Inteligencia de Negocio

Memoria Práctica 2

Segmentación para Análisis Empresarial

Nuria Rodríguez Barroso
Universidad de Granada
rbnuria6@gmail.com

Índice

1. Introducción

En esta memoria vamos a abordar un problema de aprendizaje no supervisado, clustering. Una compañía aseguradora quiere comprender mejor las dinámicas en accidentes de tráfico en España. Para ello, a partir de diversas variables que caracterizan el accidente, se pretende encontrar los grupos de accidentes similares y relaciones de causalidad que expliquen tipos y gravedad de los accidentes. Para ello, utilizaremos los datos publicados por la Dirección General de Tráfico (DGT) en [2], donde se recoge información de más de 30 variables entre los años 2008 y 2015. Para esta práctica, utilizaremos los datos del año 2013, por tanto utilizaremos una base que consiste en 89519 atributos con 32 atributos cada uno.

Realizaremos tres casos de estudio diferentes, donde reduciremos el problema original a subproblemas en búsquea de relaciones concretas: estudiaremos los accidentes en una determinada franja horaria, lugar, condiciones atmosféricas, etcétera.

En cada subproblema planteado aplicaremos cinco algoritmos de clustering, y estudiaremos el funcionamiento de cada uno de ellos. Posteriormente, seleccionaremos algunos de estos algoritmos y variaremos parámetros para analizar su efecto.

Para el estudio de todos los algoritmos utilizaremos dos medidas de error:

- **Silhouette**: es un método de interpretación y validación de la consistencia en un problema de clustering. Mide como de bien colocado está cada objeto en su clase. Este método devuelve un valor entre $[-1, 1]$, siendo una medida de como de similar es un objeto a los elementos de su clúster (cohesión) comparado con el resto de clústeres (separación). Entre el rango de valores que toma, los valores más altos indican que el objeto está bien colocado en su clúster y bien separado del resto de clústeres, mientras que un valor bajo o negativo puede significar que estemos considerando un número erróneo de clústeres en el algoritmo.
- **Calinski-Harabaz:** es un índice definido como la razón entre la dispersión interior de los clústeres y la dispersión entre los clústeres.

En cada caso de estudio vamos a aplicar cinco algoritmos, que elegiremos de entre los siguientes:

1. **K-Means**: Este algoritmo será obligatorio en todos los casos de estudio pues será tomado como algoritmo de referencia. Es un algoritmo de particionamiento iterativo en el que las instancias se van moviendo entre clústeres hasta que se alcanza el número de clústers deseado minimizando el error cuadrático entre clústeres. Necesita ser especificado el número de clústeres. Para su funcionamiento el método utiliza el concepto de centroide (media de los puntos de un mismo clúster) que no tiene por qué pertenecer al conjunto de muestra. El algoritmo terminará cuando el centroide más cercano a cada punto de la muestra sea el del clúster al que pertenece. El centroide se actualiza en cada iteración del algoritmo.

   Entre los parámetros más significativos del método en *scikit-learn* encontramos:

   - **n_clusters**: Número de clústeres a considerar (8 por defecto).
   - **init**: Método de inicialización, toma los siguientes valores:
     - *k_means++:* Opción por defecto. Selecciona los centroides de forma que se acelere la convegencia.
     - *random:* Elige los centroides aleatoriamente de entre los datos de la muestra.

    - *ndarray:* Para pasar como argumento los centroides.
  - **max_iter**: Máximo número de iteraciones en una misma ejecución, por defecto 300.
  - **n_jobs**: Número de "jobs" usadas para la computación. Si toma el valor *n_jobs = -1* se utilizarán todas las CPUs.

2. **Mini Batch K-Means**: Es una variación de K-means que usa mini-batches para reducir el tiempo de computación mientras que busca optimizar la misma función. Los mini-batches son subconjuntos de los datos de entrada aleatoriamente generados en cada iteración de entrenamiento. Estos subconjuntos disminuyen radicalmente el tiempo de computación requerido para converger a una solución local. A diferencia de otros algoritmos que mejoran el tiempo de cómputo de K-Means, por regla general Mini-Batch K-Means produce resultados poco peores que los que produce K-Means.

   Aunque los parámetros son muchos iguales a los que necesita el método K-Means, destacamos:

   - **batch-size:** Tamaño de los mini-batches, 100 por defecto.

3. **Mean Shift**: Este algoritmo busca encontrar máximos en función de densidad regular de las muestras. Es un algoritmo basado en centroides, que actualiza los candidatos a centroides de forma que sean la media entre los puntos pertenecientes a una misma región. Posteriormente, se eligen los centroides de entre los candidatos de forma que no haya dos centroides muy cercanos.

   El algoritmo fija automáticamente el número de clústeres, en lugar de depender del parámetro *bandwidth*. Parámetro que puede ser fijado manualmente o estimado con `estimate_bandwidth`, y dicta el tamaño de por la que buscar.

   Entre los parámetros más significativos del método en *scikit-learn* encontramos:

   - **bandwidth**: Si no se introduce el parámetro, se llama al método `estimate_bandwidth`.
   - **n_jobs**: Número de "jobs" usadas para la computación. Si toma el valor *n_jobs = -1* se utilizarán todas las CPUs.

4. **SpectralClustering:** Hace uso del radio espectral de la matriz de similitud de la muestra para reducir la dimensionalidad antes de aplicar clustering en dimensiones más bajas. A continuación, aplica K-Means en este espacio de dimensión reducida. Este algoritmo es especialmente eficiente si la matriz de similitud contiene muchos ceros. Tiene un buen funcoinamiento cuando utilizamos un númeor pequeño de clústeres y no se aconseja su uso para un alto númeor de estos.

   Entre los parámetros más significativos del método en *scikit-learn* encontramos:

   - **n_clusters**: Número de clústeres.
   - **eigen_solver:** Método de búsqueda de autovalores, toma los valores:
     - *arpack*
     - *obpcg*
     - *amg*
   - **n_init**: Número de veces que aplicamos el algoritmo de K-Means con diferentes centroides.
   - **affinity**: Puede tomar los valores:
     - *nearest_neighbors*

- *precomputed*
- *rbf*, por defecto.

- **n_neigbors**: Número de vecinos a utilizar para construir la matriz de similitud usando el método de vecino más cercano (ignorado cuando usamos affinity = "rbf").
- **assign_labels**: Elección del método a utilizar en el espacio de dimensión reducida. Puede tomar los siguientes valores:
  - *kmeans*, por defecto. Es sesible a la inicialización.
  - *discretize* Menos sensibilidad a la inicialización aleatoria.
- **n_jobs**: Número de "jobs" usadas para la computación. Si toma el valor *n_jobs = -1* se utilizarán todas las CPUs.

5. **AgglomerativeClustering**: es un método que implementa clustering jerárquico usando aproximación ascendete: cada observación comienza en el propio clúster y los clústeres se mezclan sucesivamente. El criterio de conexión determina la métrica usada para cada estrategia de mezcla:
   - **Ward**: minimiza la suma de las diferencias al cuadrado entre todos los clústeres. Al igual que K-Means minimiza la varianza.
   - **Maximun**: minimiza la máxima distancia entre pares de clústeres.
   - **Average**: minimiza la media de las distancias entre pares de clústeres.

   El tiempo de cómputo de este algoritmo es alto dado que es un algoritmo jerárquico.

   Entre los parámetros más significativos del método en *scikit-learn* encontramos:
   - **n_clusters**: Número de clústeres, por defecto 2.
   - **linkage**: Comentado anteriormente.
   - **affinity**: Elección de la métrica a utilizar.
     - *euclidean*, por defecto.
     - *l1*
     - *l2*
     - *manhattan*
     - *cosine*
     - *precomputed*
6. **DBSCAN**: Este algoritmo considera los clústeres como áreas de alta densidad separadas por áreas de baja densidad, de esta forma, las formas de los clústeres pueden ser de cualquier manera. A las muestras que están en áreas de alta densidad se les denomina *core samples*.

   Entre los parámetros más significativos del método en *scikit-learn* encontramos:
   - **eps**: Máxima distancia permitida entre dos muestras para poder pertenecer al mismo clúster.
   - **min_samples**: Número de muestras necesarias es un vecindario para poder considerar un punto como *core point*.

- **algorithm**: El algoritmo que utiliza el vecino más cercano.
  - *auto*, por defecto.
  - *ball_tree*
  - *kd_tree*
  - *rute*
- **metric**: Métrica usada para calcular la distancia entre instancias en un vector de características.
- **n_jobs**: Número de "jobs" usadas para la computación. Si toma el valor *n_jobs = -1* se utilizarán todas las CPUs.

7. **Birch:** Este algoritmo se utiliza para aplicar clustering jerárquico a conjuntos de datos especialmente grandes donde uno jerárquico tardaría demasiado.

   Entre los parámetros más significativos del método en *scikit-learn* encontramos:

   - **thershold**: Por defecto toma el valor de 0,5.
   - **branching_factor**: Maximum number of CF subclusters in each node.
   - **n_clusters**: Número de clústeres al final de una etapa de clustering.

Finalmente, complementaremos el análisis con representaciones gráficas así como nubes de puntos, dendogramas y mapas de temperatura. Realizaremos esto en los algoritmos de agrupamiento que mejor hayan funcionado para cada caso de estudio.

A lo largo de toda la práctica, para cualquier submuestra que obtengamos y para aquellos métodos de agrupamiento que necesiten fijar una simillar para su inicio, utilizaremos la semilla 123456 para obtener siempre los mismos resultados.

2. Caso de estudio 1

2.1. Descripción.

Para el primer caso de estudio nos planteamos estudiar los accidentes que se producen a la hora de salida del colegio y del trabajo (13-16h), así podremos obtener los principales factores de riesgo en ese tipo de accidentes. Para disminuir el número de muestras del problema consideramos el mismo problema en una provincia concreta, la provincia de Barcelona, obteniendo un caso de estudio con una base de datos de 4813 muestras.

En cuanto a la selección de características para el análisis vamos a utilizar:

1. TOT_MUERTOS
2. TOT_HERIDOS_GRAVES
3. TOT_HERIDOS_LEVES
4. TOT_VEHICULOS_IMPLICADOS

No escoges la variable TOT_VICTIMAS pues sería redundante teniendo en cuenta que estamos escogiendo todos los tipos de gravedad en las víctimas.

En la Figura 1 observamos el proceso de selección del caso de estudio.

```
#Definición del subcaso
def define_subset():
    accidentes = pd.read_csv('accidentes_2013.csv')

    #Seleccionamos aquellos accidentes que hayan tenido lugar en la provincia de Barcelona
    subset = accidentes.loc[accidentes['PROVINCIA'].str.contains("Barcelona")]
    #Seleccionamos aquellos accidentes que hayan tenido lugar en la franjahoraria [13-16]
    subset = subset.loc[(subset['HORA'] >= 13) & (subset['HORA'] <= 16)]

    #Seleccionamos aquellos atributos que queramos estudiar.
    usadas = ['TOT_MUERTOS', 'TOT_HERIDOS_GRAVES', 'TOT_HERIDOS_LEVES', 'TOT_VEHICULOS_IMPLICADOS']
    X = subset[usadas]

    return X
```

Figura 1: Definición del primer caso de estudio.

2.2. Análisis del caso de estudio.

Para el análisis de este caso de estudio hemos considerado hemos considerado 5 algoritmos de aprendizaje no supervisado:

- **K-Means** con parámetros $n\_clusters = 5$, $n\_init = 100$ y $init$= $k\text{-}means{+}{+}$.
- **MiniBatchKMeans** con parámetro $n\_clusters = 5$ y $init$= $k\text{-}means{+}{+}$.
- **MeanShift** con el parámetro de *bandwidth* estimado usando la función del mismo paquete *estimate_bandwidth* introduciendo como parámetro de esta $quantile = 0.3$.
- **Ward AgglomerativeClustering** con parámetro $n\_clusters = 100$.
- **Average AgglomerativeClustering** con parámetro $n\_clusters = 100$.

