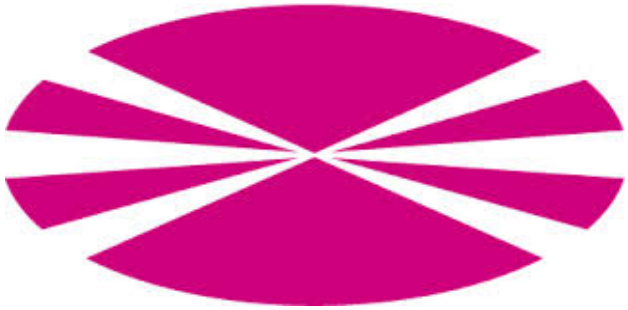
# Universidad de A Coruña



## Escuela Politécnica de Ingeniería de Ferrol

Máster Universitario en Informática Industrial y Robótica

---

# Detección de diabetes con métodos de Aprendizaje Automático

---


**Autores:**

Iván Hermida Mella

Noel Maximino Freire Mahía


**Asignaturas:**

Python para Ingenieros Avanzado

Aprendizaje Automático II

## Mayo 2025

# Índice

# 1. Introducción

## 1.1. Introducción. Objetivo del proyecto

El objetivo del presente proyecto es la aplicación de diferentes técnicas de aprendizaje automático para la obtención de modelos de clasificación que permitan realizar una predicción de la presencia de diabetes en pacientes. Este problema se afronta desde 2 perspectivas diferentes, que se plantean de forma independiente:

1. En primer lugar, se aplicarán técnicas de reducción de la dimensionalidad de los datos (PCA, ICA, etc.) como fase previa a un modelo de aprendizaje supervisado, pues se dispone de la etiqueta de la clase real para cada dato.

2. Tras esto, se aplican técnicas de agrupamiento (clustering) para plantear el problema como un problema de agrupamiento, sin considerar, por tanto, las etiquetas de clase.

Además, en el caso **1** (reducción de la dimensionalidad + clustering), se probraán diferentes modos de ejecución con la finalidad de evaluar los tiempos de ejecución en cada uno de los modos. Esto tiene sentido ya que para entrenar estos modelos se emplea validación cruzada, por lo que cada modelo es entrenado varias veces, pudiendo paralelizar estas tareas.

## 1.2. Conjunto de datos

El conjunto de datos seleccionado para este proyecto consta de 70.692 respuestas al test BRFSS2015 para la detección de diabetes. El *Behavioral Risk Factor Surveillance System* (BRFSS) es una encuesta telefónica sobre salud que los CDC realizan anualmente. Cada año, la encuesta recopila respuestas de más de 400.000 estadounidenses sobre conductas de riesgo para la salud, enfermedades crónicas y el uso de servicios preventivos.
Los datos se dividen equitativamente entre dos clases:

- **Clase 0**: Sin diabetes.
- **Clase 1**: Prediabetes / diabetes

Estos datos se contienen en un archivo CSV de 70.692 filas (nº de muestras) y 22 columnas, que representan cada una de las siguientes variables:

1. **Diabetes_binary**: Es la etiqueta de clase para cada una de las muestras. Como se explica anteriormente, existen únicamente dos clases, por lo que este valor será 0 o 1.

2. **HighBP**: Es una variable binaria que indica si el paciente presenta un valor normal de presión arterial (0) o superior al recomendado (1).

3. **HighChol**: Es una variable binaria que indica si el paciente presenta un nivel de colesterol adecuado (0) o superior al recomendado (1).

4. **CholCheck**: Es una variable binaria que indica si el paciente ha realizado una prueba de colesterol en los últimos 5 años (1) o no (0).

5. **BMI**: Es una variable numérica (de tipo entero) que indica el índice de masa corporal del paciente (cociente entre el peso en kilogramos y el cuadrado de la altura en metros del paciente).

6. **Smoker**: Es una variable binaria que indica si el paciente es fumador o no (0 - no fumador, 1 - fumador).

7. **Stroke**: Es una variable binaria que indica si el paciente ha sufrido alguna vez un *ictus*.

8. **HeartDiseaseorAttack**: Es una variable binaria que indica si el paciente sufre o ha sufrido una enfermedad cardíaca o un infarto de miocardio (0 - no, 1 - sí).

9. **PhysActivity**: Es una variable binaria que indica si el paciente ha realizado actividad física en los últimos 30 días previos al test de diabetes (0 - no, 1 - sí).
10. **Fruits**: Es una variable bianria que indica si el paciente consume frutas de forma habitual (al menos una vez al día).
11. **Veggies**: Es una variable binaria que indica si el paciente consume vegetales de forma habitual (al menos una vez al día).
12. **HvyAlcoholConsump**: Es una variable binaria que indica si el paciente presenta un consumo elevado de alcohol (0 - no, 1 - sí). En hombres adultos, se considera un consumo elevado la ingesta de más de 14 bebidas alcólicas semanales, que se reduce a 7 en el caso de mujeres adultas.
13. **AnyHealthcare**: Es una variable binaria que indica si el paciente cuenta con algún tipo de cobertura de atención médica o seguro médico (0 - no, 1 - sí).
14. **NoDocbcCost**: Es una variable binaria que indica, en caso de que valga 1, que el paciente ha necesitado acudir a un médico en los últimos 12 meses pero no lo ha hecho debido al coste.
15. **GenHlth**: Es una variable numérica. Su valor viene "estimado" directamente por el paciente, en respuesta a como considera el mismo su estado de salud:
    - 1 $\Longrightarrow$ Excelente
    - 2 $\Longrightarrow$ Muy bueno
    - 3 $\Longrightarrow$ Bueno
    - 4 $\Longrightarrow$ Regular
    - 5 $\Longrightarrow$ Malo
16. **MentHlth**: Es una variable numérica (de tipo entero) con valor entre 1 y 30 que indica el número de días mensuales que el paciente calificaría su estado de salud mental como "pobre" o "malo".
17. **PhysHlth**: Es una variable numérica (de tipo entero) que indica el número de días mensuales (1-30) en que el paciente presenta dolor por alguna enfermedad o lesión física.
18. **DiffWalk**: Es una variable binaria que indica si el paciente presenta dificultades para andar o subir escaleras (0 - no, 1 - sí).
19. **Sex**: Variable binaria que indica el sexo del paciente (0 - femenino, 1 - masculino).
20. **Age**: Es una variable numérica (de tipo entero), que indica la edad del paciente, aunque no directamente, si no a través de una escala de 13 niveles (AGEG5YR).
21. **Education**: Variable numérica (de tipo entero) que indica, con una escala entre 1 y 6 (EDUCA), el nivel de estudios del paciente.
22. **Income**: Variable numérica (de tipo entero) que indica, con una escala entre 1 y 8 (INCOME2), el nivel de ingresos del paciente.

El conjunto de datos se ha extraído del repositorio Kaggle, y es accesible mediante la siguiente URL: `https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset?select=diabetes_binary_5050split_health_indicators_BRFSS2015.csv`

### 1.3. Estructura del programa en Python

A la hora de implementar el programa en Python, se ha dividido el código en 14 módulos diferentes que cumplen funciones diferenciadas.
El módulo principal es **main.py**, desde donde se llama a los diferentes módulos del programa.
En el módulo **subir_datos.py** se define la función *cargar_datos(path,target)*, que a partir del nombre del archivo CSV y el nombre de la columna que contiene las etiquetas permite importar los datos.
Una vez que se han importado los datos, aquellas variables que presenten valores numéricos se normalizan con media 0 y desviación típica 1, a través de la función *normalizar_datos(x)*, definida en el módulo **normalizar.py**.
En el módulo **Seleccion_ejecucion.py** se definen las funciones necesarias para la ejecución secuencial, multihilo y multiproceso de ciertas secciones de código.
Para realizar la clasificación no supervisada mediante clustering se emplea el módulo **clustering.py**, donde se definen las funciones para aplicar los diferentes modelos. Los resultados del clustering se guardan en un archivo Excel empleando el módulo **metricas_clustering.py**. Los resultados de esta clasificación no supervisada se muestran gráficamente empleando el método t-SNE (t-Distributed Stochastic Neighbor Embedding) empleando el módulo **tsne.py**.
Para la reducción de la dimensión de los datos de entrada y la posterior clasificación supervisada se emplean los 7 módulos restantes:

- **Principales_caracteristicas.py** $\Longrightarrow$ Contiene la función que aplica el análisis de componentes principales al conjunto de datos.
- **Matriz_correlacion.py** $\Longrightarrow$ Contiene la función para calcular y representar la matriz de correlación entre las variables originales y la variable de salida.
- **Modelos.py** $\Longrightarrow$ Contiene las funciones para definir los diferentes modelos de clasificación supervisada al conjunto de datos.
- **entrenar_modelos.py** $\Longrightarrow$ Se definen las funciones para realizar el entrenamiento de los diferentes modelos definidos.
- **guardar_metricas.py** $\Longrightarrow$ Permite escribir en un archivo Excel los resultados obtenidos por los diferentes modelos.
- **leer_metricas.py** $\Longrightarrow$ Contiene las funciones necesarias para leer los archivos Excel generados por el módulo anterior y representar gráficamente los resultados.
- **comparacion_n_covariables.py** $\Longrightarrow$ Contiene las funciones para aplicar el test estadístico entre los diferentes modelos empleados y comparar sus resultados.

