Máster Universitario en Ingeniería Informática

Tratamiento Inteligente de Datos

PRÁCTICA 2: Agrupamiento

Wine

UNIVERSIDAD DE GRANADA

Carlos Santiago Sánchez Muñoz

Grupo de prácticas 1 - Jueves

Email: carlossamu7@correo.ugr.es

12 de noviembre de 2020

Índice

1. Descripción del problema

La base de datos contiene los datos reales de 178 vinos de una misma región de Italia. Cada instancia está compuesta por trece atributos numéricos más una clase (la primera columna) que determina el nivel de alcohol del vino (tres tipos; `1`, `2` y `3`). Dicha clase es la solución al proceso de clustering por lo que se eliminará en durante el desarrollo.

Los trece atributos que se van a tratar, de los cuales el primero es la clase, son:

1. Alcohol
2. Malic acid
3. Ash
4. Alcalinity of ash
5. Magnesium
6. Total phenols
7. Flavanoids
8. Nonflavanoid phenols
9. Proanthocyanins
10. Color intensity
11. Hue
12. OD280/OD315 of diluted wines
13. Proline

De estos atributos `Type` (el tipo de alcohol), `Magnesium` y `Proline` son enteros, el resto son flotantes.

Comenzamos con la lectura correcta del fichero `csv` de los datos, e imprimimos información relativa a ellos:

```
""" Lectura de datos. Devuelve el dataframe.
"""
def read_data():
    names = ['Type','Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash',
             'Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols',
             'Proanthocyanins', 'Color intensity', 'Hue',
             'OD280/OD315 of diluted wines', 'Proline']
    df = pd.read_csv('wine.data', names=names)

    if(IMPRIME_INFO):
        print("\nInformacion del dataframe:")
        print(df.info())
        print("\nDataframe:")
        print(df)
    return df
```

2. Clustering jerárquico

En esta sección se va a realizar un algoritmo de *clustering* jerárquico para analizar en cuántos *clusters* diferentes se podrían agrupar los datos.

Examinar el número de *clusters* es un paso esencial en los problemas de agrupamiento. En este caso se va a realizar un dendograma y a partir de él extraer conclusiones del número de *clusters*. Hay que tener en cuenta que aunque en este caso no hay valores perdidos sí que puede haber *outliers* y esto puede hacernos errar en la decisión.

```
""" Pinta el dendograma de los datos de X.
- X: datos.
- title: titulo. Por defecto "Dendograma".
"""
def dendograma(X, title="Dendograma"):
    print("Calculando el dendograma")
    dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
    plt.xlabel('Vinos')
    plt.ylabel('Distancias íEucldeas')
    plt.title(title)
    plt.gcf().canvas.set_window_title("Practica 2 - Agrupamiento")
    plt.show()
```

En Python existe un paquete, `scipy.cluster.hierarchy`, que contiene esta funcionalidad y permite obtener de manera relativamente sencilla el dendograma de un conjunto de datos. La función implementada calcula el dendograma usando la distancia euclídea y posteriormente lo plotea.

```
# CLUSTERING JERÁRQUICO
print("\n------ Tratando los datos normales ------")
X, y =split_data(df)
dendograma(X)
```

La ejecución desde el programa `main` podría ser como la anterior y el dendograma obtenido es el de la Imagen 1. En él observamos que dependiendo de la altura a la que cortemos el número de *clusters* puede ser: 1,2,3,4,6,7, etc.

Elijo tres *clusters* ya que dos grandes *clusters* no tienen sentido ya que no se deja al algoritmo avanzar lo suficiente para realizar las divisiones. Por cómo es el dendograma, 4 *clusters* no tendría sentido, más coherente sería 6 pero parece un número excesivo. Parece razonable probar con 3 y en caso de que los resultados no son satisfactorios intentar realizar pruebas con otra elección.