Y los resultados obtenidos para cada uno de estos algoritmos se pueden observar en la siguiente tabla

Algoritmo	Número de clústeres	Tiempo (s)	Calinski-Harabaz	Silhouette
K-Means	5	0.46118783950805664	14325.196567894258	0.87376403794886348
MiniBatchKMeans	5	0.019781112670898438	13655.820078553234	0.87118889590669935
MeanShift	41	2.4297609329223633	99168.154790557266	0.96971133580604363
Ward AgglomerativeClustering	100	0.5681591033935547	2.0605358979823882e+27	nan
Average AgglomerativeClustering	100	0.5090439319610596	2.0424922871714261e+27	nan

Tabla 1: Tabla comparativa

En cuanto a los comentarios de los resultados obtenidos en la tabla 1, podemos destacar que K-Means y MiniBatchKMeans han obtenido en ambas medidas del error resultados muy parecidos, sin embargo el tiempo ha marcado la diferencia, siendo el tiempo para K-Means del orden de casi 25 veces más alto. Esto se debe a lo ya comentado en la sección 1 acerca de MiniBatchKmeans, y es que no es más que un K-Means mejorado en eficiencia utilizando mini-batches para reducir el tiempo de computación. También se puede comprobar que aunque los resultados son muy parecidos, los de MiniBatchKMeans son en ambas medidas ligeramente peores que los de K-Means. En conclusión, podemos considerar MiniBatchKMeans una mejora de K-Means pues aunque pierda un poco de eficacia, la reducción del tiempo es bastante significativa pudiendo ser su uso crucial en problemas de más altas dimensiones.

En cuanto al resto de algoritmos en comparación con nuestro algoritmo de referencia, K-means, obtenemos que MeanShift ha obtenido mejores resultados tanto para el índice de Calinski-Harabaz, obteniendo un resultado 7 veces mejor y alcanzando un resultado bastante cercano a 1 en el coeficiente Silhouette, lo que nos indica que hemos obtenido un resultado muy bueno de agrupación. Destacar que estos resultados se han obtenido utilizando un total de 41 clústeres, número muy superior a los 5 clústeres fijados para las dos versiones del K-means, por tanto sería interesante probar a aplicar K-Means más clústeres para ver si el resultado obtenido es más competitivo. Realizaremos este estudio en la parte de configuración de parámetros.

En cuanto a los algoritmos jerárquicos, estos merecen una mención aparte. Los comentarios que podemos realizar son comunes a ambos algoritmos y es que en ambos casos el índice de Calinski-Harabaz es "infinito" y el coeficiente Silhouette desconocido. Esto se debe en ambos casos a que al haber considerado un número tan alto de clústeres habrá varios de ellos que estén compuestos por un solo punto, siendo así la distancia a su propio clúster 0 y produciendo problemas en las medidas consideradas. Para el coeficiente de Silhouette devuelve nan pues la función *np.maximun* utilizada en la definición de esta medida del error devuelve nan en caso de que uno de los dos elementos a comparar sea nan. Estudiaremos esta cuestión más adelante.

2.3. Configuración de parámetros.

En esta sección vamos a realizar diferentes configuraciones de algunos de los algoritmos implementados en la sección anterior.

2.3.1. K-Means

Como ya hemos comentado en la sección anterior, la superioridad de MeanShift con respecto a K-Means se podría deber al número de clústeres elegido, y es que esto es una debilidad de este método. Al tener

que elegir el número de clústeres de antemano, limitamos la potencialidad del método pues puede ser que el número introducido esté muy alejado de la optimalidad de solución.

Para estudiar esta hipótesis, vamos a analizar este algoritmo de agrupamiento para valores de *n_clusters* de 5 (caso actual), 10, 20, 30, 40 y 50. Obteniendo la siguiente tabla de resultados:

Como podemos observar en la tabla 2, se cumple lo que advertíamos y es que el resultado del K-Means para este caso de estudio mejora según el número de clústeres considerado. Ahora bien, igual que aumentan tanto el índice de Calinski-Harabaz como el coeficiente Silhouette, también aumenta el tiempo de ejecución del algoritmo. Esto sería algo a tener en un problema de un tamaño suficiente como para que el tiempo nos causara problemas. De todas formas, el algoritmo más lento no ha tardado ni 4 s por lo que en nuestro caso no es un problema.

Número de clústeres	**Tiempo (s)**	**Calinski-Harabaz**	**Silhouette**
5	0.46118783950805664	14325.196567894258	0.87376403794886348
10	0.7016699314117432	29760.102243855948	0.95028889289950336
20	1.198307752609253	71827.688156602308	0.97464386143968473
30	1.8935821056365967	132264.15992572455	0.98377760993202834
40	2.817133903503418	402030.02971162566	0.98812002641862529
50	3.797278881072998	2463014.6241049198	0.99322583674819376

Tabla 2: Tabla comparativa

Para estudiar de forma más clara las mejoras producidas, vamos a representar en una gráfica los valores de la tabla normalizados (con min-max) de forma que podamos aprenciar el crecimiento de forma globlal.

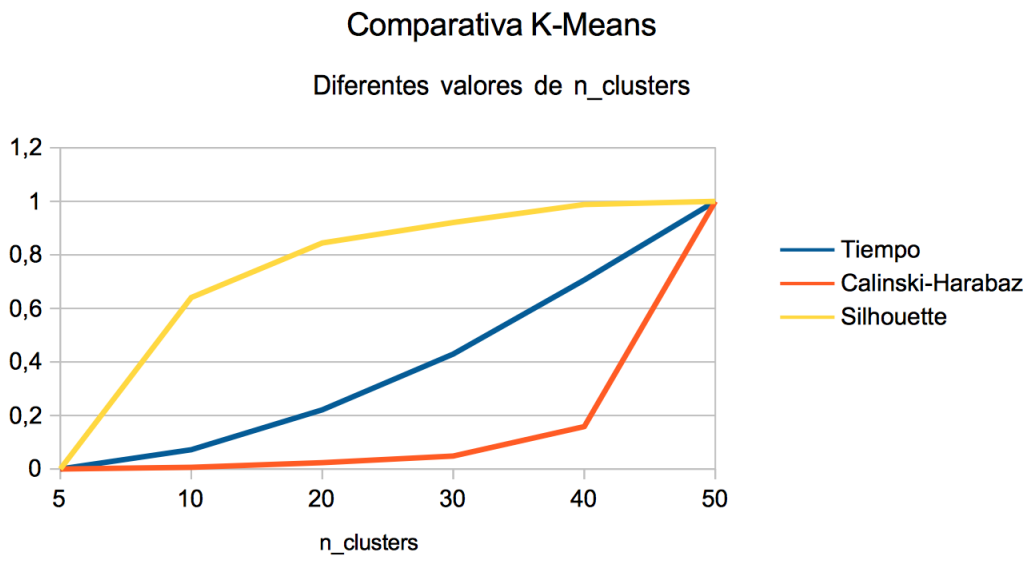

Figura 2: Comparativa KMeans

Como ya sabíamos, a la vez que aumentamos el número de clústeres en el algoritmo de K-Means,

aumentan el resto de medidas, pero en la figura 2 observamos que la manera en la que crecen estos valores no es igual. Observamos que el tiempo sigue un crecimiento prácticamente lineal mientras que coeficiente Silhouette crece rápidamente al principio quedando casi estancada al final y al contrario le ocurre al índice Calinski-Harabaz. Esta diferencia de medidas se puede deber a las características de las medidas del error utilizadas y es que el índice de Calinski-Harabaz se ve afectado por el número de clúster considerado.

Aunque en este problema podríamos escoger cualquier número de clústeres debido a los cortos tiempos de ejecución, tener en cuenta esta gráfica podría ser crucial si por ejemplo buscáramos obtener un buen resultado del coeficiente Silhouette minimizando el tiempo de ejecución necesario, podríamos quedarnos con $n\_clusters = 20$, disminuyendo bastante el tiempo de ejecución y obteniendo valores prácticamente iguales a los que obtendríamos con 50 clústeres. Además, también tener presente el problema que estamos abordando, que es un problema de agrupamiento, y es que obtener una solución de más de 20 clústeres puede ser imposible de interpretar hasta por un especialista. Por tanto, aunque aumentemos el número de clústeres para comprobar que KMeans obtiene también resultados, no tendría sentido elegir una solución con 50 clústeres.

2.3.2. Algoritmos jerárquicos.

Dado el punto de partida de estos algoritmos en la clasificación general para este caso de estudio, comprendemos que hay varias mejoras que podemos aplicar. Como ya habíamos introducido anteriormente, los extraños resultados de las medidas de error para estos algoritmos se deben a que existen varios clústeres con un solo elemento. Por ello, de manera **adicional** vamos a realizar un preprocesado de los datos eliminando los outliers antes de realizar las diferentes configuraciones de este algoritmo.

Preprocesado de datos para el jerárquico (outliers)

En principio no debería ser necesario que hubiera clústeres con un solo elemento dado que estamos considerando 100 clústeres en un conjunto de casi 5000 muestras. Por tanto, ese hecho nos está indicando que el conjunto de datos contiene anomalías o outliers, hecho que debemos solucionar pues los algoritmos jerárquicos son muy sensibles a este tipo de datos "especiales" por su funcionamiento. Crearían el clúster que solo contuviera a este dato y nunca se fusionaría con ningún otro al estar muy alejado de estos.

Vamos a realizar este análisis para uno de los dos algoritmos jerárquicos pues el proceso es bastante similar, escogeremos el algoritmo **Ward AgglomerativeClustering**.

En primer lugar, debemos obtener cuántos clústeres de los 100 generados contienen un único dato o pocos datos, para identificar a estos datos como valores perdidos. Vamos a considerar que todo clúster con menos de 2 datos es un clúster de valores perdidos. Entonces, aplicamos el método que observamos en la figura 3 obteniendo un nuevo conjunto de datos sobre el que aplicaremos el algoritmo jerárquico.

```python
def delete_outliers(prediction, subset):
    X = subset
    k = general.numero_clusters(prediction)
    #se convierte la asignación de clusters a DataFrame
    clusters = pd.DataFrame(prediction[1],index=X.index,columns=['cluster'])
    #y se añade como columna a X
    X_cluster = pd.concat([X, clusters], axis=1)

    #Filtro quitando los elementos (outliers) que caen en clusters muy pequeños en el jerárquico
    min_size = 2
    X_filtrado = X_cluster[X_cluster.groupby('cluster').cluster.transform(len) > min_size]
    k_filtrado = len(set(X_filtrado['cluster']))
    print('''De los {:.0f} clusters hay {:.0f} con más de {:.0f} elementos.
          Del total de {:.0f} elementos, se seleccionan {:.0f}'''.format(k,k_filtrado,min_size,len(X),len(X_filtrado)))
    X_filtrado = X_filtrado.drop('cluster', 1)

    X_filtrado_normal = preprocessing.normalize(X_filtrado, norm='l2')

    return X_filtrado_normal
```

Figura 3: Código de elimiación de outliers.

En cuanto al código, comentar la importancia de volver a normalizar los datos, pues aunque ya estuvieran normalizados, estos datos que hemos eliminado (outliers) obtendrán valores muy alejados del rango normal de los atributos para algunos atributos. Por tanto, la normalización de estos atributos se vería grandemente afectada por estos datos. Realizando de nuevo una normalización solucionamos este problema.

Cuando ejecutamos la función de la figura 3, obtenemos que solo 29 de los 100 clústeres considerados contienen más de 2 elementos. Por lo que se confirma la gran presencia de outliers. El conjunto que nos queda tras aplicar este preprocesado consta de 4738 muestras (figura 4).

```
De los 100 clusters hay 29 con más de 2 elementos.
          Del total de 4813 elementos, se seleccionan 4738
```

Figura 4: Localización de outliers

Para comprobar si el resto de datos que si hemos considerado contienen outliers, repetimos el proceso con unos pocos de clústeres más de los que contenían a más de dos datos en la vez anterior (35), obteniendo en siguiente resultado:

```
De los 35 clusters hay 33 con más de 2 elementos.
          Del total de 4738 elementos, se seleccionan 4735
```

Figura 5: Resultado del segundo filtrado.

En este caso se eliminan pocos datos luego nos vamos a quedar con este conjunto de datos para realizar las diferentes configuraciones para los jerárquicos.

Una vez eliminadoslos outliers, vamos a probar los algoritmos jerárquicos para diferente número de clústeres para realizar, de maneara análoga al apartado anterior un análisis de los resultados. Consideramos los resultados para $n\_clusters = 5, 10, 15, 20, 25$. Obteniendo los siguientes resultados:

A rasgos generales, observamos que en la tabla 3 se obtienen muy buenos resultados para estos métodos jerárquicos. Tanto el valor del índice de Calinski-Harabaz como el del coeficiente Silhouette aumentan en función del número de clústeres considerados.