En la figura 1 se muestra de forma esquemática la estructura del programa implementado en Python.

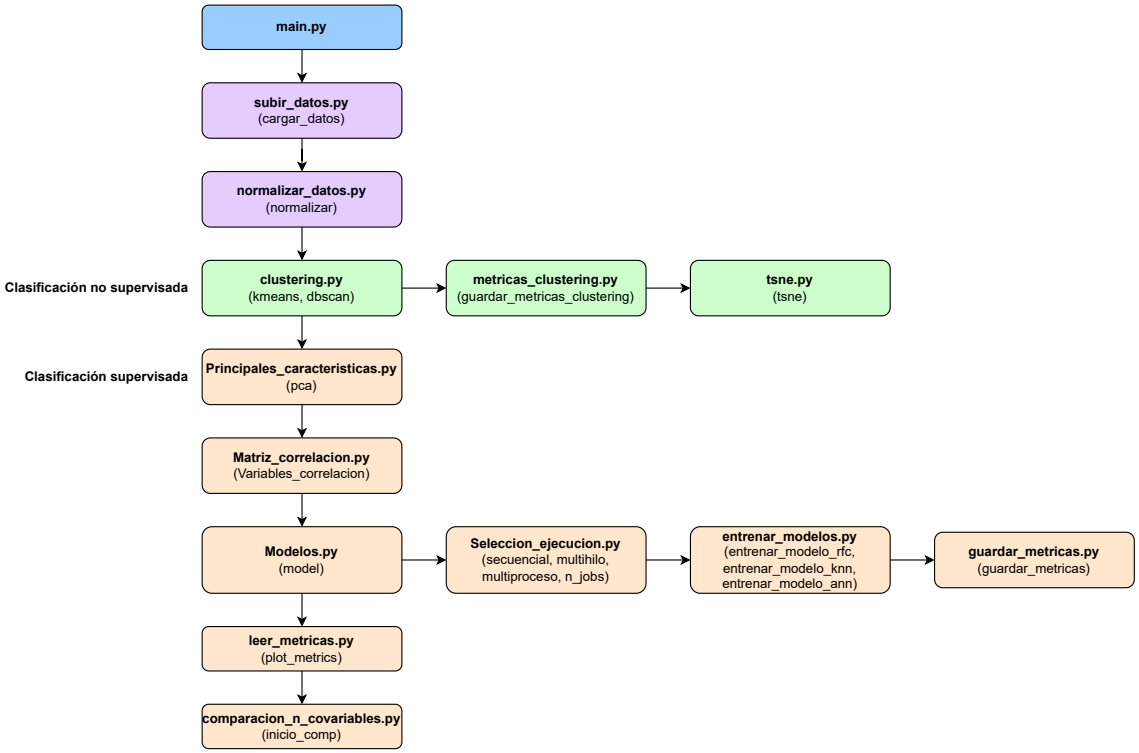



Figura 1: Esquema de la estructura del programa

## 1.4. Modos de ejecución

Como se ha explicado previamente, se van a probar diferentes formas de paralelización de procesos en el entrenamiento de los modelos de aprendizaje supervisado para ver si generan una mejor significativa en los tiempos de ejecución. Concretamente, se van a probar 4 formas de ejecución:

- **Secuencial**. El código se ejecuta paso a paso, de forma lineal, en un solo hilo y en un solo núcleo de CPU. Cabe esperar que sea el que obtenga los peores tiempos de ejecución, pues no aprovecha los múltiples núcleos del procesador y es más lento para tareas que podrían paralelizarse, como el entrenamiento en paralelo de modelos con diferentes hiperparámetros.
- **Multihilo**. Ejecuta múltiples hilos dentro de un mismo proceso. Debido al GIL (Global Interpreter Lock) de Python, los hilos no se ejecutan realmente en paralelo para tareas como el entrenamiento de modelos, sólo uno se ejecuta a la vez.
- **Multiproceso**. Lanza procesos independientes, cada uno con su propio intérprete de Python y espacio de memoria.
- **n_jobs**. Permite especificar cuántas tareas paralelas se deben ejecutar al entrenar modelos, hacer búsqueda de hiperparámetros, etc.

## 1.5. Métricas de evaluación

Una vez creados los modelos, se evaluará su calidad utilizando una serie de test e indicadores. Puesto que en este trabajo se realizaron 2 tipos de modelado, para cada uno se usaron métricas distintas para ver su rendimiento.

### 1.5.1. Clasificación supervisada

En estos modelos, al ser de clasificación binaria, no hace falta trabajar con el macropromedio, es decir, que no se necesita hacer ningún preprocesado adicional de los resultados obtenidos. A continuación se explicará que es cada métrica y su fórmula.

### Accuracy o exactitud

Es el nivel de coincidencia entre el valor predicho del modelo y su valor real o verdadero [1]. En este caso al tener 2 clases con la misma cantidad de datos esta medida será la más importante para ver la calidad del modelo y se utilizará como principal indicador para ver que modelo es el más óptimo para la correcta clasificación.

$$\text{Accuracy}_i \text{ o exactitud}_i = \frac{\text{Predicciones correctas}_i}{\text{Total de predicciones}_i} = \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i} \tag{1}$$

### Precision o precisión

Grado de coincidencia entre los valores obtenidos al realizar mediciones repetidas de una misma magnitud, bajo condiciones específicas

$$\text{Precision}_i = \frac{\text{Aciertos de los casos clasificados como positivos}_i}{\text{Casos clasificados como positivos}_i} = \frac{TP_i}{TP_i + FP_i} \tag{2}$$

### Recall o Sensibilidad

Medida utilizada en clasificación para indicar la capacidad del modelo para identificar correctamente las instancias positivas.

$$\text{Recall}_i = \frac{\text{Aciertos de los casos positivos}}{\text{Casos positivos}_i} = \frac{TP_i}{TP_i + FN_i} \tag{3}$$

### F1-Socre

"Es la media armónica entre la precisión y la sensibilidad."[2] Esta métrica tiene mucha importancia en los datasets donde los grupos sean muy desproporcionados. Como este no es el caso, aunque se usa para ver la calidad de los diferentes modelos, no es la métrica más importante.

$$\text{F1-score}_i = \frac{2 \cdot \text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{4}$$

## Curvas ROC globales y por clase

Herramienta que muestra gráficamente en el eje Y el TPR (True Positive Rate) y en el X el NFR (False Positive Rate), de esta gráfica se puede evaluar visualmente el rendimiento de un modelo de clasificación.
Aunque se pueden generar curvas ROC por clase en problemas multiclase, en este trabajo, al tratarse de un problema binario, se utilizará únicamente la curva ROC global. En este caso se hizo con el micro-avg, por lo que en lugar de usar la media, se ponderan las clases. Es decir se suman todos los valores (FP,VP,FN y VN) y se obtiene la métrica basada en todas las clases, donde las clases con más muestras tendrán una mayor importancia (en este caso 50-50)
Lo más importante de estas gráficas es el valor del AUC (Area Under the Curve), pues cuanto mayor sea esta, es decir más próxima a 1, mejor será el modelo.

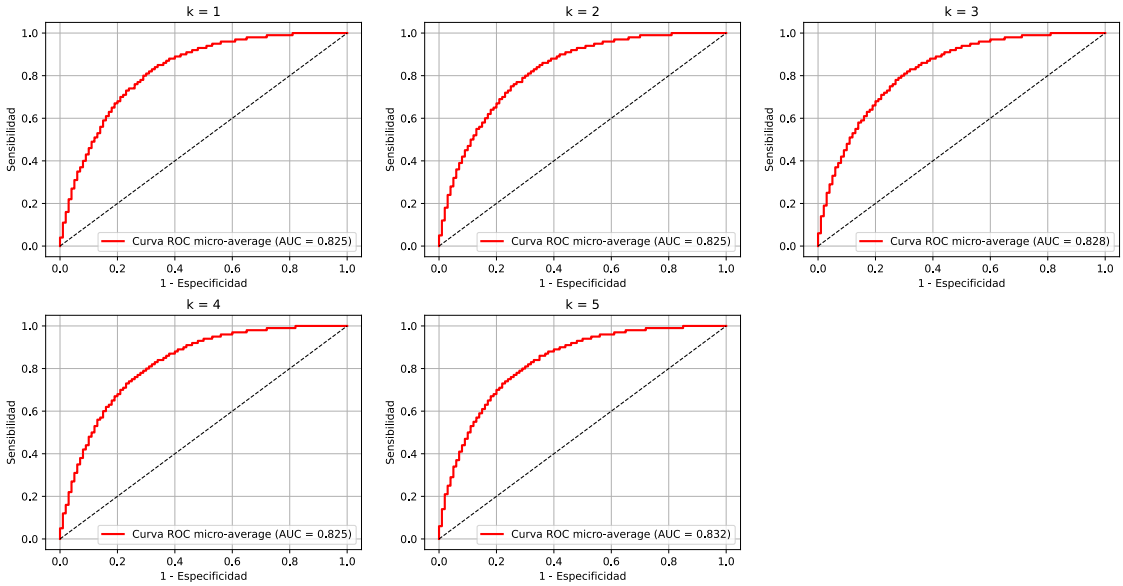



Figura 2: Ejemplo gráficas ROC

Lo que se puede ver aquí es como cambia el valor del TPR (eje Y) y del NFR (eje X) según va cambiando el valor del umbral, pues cuando se selecciona un valor muy pequeño el TPR es 1, pero el valor de NFR también suele ser muy alto, y según va aumentando el valor de este ambas variables van disminuyendo, pero lo importante es que TPR baje más lento que el NFR.