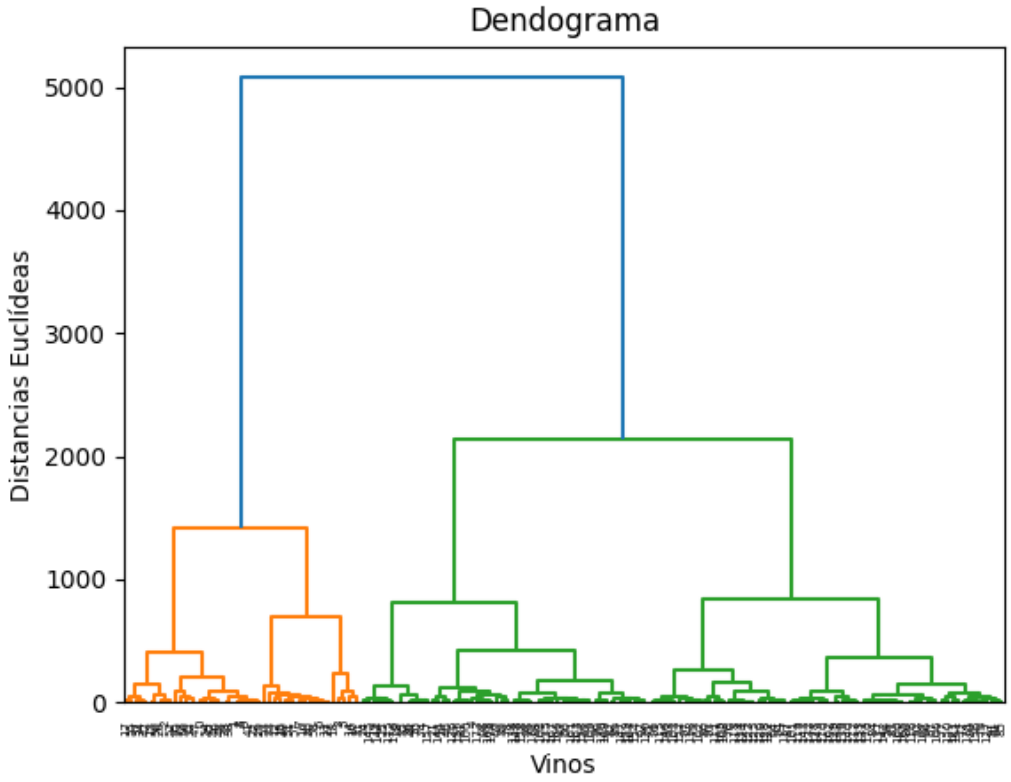

Imagen 1: Dendograma del clustering jerárquico

3. Eliminando *outliers*

En esta sección vamos a analizar la existencia de *outliers* y vamos a proceder a su eliminación en caso necesario. Posteriormente se volverá a realizar el *clustering* jerárquico y la decisión que se tomó.

```
""" Borra las instancias en donde alguna de las columnas sea un outlier.
- df: dataframe.
"""
def delete_outliers(df):
    df_outliers = df[(np.abs(stats.zscore(df)) < 3).all(axis=1)]
    df_outliers=df_outliers.reset_index()
    del df_outliers['index']
    print("Numero de outliers borrados: {}".format(len(df)-len(df_outliers)))
    return df_outliers
```

El paquete `scipy.stats` contiene un montón de utilidades matemático-estadísticas. Haciendo uso de la manera adecuada se pueden detectar *outliers* y eliminarlos.

En mi caso he borrado aquellas instancias que estaban a más de tres veces la distancia de la desviación típica respecto de la media en algún atributo. Este margen es más que suficiente, cualquier punto que se encuentre ahí es claramente un *outlier*. Para ejecutar la

función anterior simplemente:

```
# BORRANDO OUTLIERS
print("\n----- Tratando los datos que no tienen outliers -----")
df_o = delete_outliers(df)
X_o, y_o =split_data(df_o)
dendograma(X_o, "Dendograma sin outliers")
```

En la Imagen 2 observamos que se borran 10 instancias. El número total de instancias del conjunto de datos es 178, por lo que es algo que nos podemos permitir.

```
----- Tratando los datos que no tienen outliers -----
Número de outliers borrados: 10
Calculando el dendograma
```

Imagen 2: Número de *outliers* eliminados.