Algoritmo	Número de clústeres	Tiempo (s)	Calinski-Harabaz	Silhouette
Ward-5	5	0.22140002250671387	15195.02699290252	0.88578754522881009
Average-5	5	0.18479204177856445	8263.1845068171369	0.84562554908019727
Ward-10	10	0.22159481048583984	50020.397827329856	0.96339692656141185
Average-10	10	0.18349575996398926	16191.165194881463	0.90673734525512784
Ward-15	15	0.22305798530578613	161704.64026793803	0.97781537279182673
Average-15	15	0.17682099342346191	32818.760749133406	0.93201660268369158
Ward-20	20	0.22266578674316406	589805.48702640447	0.98980373428693247
Average-20	20	0.1913747787475586	589805.48702640459	0.98980373428693247
Ward-25	25	0.2427690029144287	6979797.5037246253	0.99792691350330431
Average-25	25	0.2026817798614502	6979797.5037246263	0.99792691350330431

Tabla 3: Tabla comparativa

En cuanto al tiempo de ejecución, notar que ambos algoritmos han sido bastante rápidos en todas las configuraciones de los mismos.

Finalmente, comparando ambos algoritmos obtenemos que, por regla general, Ward ha obtenido mejores resultados que Average, notándose más la diferencia en configuraciones con menos clústeres y siendo ambos prácticamente iguales en ambas medidas con un número alto de estos. Por este motivo, a partir de ahora consideraremos el algoritmo Ward cuando queramos aplicar un jerárquico.

A modo de conclusión de esta subsección, notar que la definición inicial de los algoritmos jerárquicos con un número alto de clústeres nos ha hecho darnos cuenta de la presencia de outliers en la muestra. Sin embargo, como estamos comparando ahora, estos algoritmos no precisan de tal número de clústeres para su buen funcionamiento, de hecho con menos clústeres que los anteriores han obtenido resultados incluso superiores. Esto es importante dada la importancia de la interpretabilidad en este tipo de problemas.

2.4. Interpretación de la segmentación.

Para la interpretación de la segmentación, vamos a utilizar solo algunos de los algoritmos implementados dado que ocuparía demasiado tiempo ocuparnos de todos ellos. Elegiremos aquellos que hayan funcionado bien además de aquellos que no hayan necesitado un alto número de clústeres pues esto dificultaría su interpretación.

2.4.1. Primera interpretación.

Vamos a realizar la interpretación del primer algoritmo K-Means implementado, que tenía 5 clústeres. Aunque como ya hemos visto anteriormente es el peor K-Means implementado el resultado no es malo y es más fácil su interpretación. Apoyaremos nuestro análisis en la scatter-matrix producida, que podemos observar en la figura 6.

Por la naturaleza de esta representación, obtenemos una matriz simétrica donde tenemos representadas, tanto la relación entre atributos como el valor de cada una de las variables. Así, podremos observar tanto tendencias de variables a comportarse interrelacionadas con otras como de clústeres a tomar determinado número de valores para determinados atributos. Además, en la diagonal podemos apreciar la cantidad de elementos que pertenecen a cada uno de los clústeres.

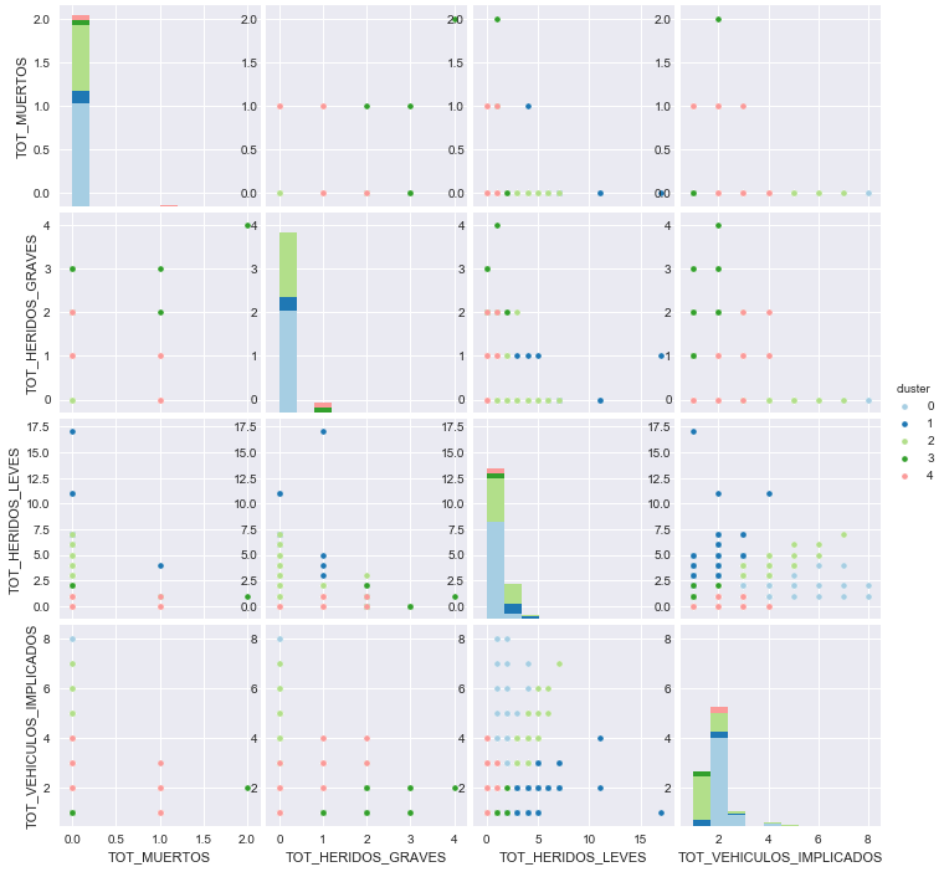

Figura 6: Scatter Matrix para K-Means

Haciendo uso de esta representación, vamos a estudiar qué características caracterizan a cada uno de los clústeres generados. A rasgos generales, vamos a utilizar más la última fila dado que esta representa las relaciones entre los vehículos implicados y las diferentes víctimas, produciendo así relaciones más interesantes.

1. **Clúster 0:** Por regla general, el clúster 0 se caracteriza por un alto número de vehículos implicados pero con un bajo número de víctimas, siendo 0 en el caso de muertos y heridos graves y un bajo número en el caso de heridos leves.
2. **Clúster 1:** Este clúster se caracteriza por haber involucrado relativamente pocos vehículos [0-4] pero de muchos heridos leves [3, +15].
3. **Clúster 2:** Este clúster recoge las muestras de aquellos accidentes en los que ha habido un número consdierable de vehículos implicados, pero no tantos como los que recoge el clúster 0 y en los que ha habido pocas víctimas leves [0,6] y ningún muerto ni herido grave.
4. **Clúster 3:** Este clúster se caracteriza de forma general por incluir aquellas muestras de accidentes en los que los vehículos implicados han sido pocos,no habido ningún muerto, muchos heridos graves y muy pocos leves.
5. **Clúster 4:** En este clúster se encuentran aquellos accidentes en los que el número de vehículos

implicados ha sido bajo-medio [0,4], habiendo 0 o 1 muertos, menos de 2 heridos leves y 0 o 1 heridos leves.

Así, a rasgos generales hemos podido agrupar los accidentes de tráfico que se producen en la provincia de Barcelona durante las horas de salida de trabajo y colegio en 5 tipos. Ahora, a partir de esta información podemos realizar una inferencia sobre cuáles pueden ser las causas de cada tipo de accidente, por ejemplo:

1. Muchos vehículos implicados y víctimas principalmente leves, seguramente un golpe producido en zonas urbanas con mucho tráfico.
2. Pocos vehículos pero muchos heridos leves, seguramente incluyendo peatones, luego seguramente teniendo lugar en zonas de mucha concurrencia de pesonas en estas horas, centro o salida de colegios.
3. Muchos vehículos involucrados y pocas víctimas, todas leves. Igualmente puede indicar que se ha producido en zona urbana, provocado por el tráfico.
4. Parecido al segundo caso, pero incluyendo a más heridos graves. Por lo que puede deberse también a accidentes en zona urbana pero de mayor gravedad.
5. Dado el bajo número de víctimas podemos deducir que son solo las que estaban en los coches (por regla general), luego se deberán la mayoría a accidentes fuera de la zona urbana.

A rasgos generales, obtenemos que casi todos los accidentes tienen características de haberse producido en zona urbana y hay varios accidentes con un número alto de heridos graves. Además, los clústeres con características más determinantes para este tipo de accidentes son los más numerosos, clúster 0 y 1 (lo observamos en la diagonal). Por tanto, confirmamos que existe el peligro en las zonas urbanas de producirse accidentes leves en esas horas del día.

2.4.2. Segunda interpretación.

Para esta segunda interpretación, vamos a considerar algún algoritmo genético para así poder apoyar nuestro análisis en un heatmap y un dendograma.

De forma análoga a la interpretación anterior, buscamos un algoritmo que nos haya proporcionado buenos resultados pero con un número de clústeres bajo. Por ello, vamos a utilizar el Ward AgglomerativeClustering definido en la parte de configuración de parámetros de los algoritmos genéticos.

De igual forma, sacaremos características a partir de la Scatter Matrix producida, en este caso representada en la figura 7.

De forma análoga al caso anterior, vamos a obtener las caracterizaciones de cada clúster:

1. **Clúster 0:** Se caracteriza por aquellos accidentes donde el número de vehículos implicado ha sido alto y no ha habido ningún herido grave.
2. **Clúster 1:** Se caracteriza por accidentes donde el número de vehículos implicados ha sido bajo, en las que solo ha habido víctimas leves pero siendo estas numerosas.
3. **Clúster 2:** Se caracteriza por aquellos accidentes donde el número de vehículos implicados ha sido relativamente bajo y el número de heridos graves ha sido muy alto.

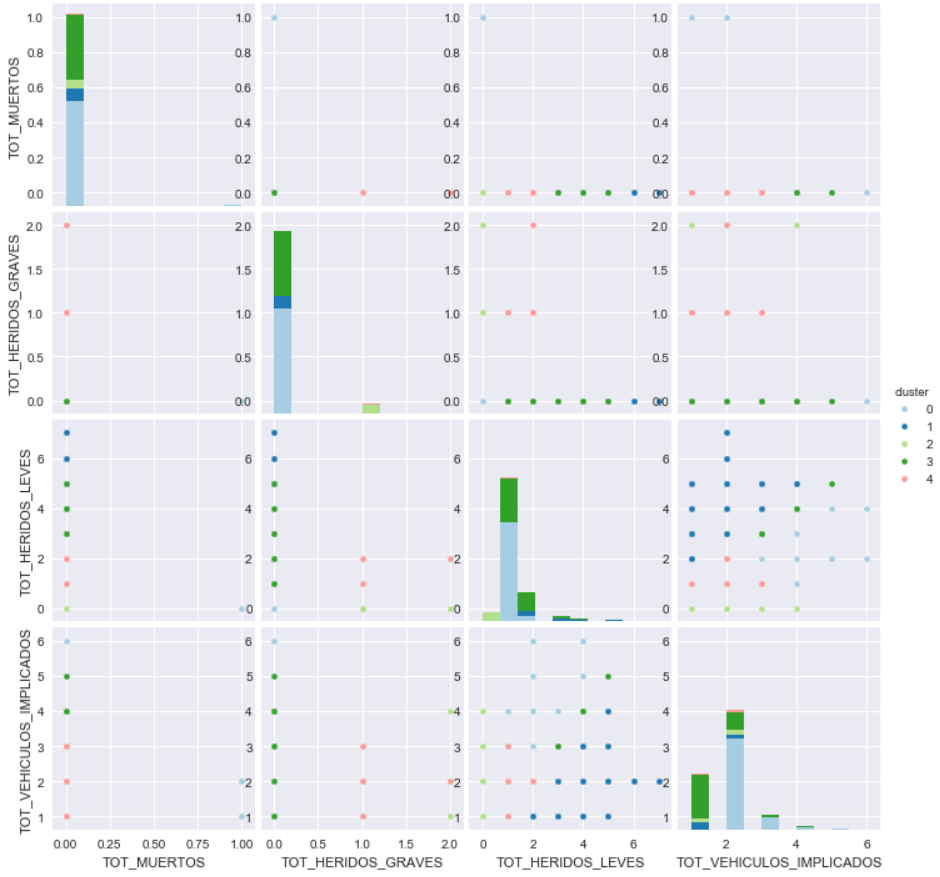

Figura 7: Scatter Matrix para Ward-5

4. **Clúster 3:** Se compone de los accidenes en los cuales se han involucrado 4-5 vehículos y donde solo ha habido heridos leves.

5. **Clúster 4:** Se caracteriza por aquellos accidentes en los que se han involucrado 1-3 vechículos y no ha habido muertos, ha habido un herido grave y 1,2 leves.