## KrusKal Wallis y Tukey HSD

Para finalizar las métricas de la sección de aprendizaje supervisado y saber que modelo se debe seleccionar, se realiza el test de KrusKal-Wallis, que nos indica si existe alguna diferencia estadística entre los modelos que se probaron. Este test evalúa la hipótesis nula de que todas las muestras provienen de la misma distribución. Se compara el p-valor con 0.01, y si este es menor se rechaza la hipótesis nula y se asegura con un 99 % que al menos uno de los modelos los modelos comparados son distintos.
Si se rechaza esta hipótesis se realiza la prueba de Tukey, la cual nos da una tabla comparando todos los modelos entre si e indicando cuales son estadísticamente iguales. A continuación se explican todas las columnas que aparecen en la tabla.

### Meandiff

Diferencia en las medias entre los dos grupos que se comparan

### P-adj

Indica que la diferencia observada entre los grupos. Cuanto más pequeño sea el valor, mayor será la diferencia

### Lower

Límite inferior del intervalo de confianza para la diferencia de medias entre los grupos

### Upper

Límite superior del intervalo de confianza para la diferencia de medias entre los grupos

### Reject

Valor booleano que indica si se rechazo o no la hipótesis nula

### 1.5.2. Clasificación no supervisada

Para evaluar los modelos de clasificación no supervisada (k-medias y DBSCAN) se han de emplear métricas de evaluación diferentes, pues estos modelos trabajan sin tener en cuenta las etiquetas de clase real de los datos de entrada. Como, aunque no se utilicen en el entrenamiento del modelo, sí se dispone de las etiquetas, podemos utilizar métricas de evaluación **extrínseca**. Las métricas que se van a emplear son la Medida V (V-Measure) y el índice de Rand ajustado (ARI, Adjusted Rand Index).

#### Medida V

La **medida V** establece la media armónica entre homogeneidad y completitud (o integridad).

- **Homogeneidad**: Cada grupo contiene sólo miembros de una única clase.
- **Completitud**: Todos los miembros de una clase están en el mismo grupo.

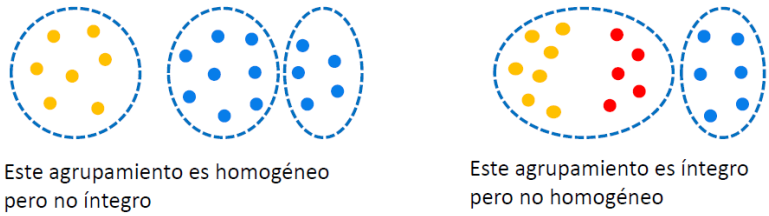



Figura 3: Concepto de homogeneidad e integridad

La expresión de la medida V es, por tanto:

$$V_{measure} = \frac{(1+\beta) \cdot homogeneidad \cdot integridad}{\beta \cdot homogeneidad \cdot integridad}$$

Donde $\beta$ es la relación de peso atribuido a cada elemento.

- $\beta = 1 \Longrightarrow$ Homogeneidad e integridad tienen el mismo peso.
- $\beta > 1 \Longrightarrow$ La integridad se pondera más fuertemente.
- $\beta < 1 \Longrightarrow$ La homogeneidad se pondera más fuertemente.

En la implementación *v_measure_score* de Scikit-Learn que se va a utilizar, el valor por defecto de $\beta$ es 1, que es un valor coherente para un conjunto de datos balanceado como el de este caso.
Los valores de homogeneidad y completitud se calculan a partir de la entropía entre las etiquetas verdaderas y las predichas.

Los valores de esta medida estarán entre 0 y 1, donde 1 implica un etiquetado perfecto.

#### Índice de Rand ajustado

El **índice de Rand** proporciona una medida de similitud entre dos agrupamientos (X e Y) considerando todos los posibles pares de muestras y contando los pares que se asignan en el mismo o en diferentes grupos.

| X/Y | Y: pares en el mismo grupo | Y: pares en grupos diferentes |
|---|---|---|
| X: pares en el mismo grupo | $a$ | $b$ |
| X: pares en grupos diferentes | $c$ | $d$ |

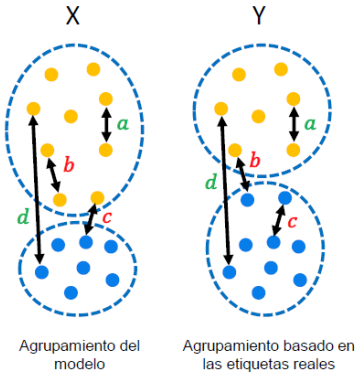



Figura 4: Índice de Rand (RI)

El índice de Rand se define como:

$$RI(X,Y) = \frac{a+d}{a+b+c+d}$$

Sin embargo, el índice de Rand no tiene en cuenta el posible acuerdo por casualidad lo que provoca que suela estar cerca de 1 incluso si los agrupamientos difieren significativamente.

El **Índice de Rand ajustado (ARI)** corrige por el azar, haciendo que puntuaciones aleatorias den un valor cercano a 0.

$$ARI(X,Y) = \frac{RI(X,Y) - E[RI]}{max(RI) - E[RI]}$$

Esta medida es simétrica, y su valor puede estar entre -1 y 1. De esta forma, el ARI asigna puntuación de 1 a la clasificación perfecta, y puntuaciones cercanas a 0 para clasificaciones aleatorias. Una puntuación inferior a 0 implicaría una clasificación peor que la aleatoria.

# 2. Reducción de la dimensionalidad. Clasificación supervisada

## 2.1. Modelos de clasificación

En esta sección se realizará una breve explicación teórica de como funciona individualmente cada modelo, los parámetros seleccionados, las métricas del mejor/mejores modelos, el método de extracción de caractísticas y número de estas con las que se trabajó.

### 2.1.1. RFC (Random Forest Classifier)

En este modelo se crean varios arboles de decisión no relacionados entre si (independientes), entrenados con una muestra aleatoria con reemplazamiento (boostrap), que posteriormente forman un "comité de expertos" y por votación mayoritaria deciden que etiqueta usar.[3]
La principal ventaja de este modelo basado en arboles de decisión es el tema de la explicabilidad. Estos son capaces de representar gráficamente el proceso que siguen para tomar una decisión, es decir no es una caja negra. A nivel médico (nuestro caso) es un gran punto a favor, pues el doctor debe saber en que basarse para diagnosticar y este algoritmo le puede destacar o proporcionar información en la cual el doctor no haya hecho el suficiente énfasis o no se haya percatado. Aquí se muestra un ejemplo de como es este árbol.

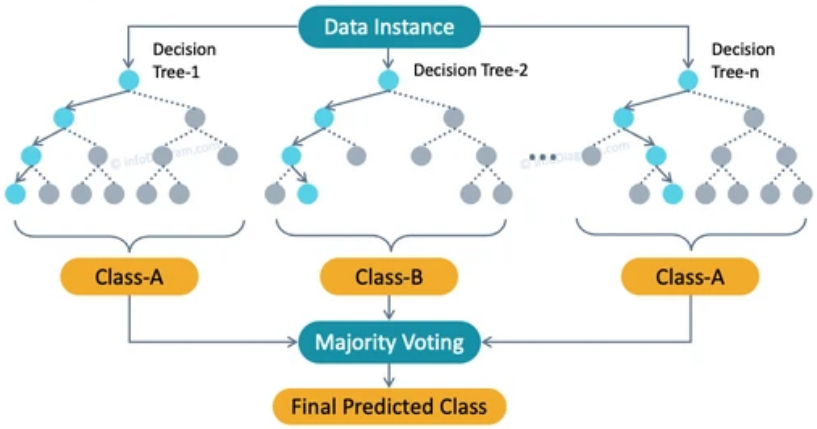



Figura 5: Funcionamiento del Random Forest

A continuación se muestran los hiperparámetros que se modificaron en el modelo de RFC y una pequeña explicación de cada uno.

1. **n_estimators**.Número de árboles que construyen el modelo. Cuantos más arboles mejor debería ser el modelo, pero mayor coste computacional.
2. **max_depth**.Número de niveles que puede tener un árbol como máximo.
3. **n_jobs**.Número de núcleos usados en paralelo
4. **criterion**. Función que indica la calidad de la división de los datos.
5. **max_features**. Indica cuántas covariables se consideran para buscar la mejor división en cada nodo de un árbol.
6. **bootstrap**.Valor booleano que indica si se usa muestreo con reemplazo para los árboles.
7. **random_state**. Semilla para aleatoriedad. Se usa para obtener siempre los mismos resultados. [4]

```
parametros_RFC = {
        'n_estimators': [1000],
        'max_depth': [5],
        'n_jobs': [1],
        'criterion': ['gini'],
        'max_features': ['sqrt'],
        'bootstrap': [True],
        'random_state': [0]
    }
```

### 2.1.2. KNN (K-Nearest Neighbors)

Se utilizó el siguiente modelo por varios motivos, el primero de ellos es que la velocidad de entrenamiento de este modelo es mucho menor que la de sus competidores, pues se trata de un algoritmo perezoso (lazy), es decir que no construye ningún modelo, simplemente almacena los datos y realiza la clasificación cuando llega un nuevo dato [5]. Esto puede resultar interesante para comprobar el funcionamiento del multihilo y multiproceso en este modelo, pues si solo almacena datos, la forma de ejecución no debería afectar mucho al tiempo de " entrenamiento". El otro motivo es que este modelo internamente realiza un proceso muy parecido al clustering, pues ambos usan la distancia a un centroide para realizar una clasificación aun determinado grupo. Dados los buenos resultados (como se verán en las siguientes secciones) se opto por probar varios modelos de clasificación no supervisada que no tuvieron los resultados esperados. A continuación se muestra una gráfica de como funciona el proceso de clasificación del KNN.