El dendograma obtenido sin outliers es el siguiente:

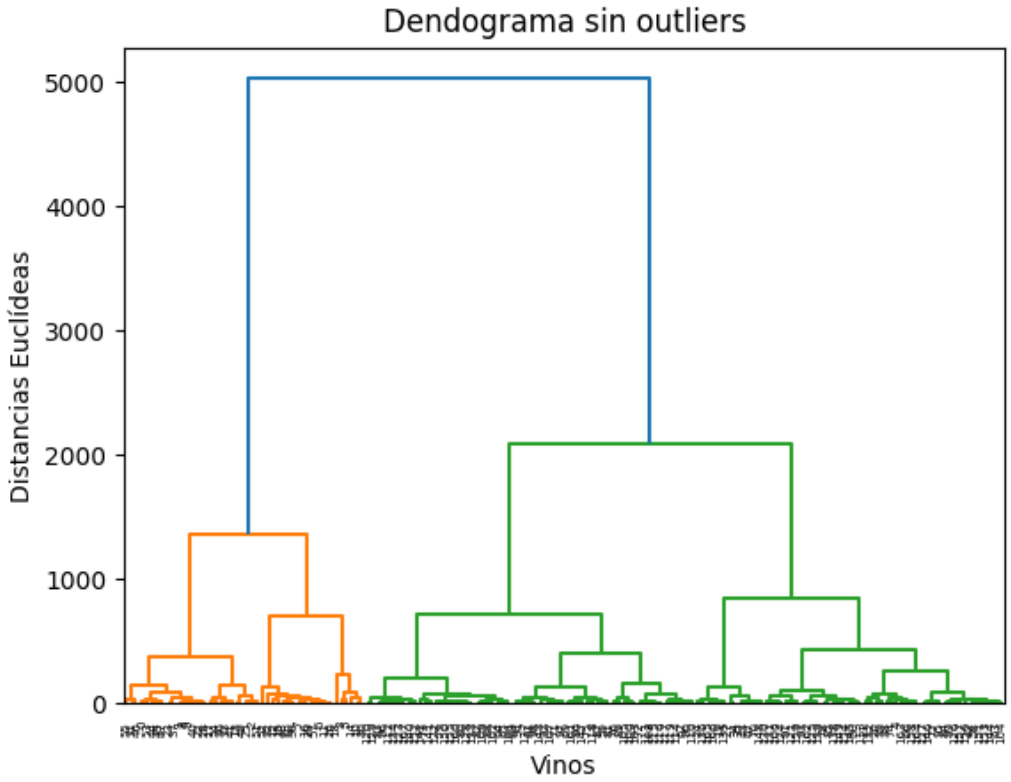

Imagen 3: Dendograma después de eliminar *outliers*

Se puede apreciar el cambio después de la eliminación, pero la decisión sigue siendo la misma: 3 *clusters*. En caso de que las cosas vayan mal se probará con 4.

4. Algoritmo k-medias

Ahora estamos en condiciones de aplicar el algoritmo k-medias a los datos con el número de *clusters* elegido. Para ello he implementado la siguiente función que usa el paquete `sklearn.cluster.KMeans`.

```
""" Aplica el clustering KMeans. Devuelve el clasificador y las etiquetas predichas.
- X: datos.
"""
def get_k_medias(X):
    kmeans = KMeans(n_clusters=3).fit(X)
    labels = kmeans.predict(X)
    print(kmeans)
    print("\nCentroides:")
    print(kmeans.cluster_centers_)
    return kmeans, labels
```

El siguiente código básicamente realiza un ploteo de los datos y en función de la clase asignada por el algoritmo k-medias le da un color. Asimismo pinta los centroides.

```
""" Aplica el clustering KMeans. Devuelve el clasificador y las etiquetas predichas.
- X: datos.
- kmeans: clasificador kmedias ya entrenado.
- labels: etiquetas predichas.
"""
def plot_cluster_primera_variable(X, kmeans, labels):
    kmeans = KMeans(n_clusters=3)
    y_kmeans =kmeans.fit_predict(X.iloc[:,0:2])
    X = X.iloc[:,0:2].values

    plt.scatter(X[y_kmeans==0, 0], X[y_kmeans==0, 1], s=10, c='red',
                label='Clase 1')
    plt.scatter(X[y_kmeans==1, 0], X[y_kmeans==1, 1], s=10, c='blue',
                label='Clase 2')
    plt.scatter(X[y_kmeans==2, 0], X[y_kmeans==2, 1], s=10, c='green',
                label='Clase 3')
    plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s=100,
                marker="*", c='black', label='Centroides')
    plt.legend()
    plt.title("Clusters de clases")
    plt.gcf().canvas.set_window_title("áPrctica 2 - Agrupamiento")
    plt.show()
```