En cuanto a la relación entre las variables consideradas, en este caso observamos en que en la mayoría de los clústeres hay una tendencia linea entre el número de vehículos implicados y el número de heridos leves. Así, ambos crecen de forma proporcional en cada uno de los clústeres. Es natural pensar en esta correspondencia lineal pues cuanto más vehículos implicados hayan, más personas puede ser propensar a sufrir daños. En cuanto a esta relación con los heridos graves y los muertos no se verifica, dado que es más raro que esto ocurra.

Para apoyar nuestro análisis, vamos a utilizar un dendograma junto con un heatmat. Para ello utilizamos el paquete *seaborn* que nos ofrece ambas representaciones en una. Estas representaciones tienen sentido para un número pequeño de muestras. Por eso, para este caso nos vamos a quedar con un conjunto de 500 muestras que sacamos del conjunto previamente filtrado que estamos considerando hasta ahora. Utilizamos el método *ward* pues ha sido el que ha demostrado proporcionar mejores resultados en este caso de estudio concreto.

A rasgos generales, podemos comprobar que en la mayoría de los clústeres generados hay muy pocos

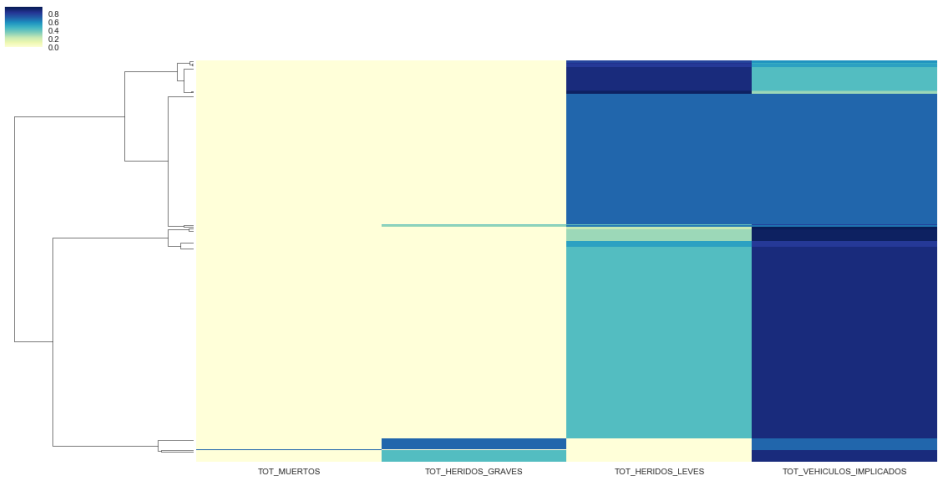

Figura 8: Dendograma y Heatmap obtenidos.

muertos y muy pocos heridos graves (solo en dos de ellos hay más heridos graves). En cuanto a los heridos leves, en los mismos clústeres que había muchos heridos graves, hay pocos leves, concentrándose el resto de heridos leves en el resto de clústeres. En cuanto a los vehículos implicados, es por regla general muy alto.

Entrando un poco más en detalle, si observamos que el clúster más grande es aquel en el que hay muchos vehículos implicados, bastantes heridos leves y muy pocos o ningún herido grave y muertos. Además, el segundo clúster más grande recoge los accidentes que han implicado un menor número de vechículos pero que han implicado más heridos leves. Entre estos dos clústeres se recoge a gran parte de la muestra considerada.

Por tanto, según esta interpretación llegamos a la misma conclusión que en la interpretación anterior y es que por el tipo de accidente, muchos vehículos involucrados y muchos heridos leves, seguramente se traten de accidentes ocurridos en zona urbana. Por tanto, hemos conseguido demostrar lo que buscábamos y es el alto peligro de la zona urbana en la franja horaria de salida del colegio y del trabajo dada la cantidad de peatones que hay a esa hora por la calle y al tráfico que los trabajadores producen.

3. Caso de estudio 2

3.1. Descripción.

En este caso de estudio vamos a analizar un problema más general, los accidentes en los que se ha producido una colisión de vehículo en marcha, ya sea fronta, lateral, etcétera. Esperamos encontrar relaciones entre los tipos de víctimas y el número de vehículos involucrados. En este caso, no esperamos que haya tanta frecuencia de "accidentes urbanos" como hemos deducido en el caso anterior.

Si realizamos este filtro obtenemos un total de 49280 muestras, que es una complejidad computacional en tiempo y memoria inabordable por algunos algoritmos. Por tanto, seguimos una estrategia parecida al caso anterior estudiando solo aquellos accidentes que se hayan producido en la Comunidad Autónoma de Cataluña. Así reducimos el problema a 14273 muestras.

De forma análoga al caso de estudio anterior, utilizamos las variables:

- TOT_MUERTOS
- TOT_HERIDOS_GRAVES
- TOT_HERIDOS_LEVES
- TOT_VEHICULOS_IMPLICADOS

En la figura 9 observamos el proceso de selección del caso de estudio:

```
#Definición del subcaso
def define_subset():
    accidentes = pd.read_csv('accidentes_2013.csv')


    subset = accidentes.loc[accidentes['TIPO_ACCIDENTE'].str.contains("Colisión de vehículos en marcha")]
    subset = subset.loc[subset['COMUNIDAD_AUTONOMA'].str.contains("Cataluña")]

    usadas = ['TOT_MUERTOS', 'TOT_HERIDOS_GRAVES', 'TOT_HERIDOS_LEVES', 'TOT_VEHICULOS_IMPLICADOS']
    X = subset[usadas]

    return X
```

Figura 9: Definición del segundo caso de estudio.

3.2. Análisis del caso de estudio.

Para el análisis de este caso de estudio hemos considerado 5 algoritmos de aprendizaje no supervisado:

- **K-Means** con parámetros $n\_clusters = 5$, $n\_init = 100$ y $init = kmeans++$.
- **MeanShift** con parámetro de *bandwidth* estimado usando la función del mismo paquete *estimate_bandwidth* introduciendo como parámetro de esta $quantile = 0.3$
- **DBSCAN** con el parámetro $eps = 0.1$
- **Birch** con parámetros $n\_clusters = 5$ y $thershold = 0.1$. Utilizamos este algoritmo pues se aplica cuando queremos utilizar un jerárquico en un conjunto de datos grande y este es el subcaso más grande que vamos a considerar.

- **SpectralClustering** con parámetros *n_clusters* = 5 y *affinity* = 'rfb'.

Los resultados obtenidos para cada uno de los algoritmos en la tabla 4

Algoritmo	Número de clústeres	Tiempo (s)	Calinski-Harabaz	Silhouette
K-Means	5	0.7914528846740723	36154.186827454883	0.86718768508397859
MeanShift	41	0.36252379417419434	20174.548726417921	0.9040218422644386
DBSCAN	19	1.9177911281585693	429.47360448959989	0.29334981735450749
Birch	5	0.5069057941436768	17121.457021867674	0.82431192856413005
SpectralClustering	5	67.81920003890991	23500.393871329426	0.81418608577542995

Tabla 4: Tabla comparativa

A grandes rasgos en la tabla 4 podemos observar que todos los algoritmos se comportan de forma relativamente buena a excepción de DBSCAN. Esto se puede deber a que el conjunto de partida presenta grandes diferencia entre las densidades (DBSCAN es un método basado en densidades como hemos explicado en 1) o a que el parámetro pasado como argumento no es el adecuado. Trataremos esto en la configuración de los algoritmos.

En cuanto a los tiempos de ejecución, observamos que SpectralClustering consume muchísimo más tiempo de ejecución y sin embargo es el que peor coeficiente Silhouette consigue después de DBSCAN. El gran tiempo de cómputo se debe al manejo de la matriz de similitud en búsqueda de los valores propios o autovalores. El motivo de que el índice de Calinski-Harabaz sea más alto que en la mayoría del resto de algoritmos es que este índice funciona especialmente bien en conjuntos convexos, en aquellos algoritmos basados en distancias.

Podemos ver esto último reflejado en el valor del índice de Calinski-Harabaz para el K-Means, que el valor más alto de los cinco algoritmos utilizados a pesar de tener un coeficiente Silhouette algo más baja que el MeanShift. También en la línea de estos comentarios, podemos notar que DBSCAN tiene un índice de Calinski-Harabaz muy bajo, que se debe en parte al mal funcionamiento del agrupador y a la naturaleza del método, al ser basado en densidades perjudica al índice.

Tras este vistazo rápido a los primeros resultados se nos ocurren algunas posibles mejoras según un cambio de parámetros que vamos a considerar en la siguiente subsección.

3.3. Configuración de parámetros.

En esta sección vamos a realizar diferentes configuraciones de algunos de los algoritmos implementados en la sección anterior.

3.3.1. DBSCAN

Como ya hemos comentado, el algoritmo DBSCAN nos proporciona resultados peores a los esperados, por tanto, vamos a comprobar si esto se puede solucionar haciendo uso de alguno de sus parámetros. Para estas comprobaciones vamos a modificar:

1. *eps*, que es la máxima distancia permitida entre dos muestras para pertenecer al mismo clúster. Vamos a disminuir este valor ya que hemos visto que con 0.1 no obtenemos resultados demasiado buenos. Vamos a considerar *eps* = 0.01, 0.05, 0.1 (ya utilizado).

Algoritmo	Número de clústeres	Tiempo (s)	Calinski-Harabaz	Silhouette
DBSCAN eps=0.01	36	1.8060221672058105	10386.48323116352	0.9848512797113822
DBSCAN eps=0.05	26	1.5406363010406494	21577.083098471317	0.94758571940138336
DBSCAN eps=0.1	19	1.553555965423584	429.47360448959989	0.29334981735450749

Tabla 5: Tabla comparativa

Observamos en la tabla 5 que lo que pensábamos era cierto, el algoritmo implementado en principio para DBSCAN con valor de $eps = 0.1$, permitía que pertenecieran al mismo clúster dos instancias que estaban demasiado separads. El valor de *eps* no se puede estimar a priori dado que depende de cada problema y conjunto de muestra, por eso es necesario realizar esta configuración de los parámetros.

Ahora bien, en el coeficiente Silhouette observamos la clara mejoría a la hora de disminuir la distancia máxima entre dos instancias de una misma clase y el tiempo es muy parecido en todos los casos, el problema viene cuando nos fijamos en el índice de Calinski-Harabaz. En ambos resultados obtenemos que son valores bajos relacionados por ejemplo con el valor para el K-Means que observamos en la tabla 4, se debe a que aunque hayamos mejorado el método, sigue siendo un método basado en densidades por lo que se ve afectado en este sentido. Además, observamos que aunque en ambos casos aumenta con respecto al original, lo hace de forma irregular pues es mejor en el segundo algoritmo utilizado, esto se debe a que el índice de Calinski-Harabaraz se ve afectado negativamente por el aumento del número de clústeres, según la fórmula que introducíamos en la Sección 1, y para $eps = 0.01$ se han generado 10 clústeres más de los que se necesitaban para un valor de $eps = 0.05$.

Para reforzar nuestros argumentos de forma visual, normalizamos el valor de las medidas de error con una normalización min-max y obtenemos el siguiente gráfico:

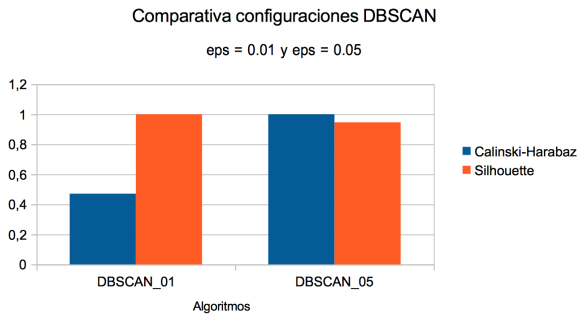

Figura 10: Comparación configuraciones del DBSCAN.

En la figura 10 , observamos más claramente la proporción de mejora del Calinski-Harabaz en la segunda configuración frente a la proporción de mejora del valor de Silhouette en la primera configuración.

En conclusión, dependerá del problema concreto y de nuestros intereses, pero quizás en este caso particular sería mejor quedarse con la opción de **DBSCAN eps = 0.05** pues el valor del coeficiente Silhouette es un valor bastante competitivo también pero el índice de Calinski-Harabaz es claramente superior. Sin embargo, para esta configuración el número de clústeres generado ha sido 26, un número demasiado alto para un problema de clustering. A no ser que los expertos sean capaces de deducir conclusiones de un problema en el que consideramos 26 agrupaciones diferentes, tendríamos que plantearnos utilizar un algoritmo de agrupación más eficiente.