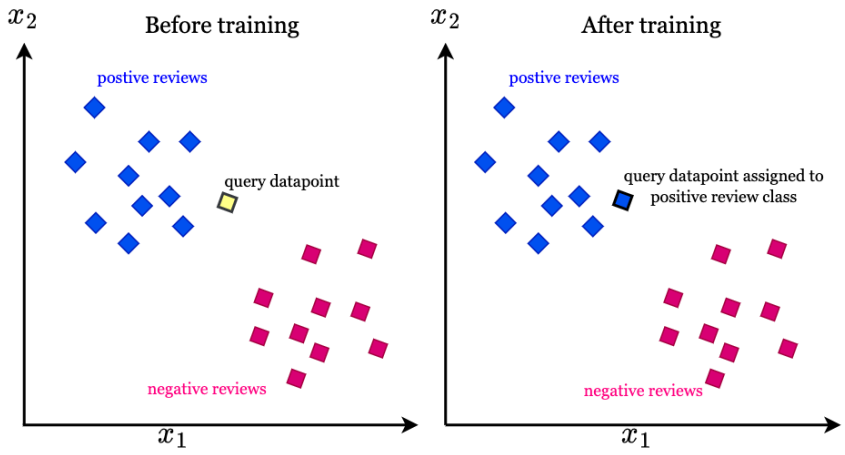



Figura 6: Funcionamiento del KNN

Por último se explicarán los hiperparámetros seleccionados en este modelo.

1. **n_neighbors**.Número de vecinos cercanos que el modelo usa para predecir una clase. Se usó la formula $k < \sqrt{n}$, siendo n el número de datos de entrenamiento, en este caso $k < \sqrt{70693} \approx 265$. Por lo que se probaron con varios valores, destacando k=7 y k=260, no tanto por su diferencia en las métricas de validación si no por su gran diferencia numérica.

| Fold | Accuracy | Precision | Recall | F1_Score |
|---|---|---|---|---|
| 1 | 0.7219 | 0.7229 | 0.7219 | 0.7216 |
| 2 | 0.7211 | 0.7220 | 0.7211 | 0.7208 |
| 3 | 0.7208 | 0.7219 | 0.7208 | 0.7204 |
| 4 | 0.7096 | 0.7111 | 0.7096 | 0.7092 |
| 5 | 0.7191 | 0.7205 | 0.7191 | 0.7186 |

Cuadro 1: Resultados de validación cruzada con k=7

| Fold | Accuracy | Precision | Recall | F1_Score |
|---|---|---|---|---|
| 1 | 0.7469 | 0.7509 | 0.7469 | 0.7459 |
| 2 | 0.7493 | 0.7538 | 0.7493 | 0.7482 |
| 3 | 0.7444 | 0.7480 | 0.7444 | 0.7434 |
| 4 | 0.7304 | 0.7345 | 0.7304 | 0.7292 |
| 5 | 0.7479 | 0.7530 | 0.7479 | 0.7466 |

Cuadro 2: Resultados de validación cruzada con k=260

Viendo estas tablas se selecciono una k de 260.Se vio que cuanto más aumentaba el valor de k mejores eran los resultados, pero nunca se probo a sobrepasar el valor de 260 por lo visto anteriormente.

2. **weights**. Formato en el que pondera el valor de cada vecino. En este caso los vecinos más cercanos tendrá un mayor peso.

3. **algorithm**. Algoritmo para calcular los vecinos más cercanos, al no tener claro cual usar se indico que elija el propio modelo de forma automática.

4. **leaf_size**. Parametro que principalmente sirve para ajustar la eficacia del modelo

5. **metric**.Métrica usada para el calculo de las distancias. Se usa la que viene por defecto, es decir la minkowski.

6. **n_jobs**.Cantidad de núcleos disponibles para usar.

7. **p**.Parámetro para la métrica Minkowski. En este caso se usara la euclidiana (p=2). [6]

Aquí se muestra el código implementado en el trabajo con la selección de los hiperparámetros.

```
parametros_KNN = {
    'n_neighbors':[260],
    'weights':['distance'],
    'algorithm':['auto'],
    'leaf_size':[40],
    'metric':['minkowski'],
    'n_jobs':[1],
    'p':[2  ]
}
```

### 2.1.3. ANN (Artificial Neuronal Network)

El último modelo con el que se trabajo fue una red neuronal artificial, pues se además de ser uno de los modelos más comunes, se pensó un poco en la parte de tiempos y aquí al no tener el parámetro de n_jobs, se podría ver de una manera más clara la diferencia entre los 3 tipos de ejecución: secuencial, multihilo y multiproceso.

El número de capas y de neuronas por capa se decide de manera prácticamente arbitraria, intentando obtener un compromiso entre precisión y tiempo de ejecución del modelo, y respetando que la última capa debe tener tantas neuronas como clases existen en el dataset (4 en este caso). Para la entrada se tiene que usar una variable, pues el tamaño cambia según el número de componentes extraídos del PCA o el ICA.La estructura de la red utilizada se muestra en la figura **??**.

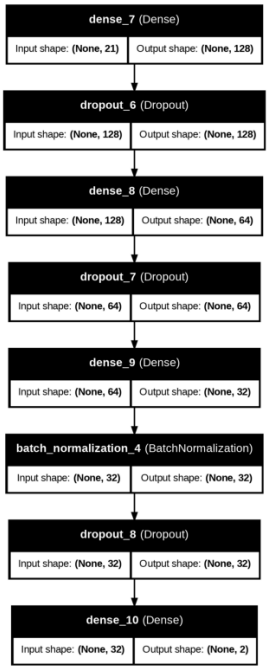



Figura 7: Estructura de la red de neuronas artificiales sin extractor (21 componentes)

## 2.2. PCA (Principal Component Analysis)

Técnica para reducir la dimensionalidad de un conjunto de datos maximizando el valor de la varianza. Gracias a esta reducción se consigue:

1. Eliminar la redundancia en algunas covariables.
2. Simplificar las operaciones computacionales.
3. Visualización más sencilla.

El nuevo grupo de variables conocido como componentes principales es una combinación lineal de las variables originales,de manera que el primer componente tiene el mayor valor de varianza, el segundo tiene la siguiente mayor varianza, etc. Cabe destacar que todos estos componentes son ortogonales, por lo que no existe ninguna correlación entre ellos [7].
A continuación se puede ver la gráfica de las componentes del PCA del dataset usado, siendo el eje y el porcentaje de varianza y en el eje x el componente.

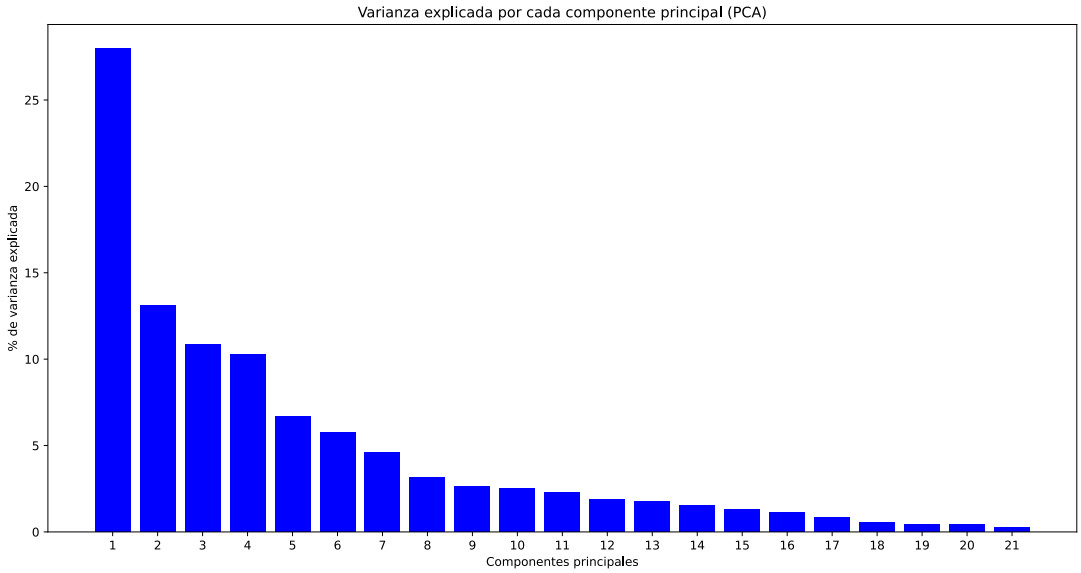
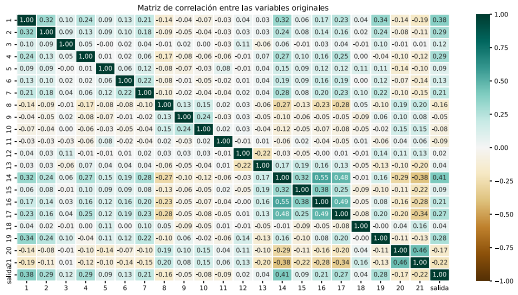
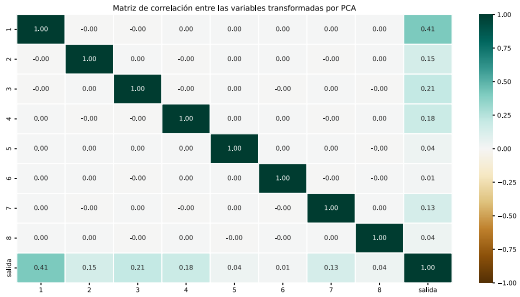



