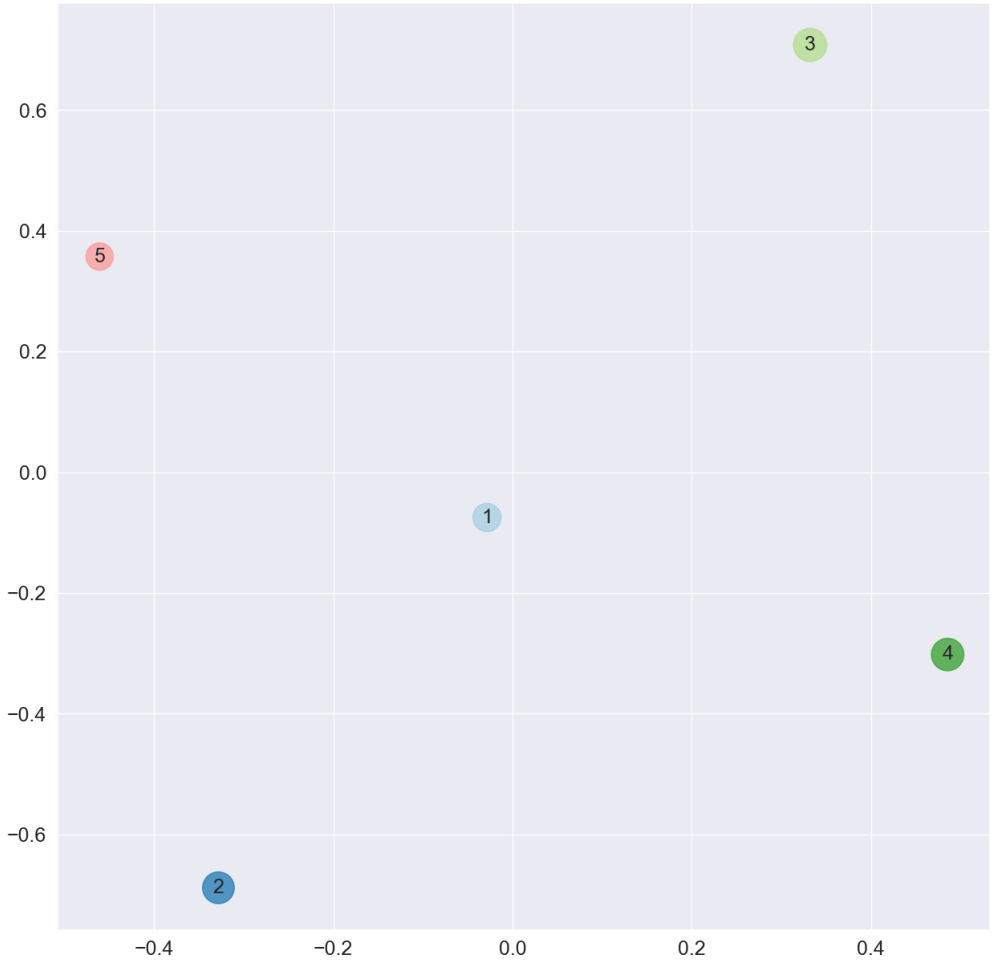

Figure 15: Distancia relativa entre centroides de clusters

Clust	voto PSOE	voto PP	voto Vox	voto Sumar	camb climat	migrac	inflac	pandem
1	Alto	Alto	Bajo	Bajo				
2	Alto	Bajo	Bajo	Alto				
3	Bajo	Alto	Alto	Bajo				
4	Alto	Bajo	Bajo	Bajo				
5	Bajo	Bajo	Bajo	Bajo				

Table 8: Variables necesarias para identificar cada cluster

Con solo las cuatro primeras variables (simpatía con los partidos más grandes) conseguimos identificar los cinco clusters.