3.3.2. Birch

En este apartado nos disponemos a probar distintas configuraciones del método de agrupamiento Birch dado que los resultados obtenidos según las medidas del error consideradas son mejorables. El algoritmo de Birch funciona básicamente con la creación de un árbol de características. Este árbol se compone de subclústeres de características. A la hora de unir un elemento de la muestra a un subclúster ya existente, existe un valor denominado *threshold* que determina el radio máximo que puede tener un subclúster tras añadir un nuevo elemento a este. En la configuración inicial habíamos con siderado un valor de *threshold* = 0.1. Consideraremos ahora los valores de *threshold* = 0.01, 0.05 y 0.1 (ya usado) para comprobar si exigiendo que los elementos pertenecientes a un mismo clúster estén más cercanos se consiguen mejores resultados.

Los resultados obtenidos se comprueban en la tabla 6 .

Algoritmo	Número de clústeres	Tiempo (s)	Calinski-Harabaz	Silhouette
Birch thershold=0.01	5	0.8468658924102783	4738.4733439242891	0.54283017945120593
Birch thershold=0.05	5	0.8336312770843506	3291.922062351565	0.6213521542839423
Birch thershold=0.1	5	0.5136420726776123	17121.457021867674	0.82431192856413005

Tabla 6: Tabla comparativa

Curiosamente, los datos obtenidos no han sido los esperados, si no que se ha empeorado bastante el resultado. Esto se debe a que se fijamos un radio mínimo (thershold) demasiado pequeño, puede conllevar a una división excesiva.

Probamos entonces a cambiar el número de clústeres considerado para ver si así conseguimos mejores resultados. Consideraremos *n_clusters* = 5, 10, 15, 20, 25 y 30. Observamos los resultados obtenidos en la tabla 7 .

Algoritmo	Número de clústeres	Tiempo (s)	Calinski-Harabaz	Silhouette
Birch-5	5	0.5017960071563721	17121.457021867674	0.82431192856413005
Birch-10	10	0.5137999057769775	9006.64187149584	0.81640799518016738
Birch-15	15	0.5181581974029541	7380.0072834766788	0.81677529518571512
Birch-10	20	0.5040919780731201	11111.574122831509	0.8446212242143406
Birch-25	25	0.5516271591186523	8977.7466104123687	0.83472920678290963
Birch-30	30	0.5231690406799316	7673.7891658094231	0.83545390270552933

Tabla 7: Tabla comparativa

Observamos en los resultados generados que aunque hayamos mejorado (en algunos casos) el valor para el coeficiente Silhouette, el índice de Calinski-Harabaz ha empeorado en todos los casos, indicándonos que la poca mejora que se produce no merece la pena dado el número de clústeres utilizados. Esta conclusión la podemos sacar gracias a que el índice de Calinski-Harabaz se ve influenciado negativamente por el número de clústeres. De forma análoga a apartados anteriores representamos en una gráfica las medidas de error normalizadas para poder comprobar como mejoran o empeoran en proporción.

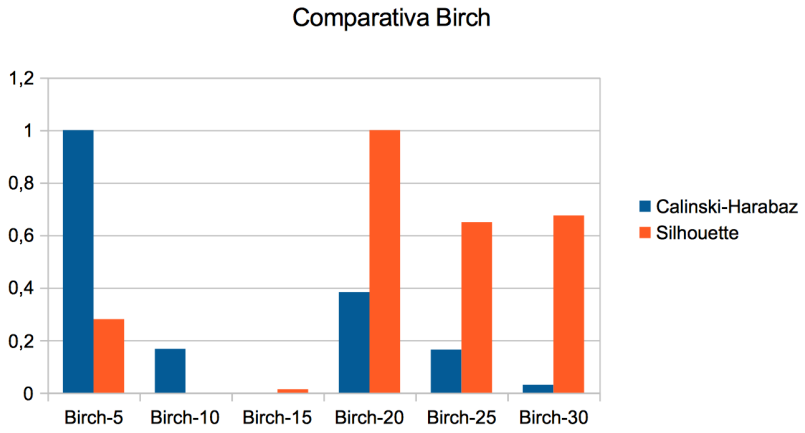

Figura 11: Comparativo de Birch cambiando número de clústeres.

En la tabla obsevamos claramente la influencia del aumento del número de clúster con respecto a los resultados obtenidos. Dado que ninguno de los métodos necesita mucho tiempo de ejecución podríamos elegir cualquiera de las configuraciones en función de nuestras necesidades. Si necesitamos más interpretabilidad escogeríamos un número más bajo de clústeres, pero si necesitamos el coeficiente Silhouette más alto claramente escogeríamos Birch-20, que aún podría ser interpretable en algún problema muy complejo.

3.3.3. K-Means

De forma análoga al caso de estudio anterior, nos disponemos a probar el algoritmo de K-Means con un número más alto de clústeres dado que el algoritmo DBSCAN que no necesita fijar el número de clústeres de altemano nos ha devuelto resultados con mayor número de estos.

Probaremos las configuraciones para $n\_clusters = 10, 20, 30, 40$ y 50.

Número de clústeres	Tiempo (s)	Calinski-Harabaz	Silhouette
5	0.7914528846740723	36154.186827454883	0.86718768508397859
10	1.4450700283050537	57391.708830182208	0.94536971154241689
20	2.972598075866699	107840.62197781177	0.96540748998566206
30	4.448544025421143	155459.01704207418	0.97789353562188497
40	6.025209188461304	214084.62464190929	0.98559107411074198
50	7.412207841873169	312973.80809317052	0.99034762577908197

Tabla 8: Tabla comparativa

En este caso hemos obtenido una gran mejoría, obtienendo mejores resultados según ambas medidas del error llegando a obtener en el coeficiente Silhouette prácticamente 1. Sin embargo, en este caso sí que ha aumentado el tiempo de ejecución en casi un 10 %. Esto no supone un problema pues los datos son bastante manejables, pero en un caso de estudio más grande podría producirnos un cuello de botella.

En conclusión, en nuestro caso concreto el mejor algoritmo de K-Means generado ha sido considerando 50 clústeres, aunque este resultado no es útil en la realidad pues sacar conclusiones de una agrupación con 50 clústeres sería prácticamente imposible. Por tanto, para problemas de clustering reales, nos conformaríamos con con el algoritmo **K-Means con 10 clústeres** pues nos ha proporcionado un alto coeficiente Silhouette mientras sigue siendo interpretable.

3.4. Interpretación de la segmentación.

Para la interpretación de la segmentación, vamos a utilizar solo algunos de los algoritmos implementados dado que ocuparía demasiado tiempo ocuparnos de todos ellos. Elegiremos aquellos que hayan funcionado bien además de aquellos que no hayan necesitado un alto número de clústeres pues esto dificultaría su interpretación.

3.4.1. Primera interpretación.

Para esta primera interpretación elegimos el algoritmo K-Means con la configuración inicial, con 5 clústeres. Elegimos este algoritmo pues de las configuraciones iniciales es el algoritmo que mejores resultados ha obtenido para ambas medidas del error de entre los que utilizan 5 clústeres. Para apoyar el estudio utilizaremos la Scatter Matrix de la figura 12.

Como en los casos de estudio anteriores, vamos a sacar las caracterizaciones de cada clúster generado además de buscar relaciones entre los atributos.

De forma general, podemos caracterizar a cada uno de los clústeres de la siguiente manera:

1. **Clúster 0:** Se caracteriza por tener un número considerable de vehículos implicados en el accidente, más de 5 y producir un número medio de heridos leves, entre 3 y 4.
2. **Clúster 1:** Este clúster abarca aquellos accidentes en los que ha habido dos heridos graves y el número de heridos leves se corresponde de forma casi lineal con el número de vehículos implicados. Es decir, en accidentes con pocos vehículos implicados hay menos heridos leves y en los que tienen más vehículos implicados habrá más heridos leves. Este hecho lo afirmamos pues en la tabla de TOT_VEHICULOS_IMPLICADOS x TOT_HERIDOS_LEVES, los puntos correspondientes al clúster 1 reprensentan prácticamente la gráfica de y=x (dependencia lineal).
3. **Clúster 2:** Este clúster se caracteriza por aquellos accidentes que han involucrado a muertos y bastantes heridos graves (2-4) mientras que hay un bajo número de heridos leves (0-3). Independientemente del número de vehículos implicados.
4. **Clúster 3:** Este clúster se caracteriza por aquellos accidentes en los que se han involucrado pocos vehículos (menos de 5) producendo pocos heridos graves, ningún muerto y muchos heridos leves (incluso más de 10).
5. **Clúster 4:** Este clúster se caracteriza por aquellos accidentes en los que el número de víctimas totales ha sido bajo (las víctimas de todas las gravedades es bajo).

A partir de las características de cada clúster, podemos deducir que tipos de accidentes se almacenan en cada uno de ellos:

1. En el clúster 0, dado el alto número de vehículos implicados y la levedad de las víctimas podemos suponer que se trata de un accidente en vía urbana.

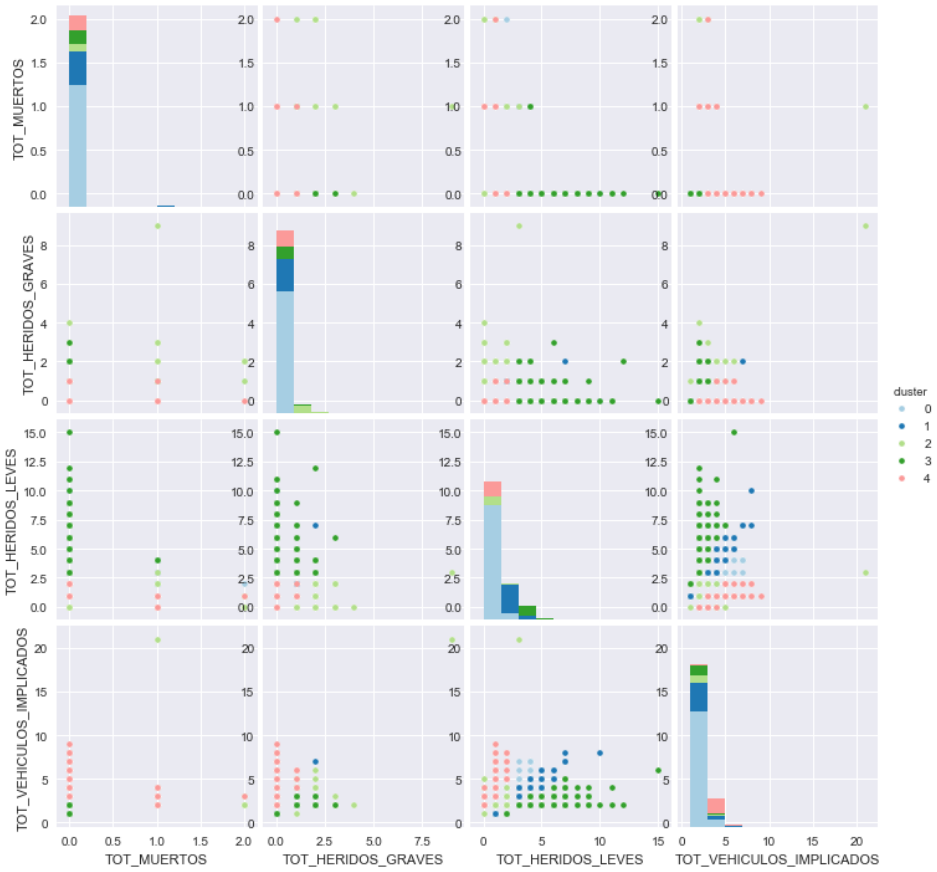

Figura 12: Scatter Matrix para K-Means con 5 clústeres.

2. En el clúster 1, al haber una correspondencia lineal entre el número de heridos y el número de vehículos implicados, llegando a haber muchos, también podría haber ocurrido en vía urbana.
3. En el clúster 2, por la gravedad de las víctimas la mayoría de los accidentes incluidos serán en carreteras nacionales o autovías.
4. En el clúster 3, es parecido al número clúster 0 pero involucrando pocos vehículos, podría deberse tambn a accidentes en vía urbana pero con menos implicados.
5. En el clúster 4 no podemos sacar conclusiones muy representativas.

Obtenemos así, a rasgos generales podríamos clasificar los accidentes con colisión de vehículos en la Comunidad Autónoma de Cataluña en cinco tipos. Aunque en este caso no hemos sacado tantas conclusiones como en el caso de estudio anterior, notar que si observamos la diagonal de la Scatter Matrix, comprobamos que el clúster 0 es notablemente más grande que el resto de clúster (agrupando a más de la mitad de los datos de la muestra). Por tanto, podríamos deducir que más de la mitad de los accidentes estudiados se han producido en vías urbanas. Esta conclusión nos llevaría a pensar que la zona más peligrosa para la conducción son las vías urbanas, sin embargo, la gravedad de los heridos suele ser más baja que en otros tipos de accidentes (como en los que recoge el clúster 2, aunque sea menos numeroso).