Figura 8: PCA con 21 componentes

Viendo este gráfico se optó por realizar varias pruebas con 4, 8 y 11 componentes, pues se considera que existe un pequeño escalón en el nivel de la varianza entre esos componentes y los siguientes. Para finalizar con este apartado se realiza una prueba para ver si realmente los componentes principales son ortogonales. Se seleccionan los 8 primeros valores y se calcula la matriz de correlación, ademas se compara con la obtenida de las variables originales.




(a) Matriz de correlación de todas las variables originales




(b) Matriz de correlación de los 8 primeros componentes del PCA

Figura 9: Comparación de las matrices de correlación

Se puede observar que, a pesar de que las variables originales presentan un alto nivel de correlación entre sí 9a, los componentes principales obtenidos mediante PCA son estadísticamente independientes entre ellos. Esto se puede ver en la matriz 9b , donde todos los elementos fuera de la diagonal y de la última fila/columna son ceros, indicando ausencia de correlación. La diagonal, compuesta por unos, representa la autocorrelación de cada componente consigo misma. Cabe destacar que se ha incluido la variable de salida en el análisis para visualizar su relación con los componentes principales en la última fila o columna de la matriz.

## 2.3. ICA (Independent Component Analysis)

La técnica ICA se usa para reducir la dimensionalidad de un conjunto de datos, como PCA, pero al contrario de esta los componentes de ICA (como cuyo nombre indica) no dependen unos de otros, es decir que los componentes no comparten ninguna dependencia, ni siquiera no lineal, es decir que son variables totalmente independientes unas de otras. En esta imagen se puede ver un ejemplo de como trabaja PCA e ICA.

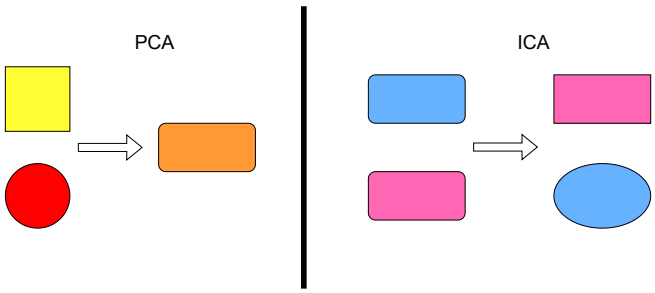



Figura 10: Diferencia entre PCA e ICA

Con esta imagen se intenta explicar como PCA trata de combinar varias covariables en una única (en esta caso fusionando forma y color), en cambio ICA es capaz de dividir una componente en varias (en este ejemplo la forma)

# 3. Clustering. Clasificación no supervisada

Tras aplicar diferentes modelos de clasificación supervisada al conjunto de datos, en vista de los decentes resultados obtenidos con el modelo kNN (cuyo algoritmo es similar a algunos algoritmos de clasificación no supervisada, como puede ser k-medias o los modelos basados en densidad) se decide aplicar técnicas de aprendizaje no supervisado para ver cuanto empeora la calidad de la clasificación al no disponer de las etiquetas de clase reales.

## 3.1. Agrupamiento por particiones: k-medias

El método k-medias (*k-means*) es un algoritmo de clustering no supervisado que se utiliza para agrupar un conjunto de datos en $k$ grupos o clusters, de forma que este número $k$ de clusters es un parámetro que se ha de introducir al modelo y que ha de ser especificado por el usuario.
El algoritmo divide los datos en grupos buscando que:

- Los datos dentro de un grupo sean lo más similares posible entre si.
- Los grupos estén lo más separados posible entre si.

Para definir estos grupos, se establecen tantos centroides como grupos se deseen ($k$). Estos centroides se reubican de forma iterativa, de forma que se reasignan los datos a un grupo de forma iterativa partiendo de un agrupamiento inicial.

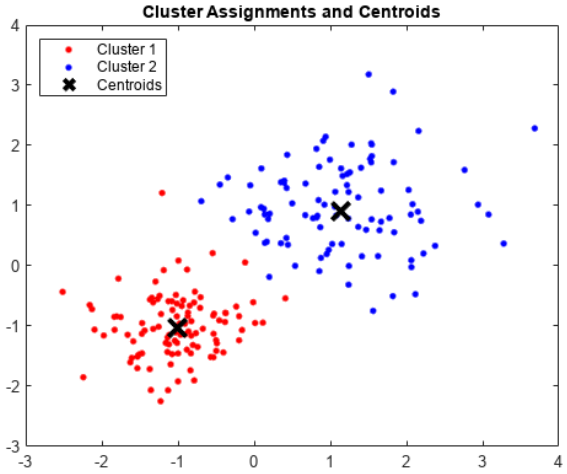



Figura 11: Clustering para un conjunto de datos de dos dimensiones

El funcionamiento secuencial de este algoritmo funciona de la siguiente forma:

1. Se eligen $k$ puntos aleatorios del conjunto como centroides iniciales. Existen algoritmos que permiten obtener unos mejores valores iniciales para los centraoides, como **k_means++**, que es el empleado en este proyecto y que se explicará a continuación.
2. Cada punto del dataset se asigna al centroide más cercano según la distancia euclídea:

$$d(x, y) = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}$$

3. Se actualiza la posición de los centroides. Esta nueva posición será el promedio de todos los puntos asignados a su clúster.
4. Se repite iterativamente este proceso hasta que los centroides no varíen o hasta que se alcace el número máximo de iteraciones definido por el usuario.

El algoritmo k-medias se implementa en Python con la librería Scikit-Learn, de forma que se deben tener en cuenta una serie de hiperparámetros para obtener los mejores resultados posibles.

- **n_clusters**: Es el valor de $k$, el número de clusters en los que se desea dividir el conjunto de datos.
  En este caso concreto, sabemos que solo existen 2 clases al contar con la etiqueta de cada dato. Sin embargo, como estamos planteando un problema de clasificación no supervisada, vamos a utilizar algún método para determinar este número de clusters.
  En la figura 12 podemos ver como varían la inercia y el coeficiente de silueta en función de $k$.

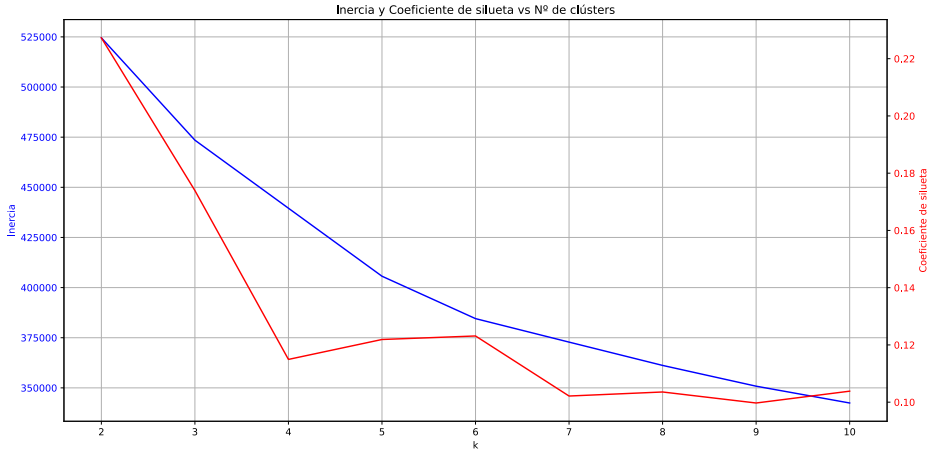



Figura 12: Inercia y coeficiente de silueta en función del nº de clusters

Uno de los métodos mas habituales para obtener el valor de $k$ es el método *elbow*, que consiste en seleccionar el valor de $k$ a partir del cual la inercia no disminuye significativamente (el valor de $k$ que forma un "codo" en la gráfica). Sin embargo, podemos observar que en este caso no existe ningún "codo" claramente definido en la gráfica que representa los valores de inercia.

Por lo tanto, nos vamos a basar en el valor del coeficiente de silueta para cada valor de $k$.
El coeficiente de silueta se utiliza para estudiar la distancia de separación entre los grupos resultantes. Se calcula utilizando la distancia media dentro del grupo ($a$) y la distancia media del grupo más cercano ($b$) para cada muestra:

$$s = \frac{b - a}{max(a, b)}$$

- **a**: distancia media entre una muestra y todos los demás puntos del mismo grupo.
- **b**: distancia media entre una muestra y todos los demás puntos en el siguiente grupo más cercano.