2.2.4 Estudio de Parámetros - Algoritmo Meanshift

Vamos a analizar el comportamiento de Meanshift según su parámetro bandwith. Hemos variado el valor del parámetro desde 0.1 hasta 1, centrándonos en el intervalo [0.7,0.8]. A continuación se muestran los resultados:

Algoritmo	bandwith	Tiempo (s)	Silhouette	Calinski-Harabasz	Número de clusters
meanshift	0.100	0.154	0.055	549.50486	366
meanshift	0.200	0.234	0.102	77.00944	340
meanshift	0.300	0.261	0.137	35.66736	304
meanshift	0.400	0.333	0.138	16.36725	216
meanshift	0.500	0.291	0.099	13.84278	123
meanshift	0.600	0.498	0.099	15.84389	61
meanshift	0.700	0.892	0.101	26.47205	14
meanshift	0.720	1.209	0.098	28.98231	11
meanshift	0.750	1.404	0.082	27.15215	9
meanshift	0.760	1.169	0.085	34.28803	6
meanshift	0.770	1.073	0.085	34.31722	6
meanshift	0.780	1.049	0.087	34.24051	6
meanshift	0.800	1.164	0.174	69.26129	3
meanshift	0.900	1.666			1
meanshift	1.000	1.477			1

Table 9: Resultados de Meanshift por parámetros

Vemos que el tiempo de ejecución no se mantiene constante, sino que va aumentando a medida que el parámetro bandwith se hace más grande. Esto parece ocurrir porque para tamaños del bandwith demasiado grandes, el algoritmo va desplazando los centros de los clusters hacia regiones más densas hasta que sólo queda un cluster. Sólo se genera un número de clusters óptimo para un bandwith entre 0.7 y 0.8. Vemos que el índice de Calinski-Harabasz en general es bajo excepto en el caso bandwith=0.1, pero aquí es alto porque el número de clusters es prácticamente igual al número total de objetos, por lo que no nos sirve. En los dos últimos casos no hay medidas porque sólo hay un cluster.

2.2.5 Estudio de Parámetros - Algoritmo K-means

Vamos a analizar el comportamiento de K-means según el número de clusters que genera, de 1 a 20. A continuación se muestran los resultados:

Algoritmo	clusters	Tiempo (s)	Silhouette	Calinski-Harabasz
kmeans	1.000	0.038	0.178	83.39922
kmeans	3.000	0.036	0.187	93.78387
kmeans	5.000	0.038	0.175	76.02407
kmeans	7.000	0.035	0.172	65.14288
kmeans	9.000	0.044	0.142	55.48638
kmeans	11.000	0.044	0.159	49.79095
kmeans	13.000	0.045	0.152	46.31021
kmeans	15.000	0.046	0.148	41.64218
kmeans	17.000	0.049	0.158	40.42823
kmeans	19.000	0.052	0.159	38.63540

Table 10: Resultados de kmeans por parámetros

Observamos que conforme el algoritmo produce más clusters, el índice de Calinski-Harabasz va disminuyendo su valor, por lo que las agrupaciones parecen ser cada vez peores.

2.2.6 Interpretación de la Segmentación

Como conclusión general del caso de estudio 2, podemos concluir que las variables más relevantes en este caso han sido la probabilidad de voto de los distintos partidos (primeros cuatro atributos), pues solamente con ellas se podrían distinguir todos los clusters.

A la vista de los resultados obtenidos y de los gráficos visualizados, podríamos concluir que los jóvenes están bastante interesados por los problemas de actualidad, ya que en casi todos los clusters la tendencia ha ido conforme a un nivel de preocupación alto. Además esto parece ser independiente del partido político con el que se simpatiza. Quizás hay pequeños detalles que destacar como que los simpatizantes de izquierda se interesan más por temas como el cambio climático mientras que los de derecha parecen más preocupados por los flujos migratorios.

2.3 Caso de estudio 3

2.3.1 Descripción del Caso de Estudio

En este tercer caso de estudio vamos a tomar los encuestados de sexo femenino. Tomaremos a las personas con el atributo sexo=1 (una vez normalizado), que corresponde a las mujeres. Las variables que se pretenden estudiar están relacionadas con la realidad económica de los ciudadanos de nuestro país. La finalidad es comprobar si hay alguna relación entre la clase social, economía y edad de las mujeres con su preocupación sobre la crisis de inflación que estamos sufriendo. He elegido al colectivo de las mujeres porque son las que suelen tener personas a su cargo (ya sea hijos o familiares mayores y/o enfermos), y por tanto son las que sufren más las consecuencias de las crisis económicas. Las variables seleccionadas para el análisis son las siguientes:

- Edad.
- Clase social.
- **p1**: Situación económica (si consigue ahorrar o si ha tenido que contraer deudas).

- **p7**: Ideología (del 0 al 10, qué tan de izquieras o de derechas se considera el enucestado).
- **p9_5**: Qué ha sentido a raíz de la crisis de inflación.