3.4.2. Segunda interpretación.

Para esta segunda interpretación, dado que no hemos conseguido ningún método de agrupamiento que nos ofrezca un coeficiente de Silhouette por encima de 0.9 utilizando pocos clústeres, elegimos otro de los algoritmos definidos inicialmente, elegimos el SpectralClustering pues el resultado del coeficiente de Silhouette ha sido muy parecido al obtenido con Birch, pero el índice de Calinski-Harabaz ha sido claramente superior.

Análogamente a los casos anteriores, apoyamos nuestro análisis en la Scatter Matrix producida por este algoritmo que observamos en la figura 13 .

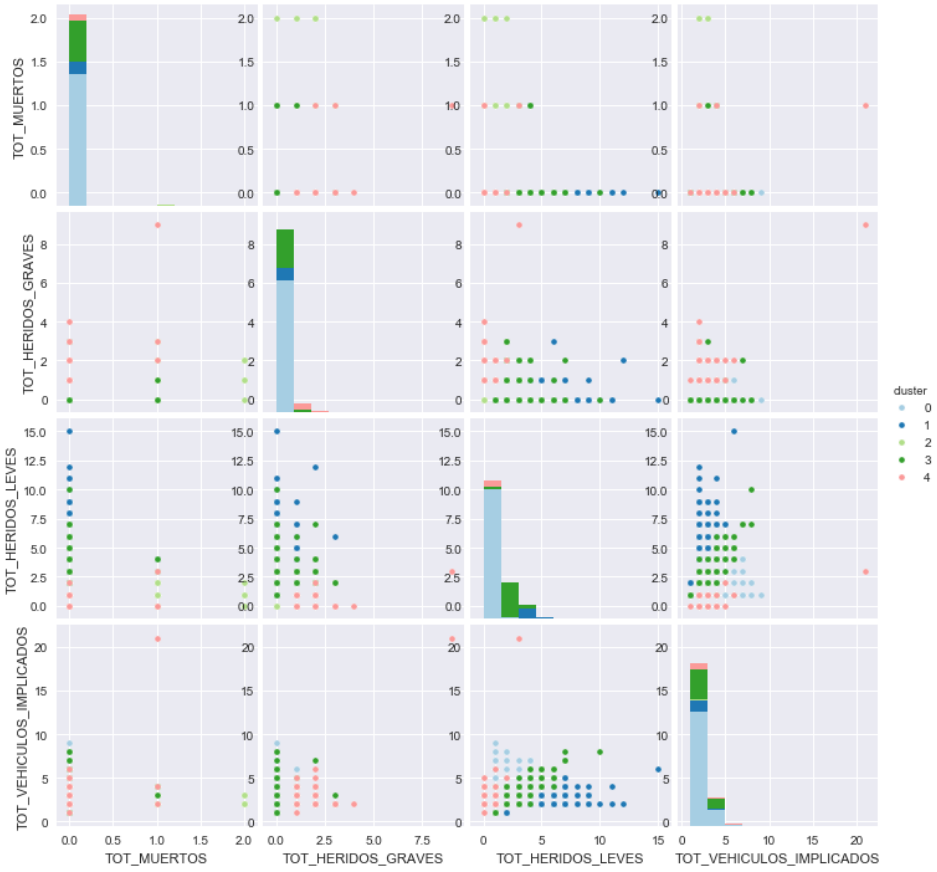

Figura 13: Scatter Matrix para SpectralClustering.

Igual que en el resto de casos, vamos a intentar caracterizar a los clústeres en función de los atributos utilizados en este caso de estudio:

1. **Clúster 0:** Este clúster se caracteriza por tener bastantes vehículos implicados y producir un número de víctimas totales bajo.
2. **Clúster 1:** Este clúster se caracteriza por involucrar pocos vehículos (menos de 5) pero producir muchas víctimas de gravedad leve y pocas de mayor gravedad, no produciendo ningún muerto.
3. **Clúster 2:** Este clúster se caracteriza por involucrar más víctimas mortales.

4. **Clúster 3:** Este clúster involucra a pocos o ningún muertos y heridos graves, pero se caracteriza principalmente por la correspondencia medio lineal que hay entre vehículos implicados y los heridos leves, al igual que en el clúster 1 de la interpretación anterior. Esto nos da un indicio de que quizás haya clústeres que recojan a los mismos datos habiendo utilizado algoritmos de agrupamiento diferentes, lo que implicaría la relación entre estos datos agrupados.
5. **Clúster 4**: Este clúster se caracteriza por aquellos accidentes en los que el número de víctimas totales ha sido bajo (las víctimas de todas las gravedades es bajo).

3.4.3. Relación entre ambas.

El habernos dado cuenta de que el clúster 3 se corresponde con el clúster 1 en el caso de la interpretación anterior, nos hace pensar si existen relaciones entre el resto de clústeres, comparamos las dos últimas filas de cada una de las matrices en la figura 14.

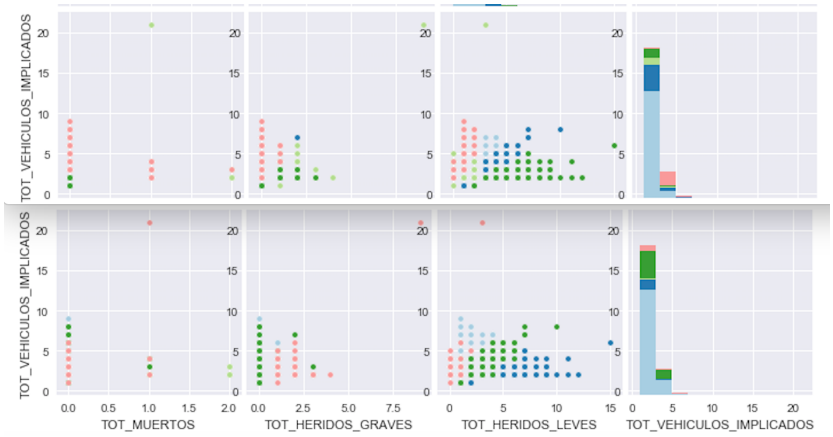

Figura 14: Comparativa entre las dos interpretaciones comentadas.

Observamos que a rasgos generales no se corresponden unos clústeres con otros, sin embargo en cuanto a la relación de vehículos implicados con el total de heridos leves, obtenemos que la clasificación si ha sido parecida.

En ambos casos hay un clúster que recoge la dependencia lineal, otro que recoge aquellos con muchos heridos leves y pocos vehículos implicados, otro que recoge a pocos vehículos leves con relativamente pocos vehículos implicados (0-5) y por último otro que recoge aquellos accidentes en los que el número de vehículos implicados han sido muchos y el númeor de heridos leves pocos. Los dos primeros clústeres comentados se diferencian de forma más significativa. Y por último, hay un clúster que representa la presencia de heridos graves y muertos.

Encontrar estas caracterizaciones comunes nos hace pensar que estas caracterizaciones más generales si podría agrupar de manera más general el caso de estudio planteado en este apartado, obteniendo así conclusiones generales de interés para la aseguradora. Como por ejemplo, en ambos casos el clúster más grande ha sido aquel que recoge los accidentes con número de vehículos involucrado alto y número de heridos leves bajo, siendo este el accidente más típico a rasgos generales.

4. Caso de estudio 3

4.1. Descripción.

En este caso de estudio, nos disponemos a estudiar los accidentes que han tenido lugar en condiciones óptimas de conducción para poder obtener conclusiones sobre este tipo de accidentes.

Definimos como condiciones óptimas de conducción a ser pleno día, con buen tiempo y la superficie de la calzada limpia.

Al igual que en el caso anterior, si realizamos este estudio en todos los accidentes ocurridos en España en el año 2013 tendríamos 53664 muestras, que sería un tamaño muestral demasiado grande. De forma análoga a la anterior, estudiamos solo aquellos accidentes que se hayan producido en la Comunidad Autónoma de Andalucía, reduciendo así el problema a 8921 muestras.

Para hacer un poco diferente este caso, vamos a incluir también el TOT_VICTIMAS para estudiar esta variable que en el resto de casos hemos despreciado por redundante. Así, las variables a usar serían:

1. TOT_VICTIMAS
2. TOT_MUERTOS
3. TOT_HERIDOS_GRAVES
4. TOT_HERIDOS_LEVES
5. TOT_VEHICULOS_IMPLICADOS

En la figura 15 observamos el proceso de selección del caso de estudio:

```
#Definición del subcaso
def define_subset():
    accidentes = pd.read_csv('accidentes_2013.csv')


    subset = accidentes.loc[accidentes['LUMINOSIDAD'].str.contains("PLENO DÍA")]
    subset = subset.loc[subset['FACTORES_ATMOSFERICOS'].str.contains("BUEN TIEMPO")]
    subset = subset.loc[subset['SUPERFICIE_CALZADA'].str.contains("LIMPIA")]
    subset = subset.loc[subset['COMUNIDAD_AUTONOMA'].str.contains("Andalucía")]

    usadas = ['TOT_VICTIMAS', 'TOT_MUERTOS', 'TOT_HERIDOS_GRAVES', 'TOT_HERIDOS_LEVES', 'TOT_VEHICULOS_IMPLICADOS']
    X = subset[usadas]

    return X
```

Figura 15: Definición del tercer caso de estudio.

4.2. Análisis del caso de estudio.

Para el análisis de este caso de estudio hemos considerado 5 algoritmos de clustering diferentes, que son:

- **K-Means** con parámetros $n\_clusters = 5$, $n\_init = 100$ y la inicialización con $init = k\text{-}means{+}{+}$
- **MeanShift** con parámetro de *bandwidth* estimado usando la función del mismo paquete *estimate_bandwidth* introduciendo como parámetro de esta $quantile = 0.3$.
- **MiniBatchKMeans** con parámetros $n\_clusters = 5$ y $init = k\text{-}means{+}{+}$.

- **AgglomerativeClustering** con parámetros *n_clusters = 5* y *linkage = 'ward'*.
- **Birch** con parámetros *n_clusters* = 5 y *threshold* = 0.1

Observamos los resultados otenidos para cada uno de los algoritmos en la tabla:

Algoritmo	Número de clústeres	Tiempo (s)	Calinski-Harabaz	Silhouette
K-Means	5	0.7914528846740723	20294.724233792091	0.84940303031825759
MiniBatchesKMeans	5	0.36252379417419434	17284.743795964318	0.84005514863306574
MeanShift	59	1.9177911281585693	83828918232	0.93844059208433639
AgglomerativeClustering	5	0.5069057941436768	17413.692118013685	0.83050311209797822
Birch	5	67.81920003890991	10259.927311060204	0.78327915590661701

Tabla 9: Tabla comparativa

En una visión general a la tabla podemos obtener resultados similares al resto de casos de estudio. Como casi siempre, el algoritmo de MeanShift ha obtenido los mejores resultados, siendo muy superiores al resto. Sin embargo este resultado ha sido obtenido haciendo uso de un elevado número de clústeres por lo que no es comparable, ya compararemos en la configuración de los parámetros.

En cuanto al algoritmo de referencia, K-Means, ha obtenido resultados relativamente buenos para el coeficiente de Silhouette pero como siempre ha obtenido un resultado bastante mejor en el índice de Calinski-Harabaz, consiguiendo un valor claramente superior al del resto de algoritmos implementados.

El resto de algoritmos tienen un comportamiento similar exceptuando el algoritmo Birch utilizado que produce un valor del Silhouette bastante inferior al resto. Este tema lo estudiaremos en la configuración de los parámetros.

4.3. Configuración de los algoritmos.

En esta sección vamos a realizar diferentes configuraciones de algunos de los algoritmos ya implementantados en la sección anterior. Con las configuraciones buscamos mejorar aquellos métodos que nos han dado un resultado peor del esperado o estudiar el comportamiento de estos al cambiar algunos parámetros.

4.3.1. Birch

Como ya hemos visto en los comentarios generales, este algoritmo es el que obtenía peores resultados en ambas medidas del error para el caso de estudio en cuestión. Por ello, nos planteamos si aumentando el número de clústeres podría llegar a ser competitivo con el resto de algoritmos.

Para comprobarlo, vamos a ejecutar el método para valores de *n_cluters* = 10, 20, 30, 40, 50 y 60. Elegimos este tope dado que el algoritmo MeanShift ha utilizado 59 clústeres para obtener el resultado, que es el mejor obtenido. Recordamos que otro parámetro de este algoritmo de clustering es el *threshold*, que representaba el radio máximo que puede tener un subclúster a la hora de añadir un nuevo elemento. Dado que estamos aumentando tanto el número de clústeres considerado, estas configuraciones las realizamos con un valor de *threshold* = 0.01.