Los valores de esta medida están en el rango [-1, 1]. Una puntuación alta de este coeficiente indica un modelo con clústeres mejor definidos.

En la figura 12 podemos ver que el mejor valor de coeficiente silueta se obtiene para $k = 2$, por lo que será este el valor que utilizaremos para **n_clusters**. Podemos apreciar que, de hecho, coincide con el número de clases reales que sabemos que existen.

- **init**. Este hiperparámetro define la forma en la que se inicializan los centroides. Disponemos de varias opciones para establecer este parámetro:

  - **random**: Selecciona como centroides iniciales valores aleatorios del conjunto de datos. Puede llevar a soluciones poco precisas si los centroides iniciales están muy cerca.

- **kmeans++**: Elige los centroides iniciales de forma inteligente y dispersa, lo que mejora la convergencia. Es la opción recomendada en la mayoría de los casos, y es la que se va a emplear en este caso.
- También podemos directamente proporcionarle un array de forma manual con los puntos que queremos que utilice como centroides iniciales.

### 3.2. Agrupamiento basado en densidad: DBSCAN

El algoritmo DBSCAN (Density-Based Spatial Clustering of Applications with Noise) es un algoritmo de clustering basado en densidad, de forma que agrupa puntos que estén cerca unos de otros sin necesidad de tener que especificar el número de clusters a obtener. También permite detectar valores atípicos (*outliers*).
Este algoritmo establece la densidad de cada punto como el número de datos que están dentro de un radio específico, dado un hiperparámetro *epsilon*.
Dependiendo del valor de densidad, y de un hiperparámetro que indica un número mínimo de puntos (*minPts*), cada punto se clasifica en uno de 3 tipos posibles:

- **Central**: Tiene al menos *minPts* puntos en su vecindad.
- **Frontera**: No cumple con tener al menos *minPts* puntos en su vecindad, pero forma parte del vecindario de algún punto central.
- **Outlier**: No es punto central ni frontera.

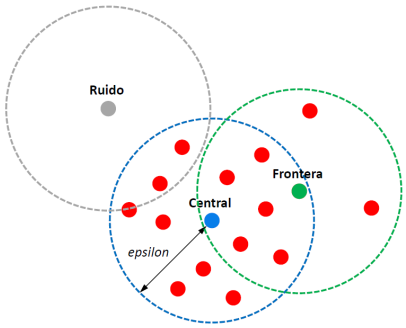



Figura 13: Tipos de puntos considerados por el algoritmo DBSCAN

Por lo tanto, para la implementación de este algoritmo en el proyecto con Scikit-Learn se ha de ajustar el valor de 2 hiperparámetros:

- **min_samples**: Es el equivalente a minPts, es el número mínimo de puntos dentro del radio (incluyendo el propio punto) para considerar que un punto es central.
  En este proyecto se utiliza la regla heurística propuesta por Sander et al., 1998 [8], que establece:
  $$min\_samples = 2 \cdot dimensión$$
  En este caso, el conjunto de datos de entrada es de 21 variables, por lo que se toma un valor de:
  $$min\_samples = 2 \cdot 21 = 42$$
- **eps**: Es el radio de vecindad alrededor de un punto, define qué tan cerca deben estar los puntos entre sí para ser considerados vecinos.
  Para determinar el valor de *eps* a emplear se utiliza el concepto de la k-distancia. La k-distancia de un punto se define como la distancia al k-ésimo punto más cercano.
  1. Se calcula la k-distancia de cada punto, siendo k = min_samples
  2. Se ordenan las k-distancias de menor a mayor para mostrarlas gráficamente.

En este caso, se calcula la 42-distancia. En la figura 14 se muestran gráficamente estas 42-distancias ordenadas de menor a mayor.

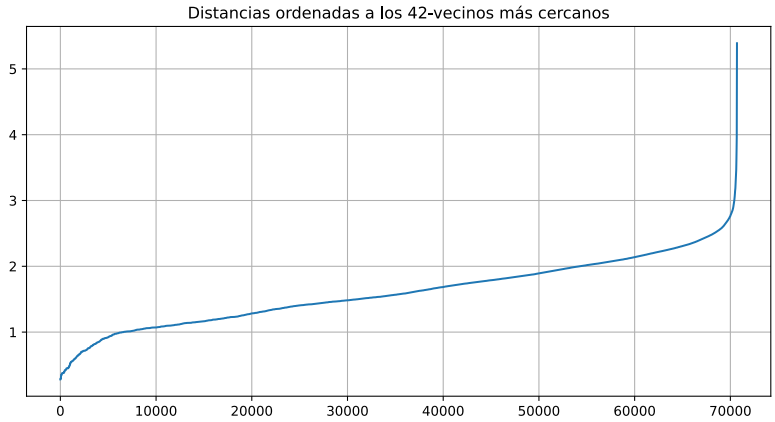



Figura 14: Distancias ordenadas a los 42-vecinos más cercanos

Para determinar el valor de *eps* debemos fijarnos en el eje Y de esta gráfica, y seleccionar el valor en el que se produce un cambio brusco en la curva. En este caso, podemos ver como este "codo" se produce aproximadamente entre $eps = 2$ y $eps = 3$, por lo que se utilizarán para *eps* valores entre 2 y 3 con saltos de 0.1.

### 3.3. Visualización de resultados con t-SNE

Como hemos visto previamente, el conjunto de datos de entrada es de 21 dimensiones, por lo que no podemos obtener gráficamente una representación de la clasificación realizada por los diferentes algoritmos de clasificación no supervisada.

Para solucionar este inconveniente, se utiliza lá técnica del t-SNE (t-Distributed Stochastic Neighbor Embedding).
El t-SNE es una técnica de reducción de dimensionalidad no lineal diseñada principalmente para visualización de datos de alta dimensión en 2D (en este caso) o 3D.
t-SNE convierte datos de muchas dimensiones (21 en este caso) en representaciones visuales en 2D, tratando de preservar la estructura local: puntos cercanos en el espacio original seguirán estando cerca en el espacio reducido.

De esta forma, podemos representar en un espacio de 2 dimensiones la clasificación real de los datos (figura 15), así como las clasificaciones realizadas por el algoritmo k-medias (figura 16) y el DBSCAN (17).