El conjunto de datos estudiado en el caso de estudio 1B está formado por **1065 individuos**.

2.3.2 Resultados de los algoritmos de Clustering

Estos son los resultados de ejecutar en python (en jupyter notebook) los cinco algoritmos seleccionados. El valor de k (número de clusters) para kmeans y single_linkage es de k=4. En meanshift se estime el valor de bandwidth (radio) dentro de la función, aunque hemos fijado el valor de quantile=0.05, es decir, hemos tomado el 5% de los datos para estimar el radio. Si no hicieramos esto, sólo se produce un cluster pues sale un bandwith demasiado grande. Para el algoritmo de birch, hemos fijado branching_factor=15 y threshold=0.1. Por último, para DBSCAN, hemos tomado minPts=15 y eps=0.126.

Algoritmo	Tiempo (s)	Calinski-Harabasz	Silhouette	Número de clusters
kmeans	0.065	249.521	0.19966	4
birch	0.100	200.236	0.14825	4
meanshift	5.008	174.685	0.20247	4
dbscan	0.008	3.431	-0.27459	3
single_linkage	0.008	159.761	0.20061	4

Table 11: Resultados de los algoritmos de clustering

Al igual que en los casos anteriores, todos los algoritmos han tenido tiempos de ejecución despreciables excepto Meanshift, debido a que tiene que calcular el bandwith. Por otro lado observamos que todos los algoritmos han producido un número de clusters óptimo, entre 3 y 4. Sin embargo, destacamos que DBSCAN tiene un índices de Calinski-Harabasz muy bajo por lo que tiene pinta de que el conjunto de datos no está bien distribuido. Hay que recalcar que Meanshift y Single Linkage han obtenido buenos índices CH, al contrario que pasó en los casos de estudio anteriores. Veamos la distribución dentro de sus clusters:

```
----- Ejecutando meanshift con radio=0.4119830186082422: 5.01 segundos,

Medidas


Calinski-Harabasz Index: 174.685, Silhouette Coefficient: 0.20247
Tamaño de cada cluster:
0:   613 (57.56%)
1:   254 (23.85%)
2:   163 (15.31%)
3:    35 ( 3.29%)
```

Figure 16: Resultados Meanshift

Vemos que el número de objetos sigue sin estar bien equilibrado pero al menos no se concentran todos en el mismo cluster como ocurría en casos anteriores.

```
----- Ejecutando Single Linkage Hierarchical Clustering: 0.01 segundos,

Medidas


Calinski-Harabasz Index: 159.761, Silhouette Coefficient: 0.20061
Tamaño de cada cluster:
0:   254 (23.85%)
1:   184 (17.28%)
2:   626 (58.78%)
3:     1 ( 0.09%)
```

Figure 17: Resultados Single Linkage

2.3.3 Visualización

Para el análisis vamos a tomar los resultados del algoritmo K-means, con k=4, pues ha obtenido los mejores resultados. Este algoritmo genera 4 clusters no demasiado descompensados entre sí, como podemos ver en la Figura 18.

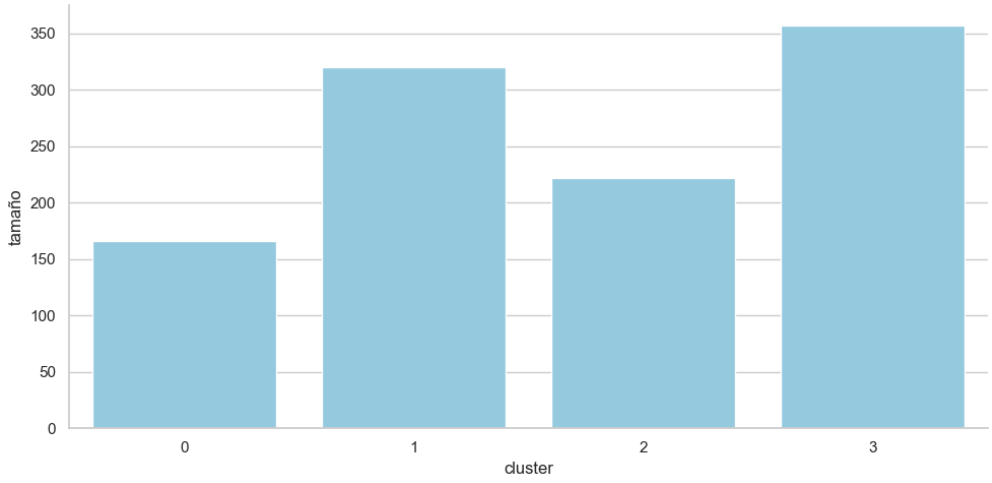

Figure 18: Gráfico de barras tamaño clusters

Seguimos con un Heatmap(19):