En la tabla 10 observamos los resultados obtenidos para este cambio de parámetros en el algoritmo Birch:

Algoritmo	Número de clústeres	Tiempo (s)	Calinski-Harabaz	Silhouette
Birch-10	10	0.5068669319152832	8662.8009839429524	0.81118078183795872
Birch-20	20	0.4963839054107666	6741.0307367164514	0.79321368716527896
Birch-30	30	0.4923591613769531	5729.12473678505	0.77672908272785746
Birch-40	40	0.4910709857940674	25427.371474148422	0.9145812529897801
Birch-50	50	0.48962903022766113	24193.137351727204	0.90755142063091243
Birch-60	60	0.4867401123046875	21076.131298613065	0.90988494610575554

Tabla 10: Tabla comparativa

Los datos obtenidos son curiosos pues vemos que, en las primeras configuraciones ambos coeficientes de error aumentan o disminuyen de forma similar, mientras que, a partir de 40 clústeres este valor empieza a disminuir aunque el valor de Silhouette aumente (lo que ocurre en la comparación entre Birch-50 y Birch-60). Esto se puede deber, como ya hemos comentado, a la influecia negativa que tiene el número de clústeres utilizado en el índice de Calinski-Harabaz, y es que los resultados para Birch-50 y Birch-60 no parecen mejorar en exceso según el coeficiente Silhouette, pero sin embargo estamos aumentando en 10 el número de clúster considerados, diminuyendo así este índice.

Además, observamos que a diferencia de la mayoría de los casos, no se produce una mejora progresiva de los resultados del algoritmo con el aumento del número de clústeres, si no que se producen ciertos siendo el mejor algoritmo implementado, según ambas medidas el Birch-40.

En cuanto a los resultados obtenidos en comparación con los 5 algoritmos definidos inicialmente, vemos que en ningún caso supera los resultados del MeanShift utilizado en ninguna de las configuraciones pero que mejora al resto de algoritmos considerados, obteniendo una solución bastante competitiva.

Por último, vamos a analizar la influencia del *threshold* para Birch-10, ya que ha conllevado mejoras con respecto a Birch-5 (implementado inicialmente) y sigue siendo interpretrable. Recordamos que en apartado anterior, una disminución del valor de *threshold* nos proporcionaba valos peores, veamos si aumentando el *threshold* en este caso conseguimos mejorar los resultados ya obtenidos.

Algoritmo	Threshold	Tiempo (s)	Calinski-Harabaz	Silhouette
Birch-10	0.01	0.6928489208221436	8662.8009839429524	0.81118078183795872
Birch-10	0.05	0.546565055847168	8909.7476501010897	0.74802515187879237
Birch-10	0.07	0.3652658462524414	9063.2856548371637	0.82824538512832768

Tabla 11: Tabla comparativa

Observamos en la tabla 11 las medidas obtenidas para estas configuraciones del anteriormente definido Birch-10. A rasgos generales vemos que el valor del índice Calinski-Harabaz es muy similar en los tres casos, mientras que el coeficiente Silhouette si que varía, obteniendo un mejor resultado para *threshold* = 0.07, pero no produciendo muchas mejoras.

Sin embargo, si nos fijamos en el tiempo, obtenemos que cuanto más aumentaba el *threshold* más bajaba el tiempo de ejecución, dado que somos más permisivos a la hora de asignar un dato a un determinado clúster. Así, hemos conseguido un resultado bastante similar (de hecho mejor aunque insignificante) en un tiempo bastante menor. Esto podría ser una cuestión a tener en cuenta si prima el tiempo de ejecución en el problema o si estamos trabajando con conjuntos de datos más grandes.

4.3.2. AgglomerativeClustering

En esta subsección, vamos a probar a implementar un algortimo jerárquico introduciendo una matriz de conectividad como se realiza en [5].

Esta matriz se define utilizando un grafo de vecinos con la función *kneighbors_graph* que recibe como parámetro el número de vecinos que consideramos y si queremos que se le incluya a sí mismo como vecino más cercano. Posteriormente, se hace esta matriz simétrica y se introduce como connectivity.

Vamos a comparar la diferencia en cuanto a las medidas consideradas si introducimos este parámetro como si no lo introducimos para valores de *n_clusters* = 5 y 10. Además, vamos a mantener el mismo método considerado en la configuración inicial, el *Ward* y estableceremos como número de vecinos a considerar 10 en ambos casos. Obtenemos:

Algoritmo	Connectivity	Número clústeres	Tiempo (s)	Calinski-Harabaz	Silhouette
Ward	No	10	1.8038010597229004	33815.17577777199	0.91453349519535532
Ward	Si	10	67.07133531570435	24623.831012430092	0.86167195966521126
Ward	No	20	1.675704002380371	76046.726922610338	0.96591466828088923
Ward	Si	20	64.31138610839844	65785.390174212327	0.96381142298804179

Tabla 12: Tabla comparativa

En la tabla de resultados obtenida, la tabla 12 podemos comprobar que no se han producido mejoras al utilizar esta matriz de conectividad. Además, el tiempo necesitado se ha disparado siendo de los algoritmos más lentos que hemos utilizado a lo largo de esta práctica. Por tanto, podemos concluir categóricamente que mejor no utiliar este parámetro y dejar funcionar al método por si solo.

Además, comprobamos que las dos configuraciones implementadas utilizando más número de clústeres mejoran bastante a la configurada anteriormente, por lo que, sí ha sido buena idea aumentar el número de clústeres a considerar para la solución.

4.3.3. K-Means

Como en los dos casos de estudio anteriores hemos analizando el K-Means obteniendo mejoras significativas al aumentar el número de clústeres considerados, vamos a analizar el comportamiento en este caso buscando una configuración del K-Means que supere al 0.9 en el coeficiente Silhouette.

En cuanto a las medidas tomadas, el comportamiento esperado es el mismo que en el resto de casos. Los resultados obtenidos para valores de *n_clusters* = 10, 20, 30, 40, 50 y 60 son:

Algoritmo	Número clústeres	Tiempo (s)	Calinski-Harabaz	Silhouette
K-Means	10	1.0530681610107422	37705.753251343354	0.91743064265958396
K-Means	20	1.8694651126861572	87285.288395396157	0.96321576327887837
K-Means	30	2.9692699909210205	133466.86544559945	0.97385750931624016
K-Means	40	4.309677839279175	173473.91370984988	0.98153859593435977
K-Means	50	4.927326202392578	238132.94090982893	0.98415986570392111
K-Means	60	6.063432216644287	411399.36012731941	0.98792073304601202

Tabla 13: Tabla comparativa

Efectivamente, obtenemos los resultados que esperábamos y es que aumentan ambas medidas con el aumento del número de clústeres aunque no de la misma manera. El coeficiente de Silhouette aumenta bruscamente al principio mientras que el coeficiente de Calinski-Harabaz aumenta bruscamente al final y el tiempo lo hace de forma lineal, como ya habíamos comentado en la Sección 2. Por tanto, podemos afirmar que este comportamiento es característico del método y no dependiente del caso de estudio en cuestión.

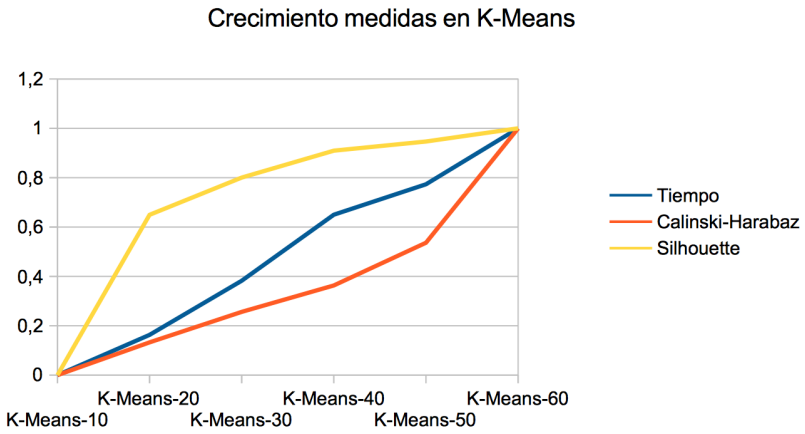

Figura 16: Estudio del crecimiento de las medidas tomadas para K-Means.

Sin embargo, estos crecimientos los hemos analizado considerando un número muy alto de clústeres para estudiar de nuevo su crecimiento. Sin embargo, en la práctica una solución con 60 clústeres no tendría sentido. Como ademñas hemos conseguido muy buenos resultados para 10 clústeres nos planteamos intentar bajar el número de clústeres para MeanShift, como analizaremos en la siguiente subsección.

4.3.4. MeanShift

En todos los casos en los que hemos utilizado alguna configuración del MeanShift hemos obtenido muy buenos resultados, los mejores. Sin embargo, el número de clústeres utilizado ha sido demasiado alto. En el resto de ocasiones, lo que hemos hecho ha sido aumentar el número de clústeres en otros métodos para conseguir superar a este algoritmo, sin embargo, esto no es útil en problemas de agrupamiento pues necesitamos en la mayoría de los casos obtener conclusiones de los grupos formados.

Por este motivo, nos disponemos a configurar el parámetro del *bandwidth* (ancho de banda) en el algoritmo de MeanShift, para esto, vamos a variar el parámetro del *quantile* en el método que lo estima. Tomaremos valor de *quantile* = 0.3, 0.4 y 0.5, obteniendo los siguientes resultados:

Algoritmo	quantile	Número clústeres	Tiempo (s)	Calinski-Harabaz	Silhouette
MeanShift	0.3	59	6.591464996337891	34267.83828918232	0.93844059208433639
MeanShift	0.4	10	9.056087970733643	2240.0597886907244	0.49093852967329532
MeanShift	0.5	3	22.70424222946167	3449.0901806076445	0.65645380850054147

Tabla 14: Tabla comparativa

Observamos en la tabla 14 que ocurre lo que esperábamos y es que el número de clústeres necesitados ha descendido de manera muy brusca. Sin embargo, no hemos obtenido un método competitivo con el resto de algoritmos implementados hasta el momento pues el coeficiente Silhouette no alcanza ni 0.7 en ninguno de los casos y el índice de Calinski-Harabaz también es excesivamente bajo. Lo que es más, el tiempo de ejecución ha aumentado bastante, por lo que no hemos conseguido ninguna confguración del MeanShift que merezca la pena pues no hemos obtenido el equilibro entre buenos resultados e interpretabilidad.

4.4. Interpretación de la segmentación.

Para la interpretación de la segmentación, vamos a utilizar solo algunos de los algoritmos implementeados. Para la elección, usamos aquellos que hayan funcionado bien además de tener un bajo número de clústeres por la interpretabilidad.

4.4.1. Primera interpretación.

Para la primera interpretación elegimos, como siempre, el algoritmo K-Means con la configuración inicial de 5 clústeres. Utilizaremos la Scatter Matrix de la figura 17.

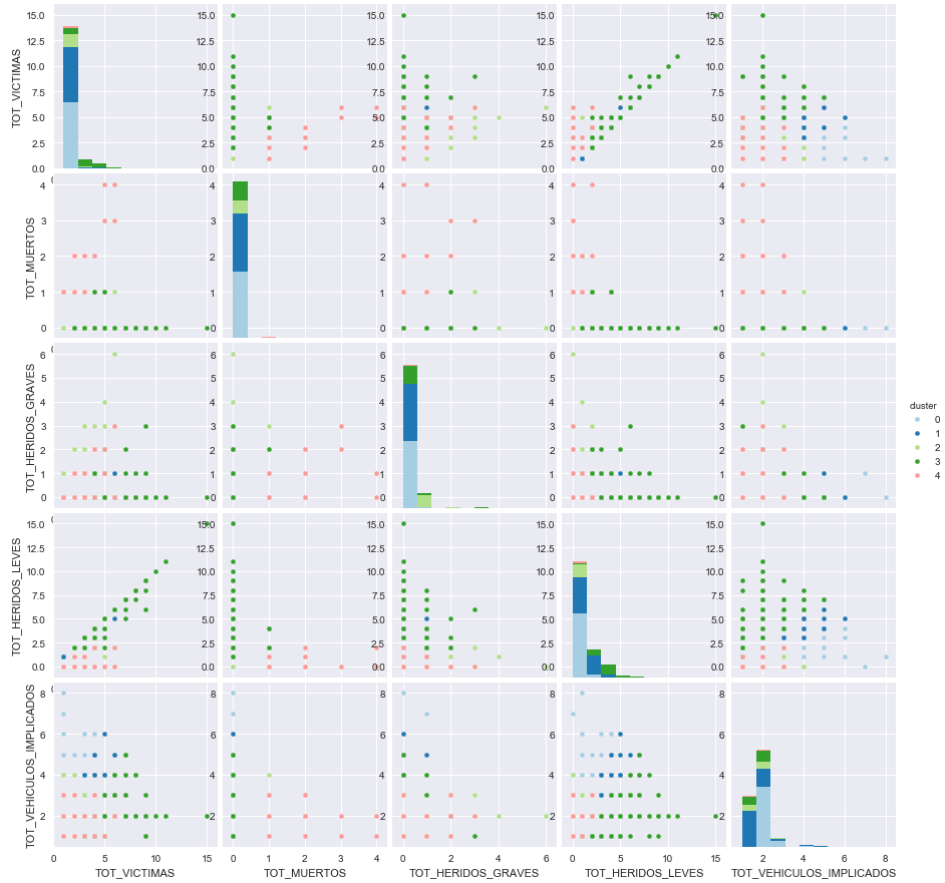

Figura 17: Scatter Matrix para K-means con 5 clústeres.