Finalmente podemos ejecutar todo con dos simples órdenes y mostrar también algunos resultados.

```
# KMEANS
print("\n------- K-MEANS -------")
kmeans, labels = get_k_medias(X_o)
# Comparamos la distribución de las clases respecto a la primera columna.
plot_cluster_primera_variable(X_o, kmeans, labels)
# Cambiamos etiquetas para ajustar el cluster. 0->3
for i in range(len(labels)):
    if labels[i]==0: labels[i]=3
print("\nEtiquetas predichas por KMEANS:") print(labels)
print("\nEtiquetas reales:") print(np.array(y_o))
print("\nNOTA: El algoritmo KMEANS tiene un gran exito en la clasificacion")
silhouette = silhouette_score(X_o, (kmeans.labels_), metric='euclidean')
aciertos = 0
for i in range(len(labels)):
    if labels[i]==y_o[i]:
```

```
        aciertos = aciertos+1
print("\nEl accuracy es: {}".format(round(aciertos/len(labels),3)))
print("\nLa puntuacion de silhouette es: {}".format(silhouette))
```

La imagen de representación de los *clusters* con sus centroides devuelta por el procedimiento anterior es la siguiente:

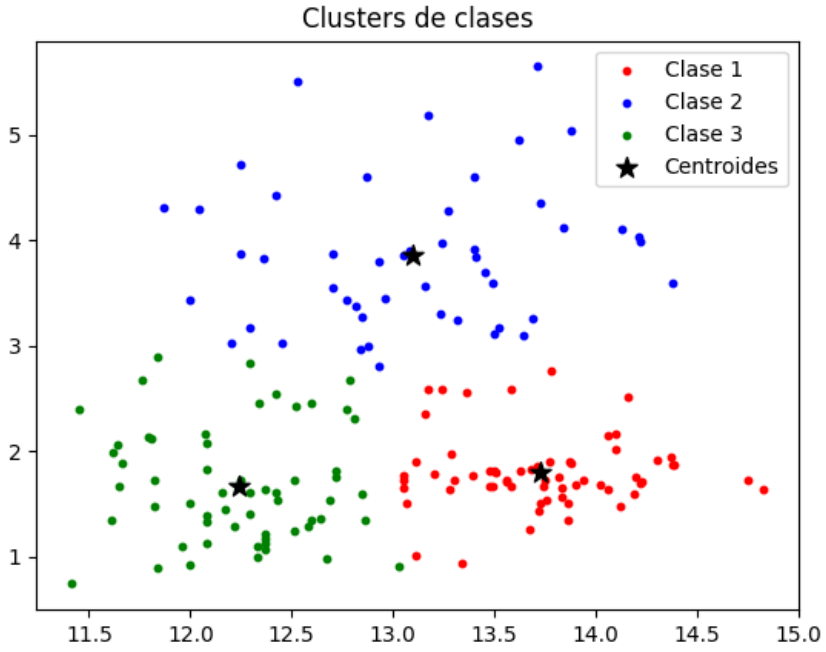

Imagen 4: *Clusters* del k-medias

Resulta visual y parece que los *clusters* están correctos. Vamos a comparar cómo se distribuyen respecto sus clases y dar el coeficiente de *silhouette*. Veamos:

```
Etiquetas predichas por KMEANS:
[1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 1 3 1 1 3 1 1 1 1 1 1 3 3 1
 1 3 3 1 1 3 3 1 1 1 1 1 1 1 1 1 1 1 1 3 2 3 3 2 2 3 2 2 3 3 2 2 3 2 2 2
 3 2 2 3 3 2 2 2 2 2 2 3 3 2 2 2 2 2 3 2 3 2 3 2 2 2 3 2 2 2 2 3 2 3 2 2 2 2
 2 2 3 2 2 2 2 2 2 2 2 3 2 2 3 3 3 3 2 2 3 3 3 2 2 3 3 2 3 3 2 2 2 2 3 3 3 2
 3 3 2 3 2 3 3 2 3 3 3 3 3 2 2 3 3 3 3 3 2]

Etiquetas reales:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]

NOTA: El algoritmo KMEANS tiene un gran éxito en la clasificación

El accuracy es: 0.714

La puntuación de silhouette es: 0.5763394801499359
```

Imagen 5: Resultados del k-medias

Un accuracy del $71,4\,\%$ es bueno, no obstante trataremos de mejorarlo más adelante.

5. Reducción de la dimensionalidad

En este apartado se van a aplicar técnicas de reducción de la dimensionalidad con el objetivo de mejorar la clasificación mediante agrupamiento. En este caso se ha optado por usar análisis de componentes principales (PCA). Posteriormente se aplicará de nuevo el algoritmo k-medias para tres *clusters*.