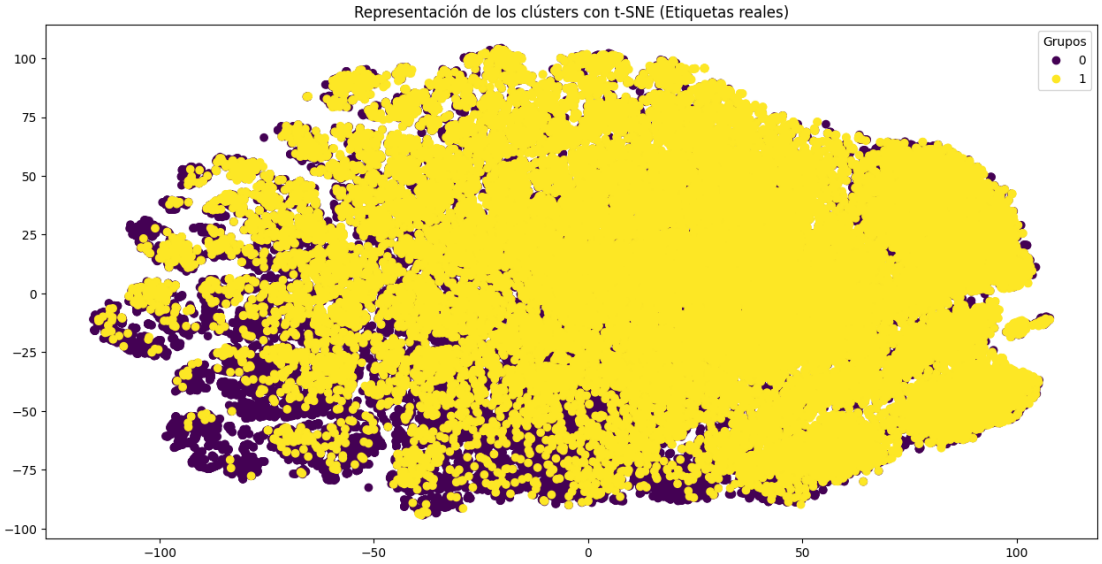



Figura 15: Clasificación real de los datos

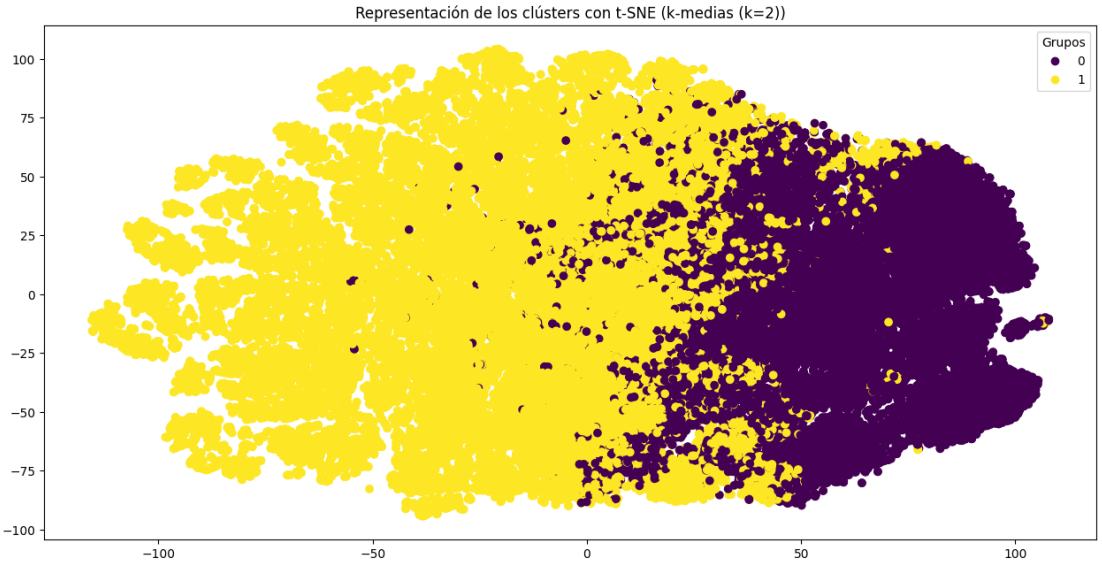



Figura 16: Clústers obtenidos por k-medias

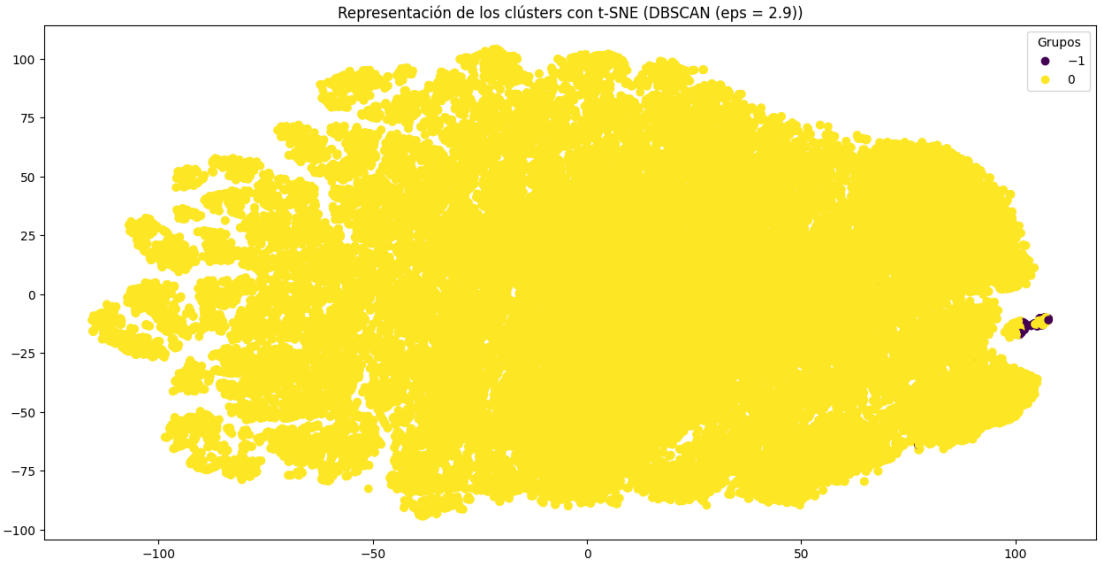



Figura 17: Clústers obtenidos por DBSCAN

En la figura 15 podemos ver que el conjunto de datos no es muy separable, ya que no existe separación clara entre los datos de ambas clases, que, a simple vista, parecen prácticamente superpuestas. Esto se puede deber en pequeña medida debido a la reducción de dimensión, pues al pasar de 21 variables a solo 2 se pierde parte de información, pero el t-SNE conserva de forma general la estructura local de los datos, por lo que podemos intuir que, este conjunto de datos concreto, no se presta especialmente bien para aplicar técnicas de clustering.

En las figuras 16 y 17 podemos ver que se confirman nuestras sospechas, pues ninguno de los modelos consigue una clasificación similar a la real. De hecho, en el caso representado para el DBSCAN (con $epsilon = 2,9$), ni siquiera obtiene 2 clases (que son las que realmente existen), y considera una única clase, caso que se repite para muchos de los valores de *epsilon* considerados, que dividen el conjunto de datos en 1, 3 o incluso 4 clusters.

# 4. Resultados

## 4.1. Resultados: reducción de dimensionalidad

En esta sección se analizarán los resultados obtenidos por las diferentes combinaciones de covariables en un modelo, la estructura será la siguiente.

1. Diagrama de cajas con el número de componentes de cada modelo. En esta imagen se representara el accuracy, pues al tratarse de un dataset muy equilibrado (como se puede ver en 1.2) se considera que la métrica más importante para ver la calidad del modelo es la exactitud.

2. Se realiza el test de Kruskal-Wallis(K-W) para ver si los modelos son estadísticamente iguales o por el contrario existe alguna diferencia debida al número de covariables. Si existe se realizará la prueba de Tukey HSD explicada en 1.5.1.

3. A continuación se explicará cual es el mejor modelo, el porqué y las gráficas mas representativas del mismo.

4. Por último se realizará una comparación de las medias de cada fold del acurracy, recall, precision y f1-score para cada modelo, destacando cual es el mejor en esos apartados.

### 4.1.1. Resultados Random Forest Classifier

Como se comentó anteriormente se empezará mostrando el diagrama de cajas de este modelo con un número diferente de covariables. Cave destacar que siempre se usó el mismo random seed independientemente del número de componentes.

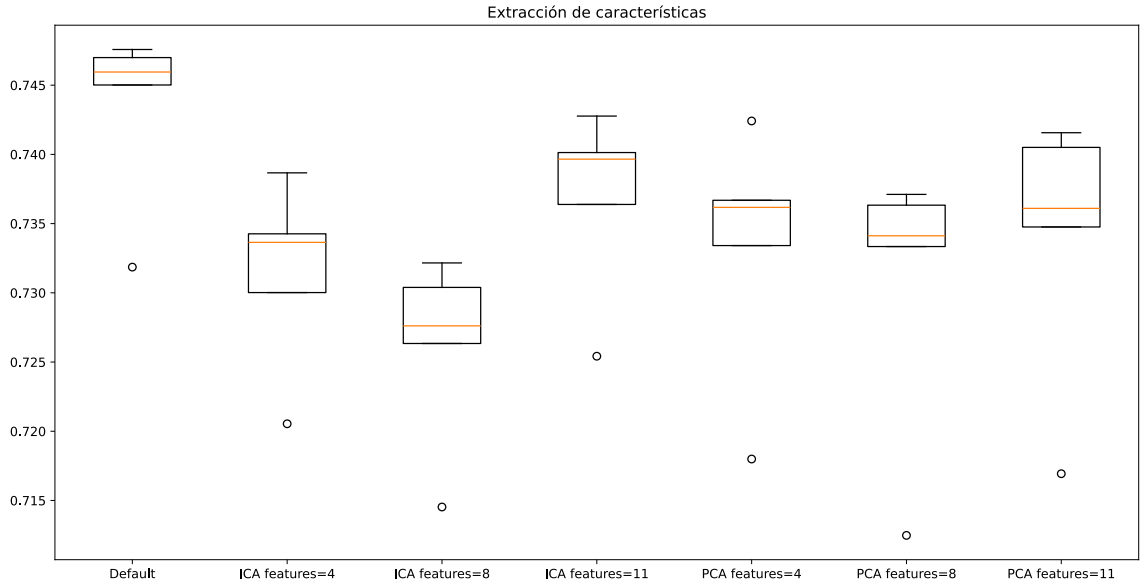



Figura 18: Diagrama de cajas RFC

A simple vista parece que el mejor modelo es el que trabaja con todas las componentes. Esto indica que las variables del conjunto de datos esta bien seleccionadas, pues no hay ninguna (o casi) que sea redundante o que aporte información errónea al modelo. También llama la atención que con todos los modelos existe un conjunto de datos con el que el modelo baja la exactitud notablemente.
En el siguiente paso se realiza la prueba de K-W, en la cual obtenemos lo siguiente:

$$p\text{-}valor\ KrusW = 0{,}045488737398838706 > 0{,}01 \rightarrow \textit{Aceptamos la hipótesis: los modelos son iguales}$$

Teniendo este resultado no es necesario realizar el test de Tukey, y se puede decir que el mejor modelo no se puede determinar por métodos estadísticos, por lo que es necesario usar otro tipo de criterio, en este caso se seleccionara el computacionalmente menos costoso. A priori se podría pensar

que se trata de los modelos que usan un menor número de covariables, pero esta suposición se deberá confirmar en la tabla 7. Al ser todos los modelos iguales, se seleccionó un modelo al azar para mostrar su matriz de confusión y su curva ROC .