Lo primero que observamos es que no han salido atributos muy discriminantes, exceptuando el último (preocupación sobre la crisis de inflación). Vemos que la edad es bastante similar en los cuatro clusters, exceptuando quizás el segundo cluster. Por otro lado, en la clase social, el cluster 2 parece agrupar a las clases sociales más bajas mientras que el cluster 4 agrupa a las altas. Respecto a la situación económica, los clusters 1 y 4 recogen a los encuestados que han podido ahorrar. De la ideología no podemos decir nada pues no ha sido discriminante a la hora de agrupar. Por último, sobre el quinto atributo distinguimos dos grupos: el de gente que no ha sentido nada, representados por el cluster 1, y el de gente que ha sufrido enfermedades mentales a raíz de la crisis, representados en el cluster 3.

Podemos entonces hacer deducciones sobre qué grupos de gente agrupa cada uno de los cuatro clusters generados. El primer cluster podría contener a mujeres medianamente jóvenes,

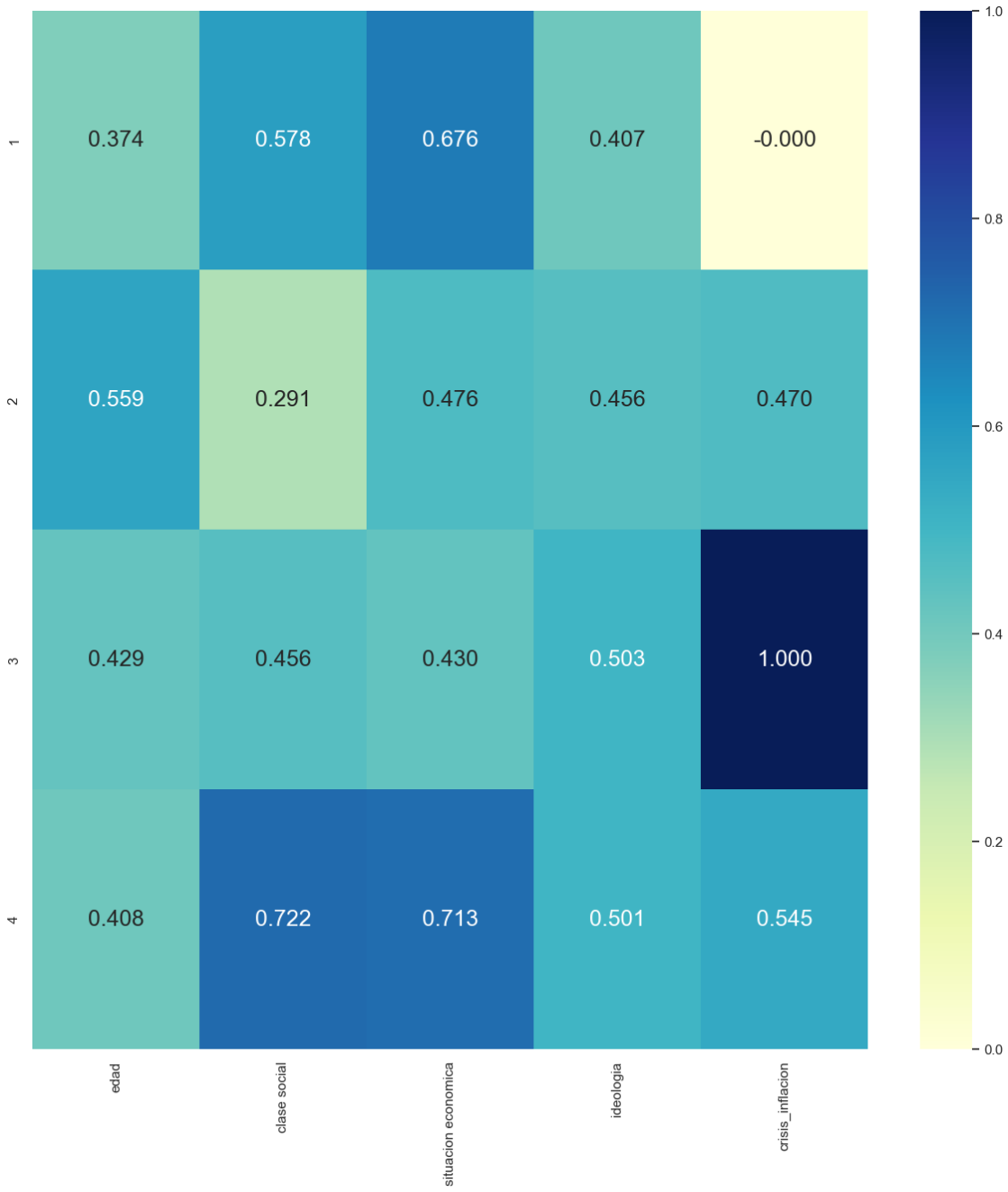

Figure 19: Heatmap para K-means

de clase social media/alta y sin dificultades económicas, que no se han visto afectadas por la crisis de inflación. El segundo cluster agruparía a mujeres un poco más mayores, de clase social baja aunque sin dificultades económicas, que han sentido desánimo y pesimismo ante la crisis. El tercer cluster contiene a mujeres de mediana edad, de clase media y situación económica normal, pero que han sufrido problemas de ansiedad y/o depresión a raíz de la inflación. El cuarto cluster contiene a mujeres también de mediana edad, de clase social alta y situación económica bastante buena, que se han visto preocupadas por la crisis pero tampoco les ha provocado problemas de salud.

A continuación, en la Figura 20 mostramos la Scatter Matrix:

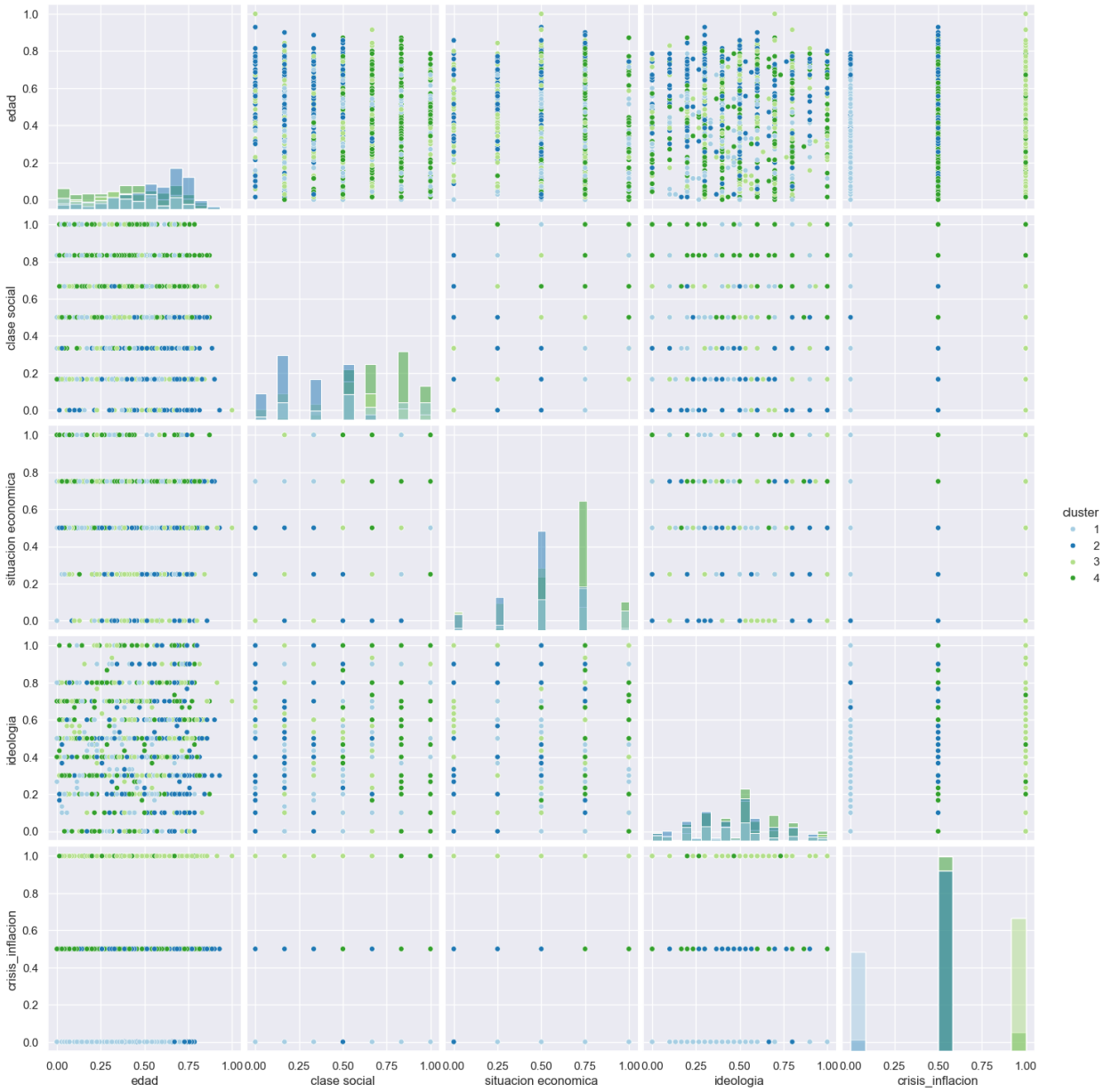

Figure 20: Scatter Matrix para K-means

Vemos que la dispersión es demasiado grande, por lo que no podemos obtener una dispersión lo suficientemente clara de los clusters. Llama la atención las graficas unidimensionales pues las variables son discretas y los datos se encuentras apilados en un diagrama de barras.

Por último, mostramos la distancia relativa entre centroides de los clusters, en la Figura 21. Podemos sacar poca información pues el gráfico no muestra nada relacionado con los diferentes atributos. Se aprecia cómo los clusters están muy separados y con tamaños (radios) similares, por lo que podríamos concluir que se ha realizado bien el agrupamiento.

En la tabla 2.3.3 se muestran qué variables son necesarias para identificar cada cluster:

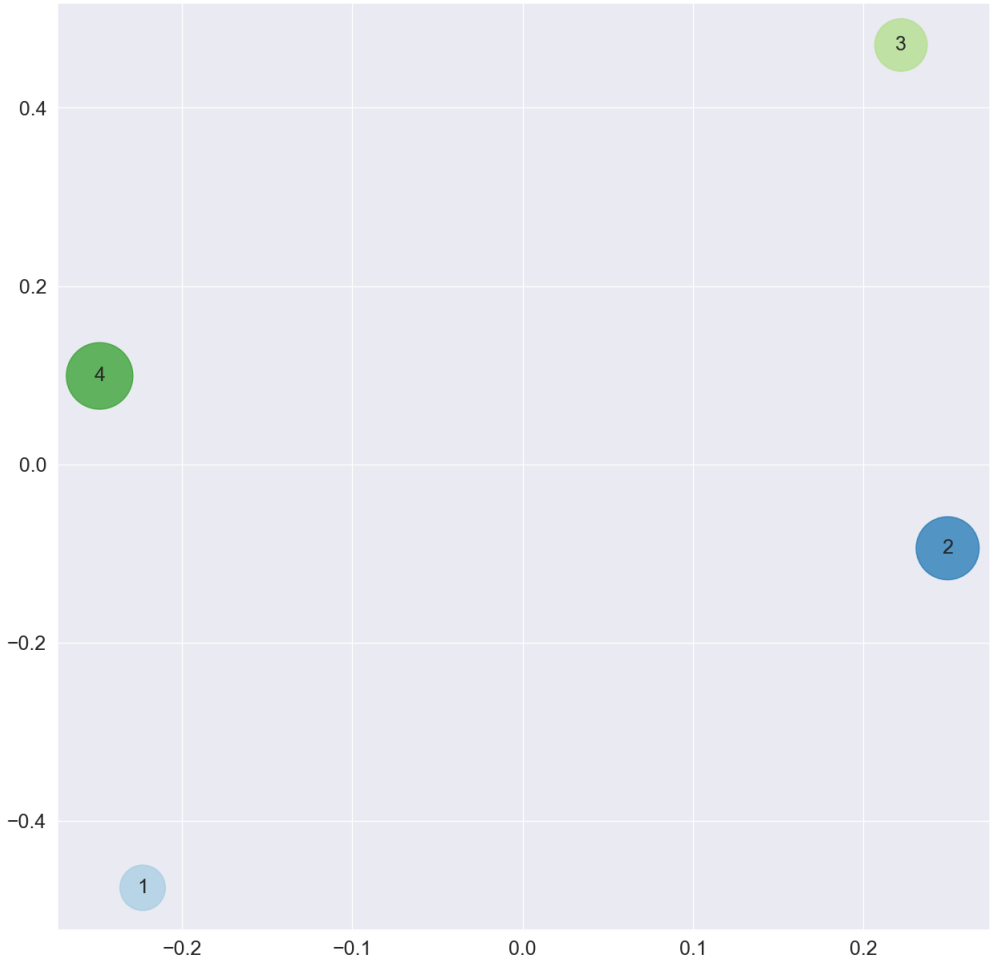

Figure 21: Distancia relativa entre centroides de clusters

Cluster	edad	clase social	situacion economica	ideologia	crisis inflacion
1		Medio			Bajo
2		Bajo			Medio
3		Medio			Alto
4		Alto			Medio

Table 12: Variables necesarias para identificar cada cluster

Con solo la clase social y la preocupación por la crisis de inflación conseguimos identificar los cuatro clusters.

2.3.4 Estudio de Parámetros- Algoritmo Single Linkage

Vamos a analizar el comportamiento de Single Linkage según sus dos parámetros. Hemos variado el número de clusters generados de 1 hasta 20. A continuación se muestran los resultados:

Algoritmo	clusters	Tiempo (s)	Silhouette	Calinski-Harabasz
single_linkage	1.000			159.76053
single_linkage	3.000	0.006	0.223	238.25949
single_linkage	5.000	0.006	0.190	120.72372
single_linkage	7.000	0.009	0.169	82.50078
single_linkage	9.000	0.008	0.116	63.43776
single_linkage	11.000	0.005	0.066	51.26989
single_linkage	13.000	0.008	0.060	43.31326
single_linkage	15.000	0.008	0.018	37.46040
single_linkage	17.000	0.009	-0.009	33.21919
single_linkage	19.000	0.005	-0.020	29.94979

Table 13: Resultados de single_linkage por parámetros

Observamos que conforme el algoritmo produce más clusters, el índice de Calinski-Harabasz va disminuyendo su valor, por lo que las agrupaciones parecen ser cada vez peores.

2.3.5 Estudio de Parámetros- Algoritmo K-means

Vamos a analizar el comportamiento de K-means según el número de clusters que genera, de 1 a 20. A continuación se muestran los resultados:

Algoritmo	clusters	Tiempo (s)	Silhouette	Calinski-Harabasz
kmeans	1.000			29.94979
kmeans	3.000	0.057	0.208	282.50488