Como en los casos anteriores, obtenedremos caracterizaciones de los diferentes clústeres ademas de buscar relaciones entre las variables utilizadas.

Se han generado 5 clústeres, y las características generales de estos son:

- **Clúster 0:** Se caracteriza principalmente por aquellos accidentes que con un número de vehículos implicado alto y pocas víctimas totales.
- **Clúster 1:** Se caracteriza también por un gran número de vehículos involucrados con la diferencia de haber más heridos leves.
- **Clúster 2:** No tienen características muy significativas.
- **Clúster 3:** Se caracteriza por un gran número de víctimas involucradas, sobretodo si hablamos de heridos leves, de hecho no contiene ningún muerto.
- **Clúster 4:** Se caracteriza por aquellos accidentes que involucran pocos vehículos y pocao heridos mientras que involucran gran número de muertes.

En cuanto a las relaciones entre las variables, observamos una clara dependencia lineal entre TOT_VICTIMAS Y TOT_HERIDOS_LEVES, de forma que cuanto más aumenta el número de víctimas más aumenta el número de heridos leves. Esto nos indica que la mayoría de las víctimas producidas en estos accidentes son leves, hecho esperado por las características del subcaso de estudio.

En cuanto al tamaño de los clústeres podemos ver en la diagonal de la matriz que claramente el clúster 4 es el más pequeño, mientras que los clústeres 0 y 1 son los más numerosos con muchísima diferencia.

Por las características generales del clúster 4, podemos deducir que han sido accidentes graves pues hay pocas vícitmas y casi todas graves o muertos, por tanto, serán accidentes en carretera nacional o autovía. Como el tamaño de este clúster es el más pequeño con muchísima diferencia, podemos obtener la conclusión de que los accidentes graves que ocurren en condiciones óptimas de conducción son los menos frecuentes.

Por otro lado, los clústeres 0 y 1 representan a accidentes con un gran número de vehículos involucrados y los heridos prácticamente todos leves. Por tanto, estos representan accidentes en zona urbana de gravedad leve.

4.4.2. Segunda interpretación.

Para esta interpetación vamos a utilizar el algoritmo jerárquico implementado en los 5 algoritmos iniciales, un AgglomerativeClustering Ward con 5 clústeres. La Scatter Matrix para este algoritmo es la de la figura 18.

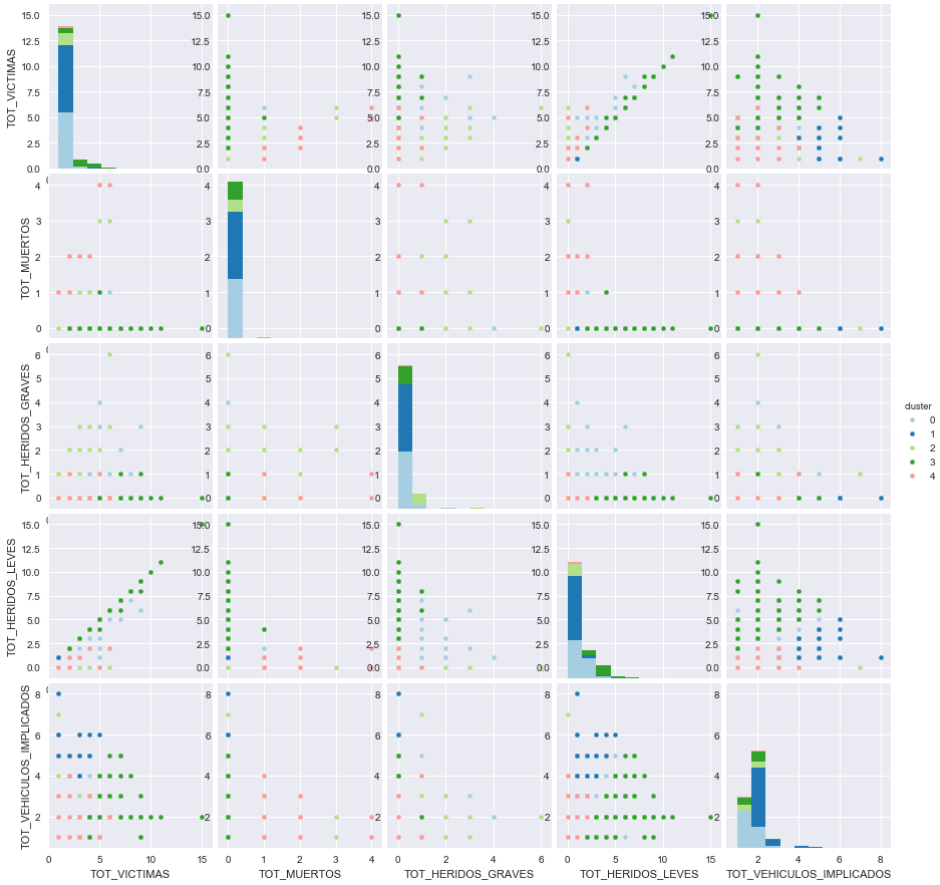

Figura 18: Scatter Matrix para AgglomerativeClustering Ward con 5 clústeres.

Como hasta el momento, obtenedremos caracterizaciones de los diferentes clústeres ademas de buscar relaciones entre las variables utilizadas.

Se han generado 5 clústeres, y las características generales de estos son:

- **Clúster 0:** Se caracteriza por aquellos accidentes que involucran muy pocos vehículos y tienen muy pocas víctimas.
- **Clúster 1:** Este clúster se caracteriza principalmente por aquellos accidentes que involucran un gran número de vehículos. Además, el número de vícitmas involucradas es relativamente bajo.
- **Clúster 2:** Se caracteriza por los accidentes que tienen un número medio de heridos graves.
- **Clúster 3:** Contiene los accidentes que causaron un mayor número de víctimas totales, en particular de víctimas leves pues son las más numerosas en este caso de estudio.
- **Clúster 4:** Se caracteriza por aquellos clústeres que involucraron pocos vehículos y pocas víctimas pero entre ellas muertos.

En cuanto a las conclusiones generales en función del tamaño y las dependencias entre variables, son las mismas que en la interpretación anterior.

Añadir que una características de los accidentes de este caso de estudio, de forma general, es que

el número total de víctimas no aumenta conforme aumenta el número de vehículos, si no que todo lo contrario. Esto se puede deber a que parte de las víctimas en estos accidentes no pertenecían a los vehículos por lo que gran parte de estos podrían ser atropellos en zona urbana, reforzando así el comentario ya realizado de que la mayoría de los accidentes ocurren en zona urbana.

4.4.3. Conclusiones de las interpretaciones.

Como hemos comentado al final de cada interpretación, el tamaño de los clústeres con características de zona urbana, la relación entre el total de víctimas y aquellas con heridas leves y la relación inversa entre el total de víctimas y el número de vehículos implicados nos lleva a la conclusión de que en este caso de estudio, los accidentes en condiciones óptimas de conducción ocurren en su gran mayoría en las zonas urbanas.

El resultado obtenido aplicando los algoritmos de agrupamiento nos parece lógico en la realidad pues en condiciones buenas de conducción, el ambiente más probable para que se produzca un accidente sería en zona urbana dada la dificultad de estas vías por la presencia de peatones y el tráfico.

5. Contenido adicional

5.1. Eliminación de outliers

Acostumbrados a la importancia del preprocesado de los datos para los problemas de clasificación y regresión, nos planteamos si el preprocesado de los datos en esta práctica no ha sido el suficiente. El único preprocesado que hemos realizado ha sido la normalización de los datos siguiente una normal L2 para que no primen unos atributos sobre otros.

Ya de forma adicional en la Sección 2 habíamos estudiado la presencia de outliers mediante los algoritmos jerárquicos. Como ya habíamos comentado, estos algoritmos se ven muy afectados por la presencia de outliers por su funcionamiento: parten de un clúster por dato y luego van fusionando clústeres pues esta fusión nunca llega a ocurrir si los datos están muy alejados del resto de datos. Por tanto, encontrábamos así una manera de preprocesar los datos eliminando estos outliers.

En esta sección, vamos a considerar una submuestra de 5000 datos elegidos de forma aleatoria de entre la base de datos entera y vamos a comparar este preprocesado anteriormente comparado con un preprocesado utilizando el algoritmo DBSCAN.

El algoritmo DBSCAN etiqueta con número de clúster -1 a aquellos datos que no consigue clasificar en ningún clúster de los utilizados. Para filtrar los datos con este método, eliminaremos todos aquellos datos cuya predicción resulte que pertenecen al clúster -1.

De esta forma, el procedimiento a seguir será aplicar el filtrado con cada uno de los métodos y comparar, en primer lugar el tamaño del subconjunto generado y, en segundo lugar, el comportamiento del método más usado a lo largo de la práctica, el K-Means con 5 clústeres para ver en cuál obtenemos mejores resultados.

Para el preprocesado vamos a utilizar como algoritmos:

1. Genético: AgglomerativeClustering ward con k = 100.
2. DBSCAN con eps = 0.1

Para el filtrado de datos en el genético utilizamos el código proporcionado en la web de la asignatura y para el filtrado con DBSCAN el siguiente código:

```
def delete_outliers_dbscan(prediction, subset):
    labels = prediction[1]
    clusters_labels = pd.DataFrame(labels,index=subset.index,columns=['cluster'])
    cluster_subset = pd.concat([subset, clusters_labels], axis=1)
    new_subset = cluster_subset[clusters_labels['cluster']!=-1]

    return new_subset
```

Figura 19: Código para filtrado con DBSCAN.

Tras filtrar obtenemos que los nuevo conjuntos de datos tienen un tamaño de:

- 4919 tras utilizar el algoritmo jerárquico.
- 4952 tras utilizar DBSCAN.

Luego, el filtrado utilizando el algoritmo jerárquico ha encontrado más outliers. Veamos cómo se comporta el método K-Means en estos nuevos subconjuntos de datos.

Filtrado	**Tiempo (s)**	**Calinski-Harabaz**	**Silhouette**
Ward	0.41770482063293457	13163.398299560537	0.86798477274531105
DBSCAN	0.4223790168762207	20299.028469201414	0.88047049623773188

Tabla 15: Tabla comparativa

Observamos en la tabla 15 observamos los resultados obtenidos en ambos conjuntos tras el filtrado. En cuanto al tiempo de ejecución, es en ambos casos muy similar dado que los subconjuntos eran prácticamente igual de grandes.

En cuanto a las medidas del error, hemos obtenido que el algoritmo de K-Means se ha comportado mejor en el subconjunto creado a partir del filtrado con DBSCAN a pesar de que se hubieran eliminado menos outliers. Esto se puede deber a que el algoritmo K-Means no se ve tan afectado por los outliers y sin embargo, es importante tener mayor número de muestras para obtener los resultados.

Referencias

[1] Dirección General de Tráfico (DGT) https://sedeapl.dgt.gob.es/WEB_IEST_CONSULTA/subcategoria.faces

[2] Apuntes de la asignatura. http://sci2s.ugr.es/sites/default/files/files/Teaching/GraduatesCourses/InteligenciaDeNegocio/Curso17-18/Tema05-Modelos%20de%20Agrupamiento%20%202017-18.pdf

[3] Wikipedia *Silhouette (clustering)* https://en.wikipedia.org/wiki/Silhouette_(clustering)

[4] Scikit-Learn *Clustering* http://scikit-learn.org/stable/modules/clustering.html

[5] Scikit-Learn *AgglomerativeClustering* http://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html#sphx-glr-auto-examples-cluster-plot-cluster-comparison-py