```
# PREPROCESANDO (StandardScaler + PCA)
print("\n------- PREPROCESANDO (StandardScaler + PCA) -------")
X_std = StandardScaler().fit_transform(X_o)
X_pca = PCA(n_components=8).fit_transform(X_std)
X_pca = pd.DataFrame(X_pca)
if(IMPRIME_INFO):
    print("Conjunto de datos despues del preprocesamiento:")
    print(X_pca)
kmeans_pca, labels_pca = get_k_medias(X_pca)
labels_pca = np.array(labels_pca)
# Comparamos la distribución de las clases respecto a la primera columna.
plot_cluster_primera_variable(X_pca, kmeans_pca, labels_pca)
# Cambiamos etiquetas para ajustar el cluster. 0->3
for i in range(len(labels_pca)):
    if labels_pca[i]==0: labels_pca[i]=3
    elif labels_pca[i]==1: labels_pca[i]=1
    elif labels_pca[i]==2: labels_pca[i]=2
print("\nEtiquetas predichas por KMEANS:")
print(labels_pca)
print("\nEtiquetas reales:")
print(np.array(y_o))
print("\nNOTA: El algoritmo KMEANS sobre los datos preprocesados tiene un
    gran exito en la óclasificacin")
silhouette_pca = silhouette_score(X_pca, (kmeans.labels_), metric='euclidean')
aciertos = 0
for i in range(len(labels_pca)):
    if labels_pca[i]==y_o[i]:
        aciertos = aciertos+1
print("\nEl accuracy es: {}".format(round(aciertos/len(labels_pca),3)))
print("\nLa puntuacion de silhouette es: {}".format(silhouette_pca))
```

Los resultados después de aplicar un escalado estándar y PCA son muy satisfactorios. Se muestra la ejecución:

```
Etiquetas predichas por KMEANS:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 1 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 3 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]

Etiquetas reales:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]

NOTA: El algoritmo KMEANS sobre los datos preprocesados tiene un gran éxito en la clasificación

El accuracy es: 0.976

La puntuación de silhouette es: 0.3293922371647779
```

Imagen 6: Resultados después de la reducción de la dimensionalidad

El accuracy obtenido es muy superior al anterior, en este caso es del $97,6\,\%$. Por tanto, hacer aplicar este tipo de técnicas resulta muy conveniente.

Para finalizar esta sección mostramos también los *clusters* obtenidos bajo este proceso en la Imagen 7.