kmeans	5.000	0.054	0.193	226.47724
kmeans	7.000	0.049	0.180	205.39721
kmeans	9.000	0.055	0.195	195.34570
kmeans	11.000	0.056	0.193	180.72764
kmeans	13.000	0.057	0.192	166.35552
kmeans	15.000	0.054	0.195	160.30665
kmeans	17.000	0.062	0.197	152.25439
kmeans	19.000	0.062	0.197	146.48689

Table 14: Resultados de kmeans por parámetros

Observamos que conforme el algoritmo produce más clusters, al principio el índice de Calinski-Harabasz aumenta hasta nº clusters=3, pero en seguida va disminuyendo su valor, por lo que las agrupaciones parecen ser cada vez peores.

2.3.6 Interpretación de la Segmentación

Estudiado el caso 3, podemos concluir de forma general que las variables más relevantes han sido la clase social y la pregunta sobre la crisis de inflación, pues sólo con ellas se podrían llegar a distinguir los 4 clusters. En vista de los resultados obtenidos y de los gráficos visualizados, no podríamos hacer grandes conclusiones pues al fin y al cabo el cluster más preocupado por la crisis de inflación es el constituido por mujeres de clase social "media"

y con una situación económica no muy mala, mientras que las mujeres de clase social más "baja" no parecen haberse visto demasiado afectadas.

3 Contenido Adicional

Cualquier tarea adicional a las descritas en este guion puede presentarse en esta sección.

4 Bibliografía

Referencias y material consultado para la realización de la práctica:

- Diapositivas de la asignatura en Prado.
- `https://docs.anaconda.com/free/anaconda/install/windows/`
- `http://scikit-learn.org/stable/modules/clustering.html`
- `http://www.learndatasci.com/k-means-clustering-algorithms-python-intro/`