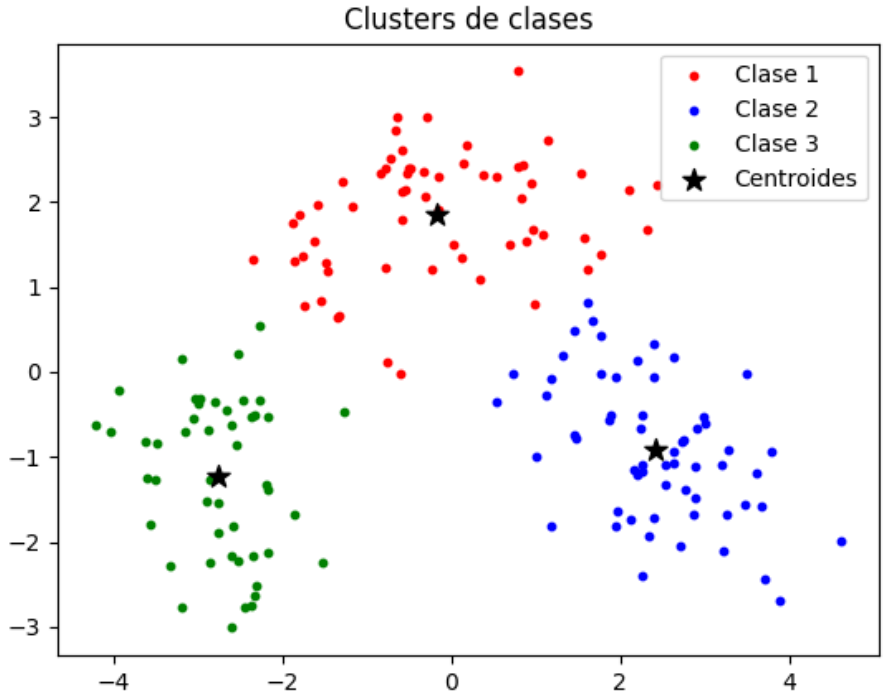

Imagen 7: *Clusters* después de la reducción de la dimensionalidad

6. Algoritmo DBSCAN

Como finalización de esta práctica se va a usar otro de los algoritmos estrella en agrupamiento, el algoritmo DBSCAN. Se ajustarán los parámetros de radio y puntos mínimos.

En Python en el paquete `sklearn.cluster.DBSCAN` está la función deseada. Tras realizar diversas pruebas, las mejores elecciones son `eps=2` y `min_samples=3`. Se trabajará sobre los datos a los que se les ha aplicado PCA.

```
print("\n------ DBSCAN ------")
db = DBSCAN(eps=2, min_samples=3).fit(X_pca)
print(db)
labs = np.array(db.labels_)
# Hacemos la asignación de labels para comparar con la primera columna:
# 2->3 1->4 0->1 -1->2
for i in range(len(labs)):
    if labs[i]==0: labs[i]=1
    elif labs[i]==2: labs[i]=3
    elif labs[i]==-1: labs[i]=2
    elif labs[i]==1: labs[i]=4
print("Etiquetas predichas por DBSCAN:")
print(labs)
print("\nEtiquetas reales:")
print(np.array(y_o))
print("\nMatriz de confusion:")
print(confusion_matrix(y_o, labs))
print("NOTA: se observan algunos outliers (cuarta clase)")
```

Procedemos a mostrar los resultados para hacer un posterior análisis. Véase la Imagen 8.

```
----- DBSCAN -----
DBSCAN(eps=2, min_samples=3)
Etiquetas predichas por DBSCAN:
[1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 4 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 1 1 1 2 2 1 2 2 2 2 1 1 1 2
 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1
 2 1 1 2 2 1 1 2 1 1 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 2 2 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3]

Etiquetas reales:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]

Matriz de confusión:
[[53  2  0  3]
 [44 18  1  0]
 [ 0  4 43  0]
 [ 0  0  0  0]]
NOTA: se observan algunos outliers (cuarta clase)
```

Imagen 8: Resultados DBSCAN

En primer lugar observamos que DBSCAN nos propone una clase `4` que parece constituida por *outliers*. También hay una clase `2` con poca frecuencia pero esta tiene una presencia suficiente como para no descartarla. Por último hay numerosas etiquetas de los tipos de vino `1` y `3`.

Comparando con las verdaderas etiquetas el accuracy obtenido es $91,95\,\%$, el cual es muy bueno. Asimismo se muestra la matriz de confusión obtenida. La precisión alcanzada por DBSCAN es excelente pero hay que resaltar que el mejor accuracy obtenido es el de

k-medias después de la reducción de la dimensionalidad.