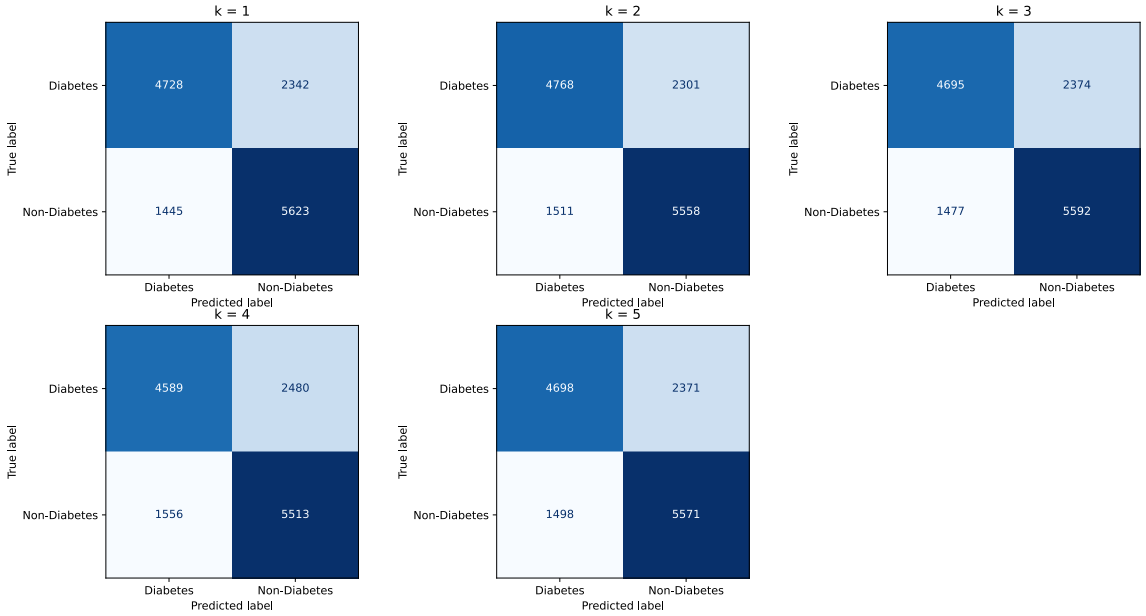



Figura 19: Matrices de confusión para cada fold del modelo de RFC con 8 componentes extraídas de ICA

Como se puede ver en la imagen 19, en este modelo se puede ver que donde más falla es en la predicción de diabetes, aproximadamente 700 fallos más que en la predicción de Non-Diabetes.

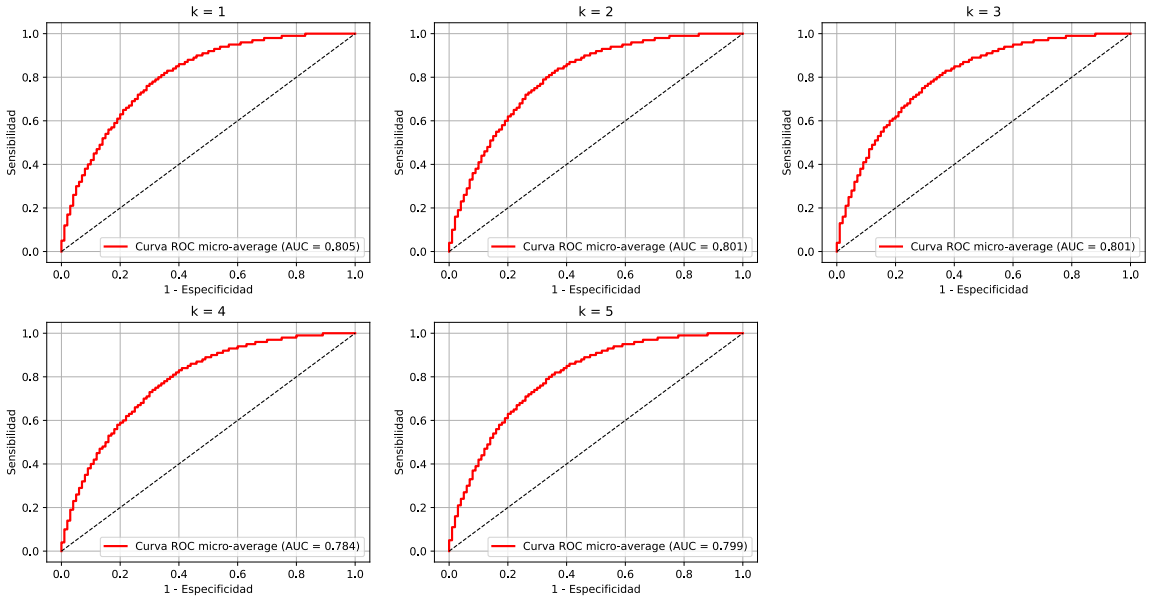



Figura 20: Curvas ROC globales para cada fold del modelo de RFC con 8 componentes extraídas de ICA

Como se puede ver en las gráficas de la imagen 20, el valor del área debajo de la curva es bastante pequeño, por lo que se podría concluir que ninguno de estos modelos podrían a tener una aplicación practica en la medicina, pues no se puede permitir una tasa de error tan alta. Para finalizar con esta sección se deja a continuación la tabla 3 donde se pueden ver los valores medios de las métricas del modelo de random forest con diferente número de componentes.

| Model | Accuracy Avg | Precision Avg | Recall Avg | F1-Score Avg |
|---|---|---|---|---|
| RFC+Covariables Defalut | 0,7435 | 0,7449 | 0,7435 | 0,7431 |
| RFC+4 covariables ICA | 0,7314 | 0,7349 | 0,7314 | 0,7304 |
| RFC+8 covariables ICA | 0,7262 | 0,7297 | 0,7262 | 0,7251 |
| RFC+11 covariables ICA | 0,7369 | 0,7396 | 0,7369 | 0,7361 |
| RFC+4 covariables PCA | 0,7333 | 0,7389 | 0,7333 | 0,7318 |
| RFC+8 covariables PCA | 0,7307 | 0,7385 | 0,7307 | 0,7284 |
| RFC+11 covariables PCA | 0,7340 | 0,7408 | 0,7340 | 0,7321 |

Cuadro 3: Tabla con las medias de las métricas más importantes del RFC con múltiples pruebas del número de covariables

### 4.1.2. Resultados K-Nearest Neighbors

Igual que en modelo anterior se comenzara por mostrar el diagrama de cajas del KNN en la imagen 21.

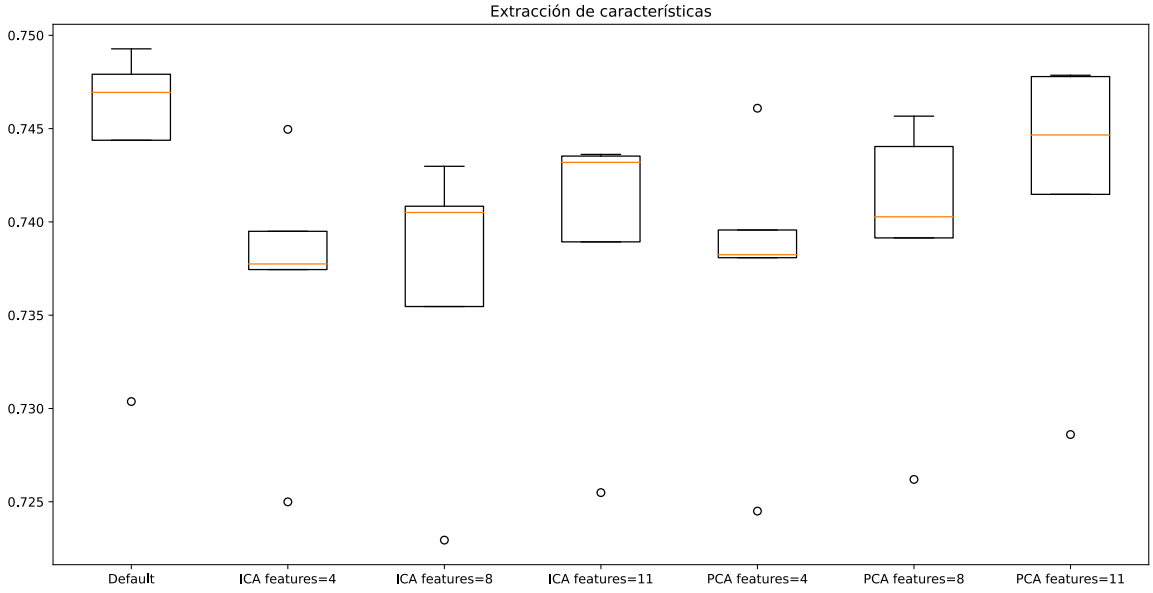



Figura 21: Diagrama de cajas KNN

Igual que en caso anterior se puede llegar a la conclusión de que el modelo con los parámetros por defecto tiene un valor de accuracy meyor que el resto, pero cuando se hace la prueba de K-W se obtiene lo siguiente:

$$p\text{-}valor\ KrusW = 0{,}24317020605030049 > 0{,}01 \rightarrow \textit{Aceptamos la hipótesis: los modelos son iguales}$$

En este caso el p-valor es mucho mayor, lo que indica que la posibilidad de que los modelos sean iguales es mucho mayor que en caso anterior. Al saber los resultados del test de K-W ya no se realizó la prueba de Tukey. A continuación se sigue el mismo proceso que en la sección 4.1.1, es decir se selecciona un modelo al azar y se dibuja sus matrices de confusión y sus curvas ROC.

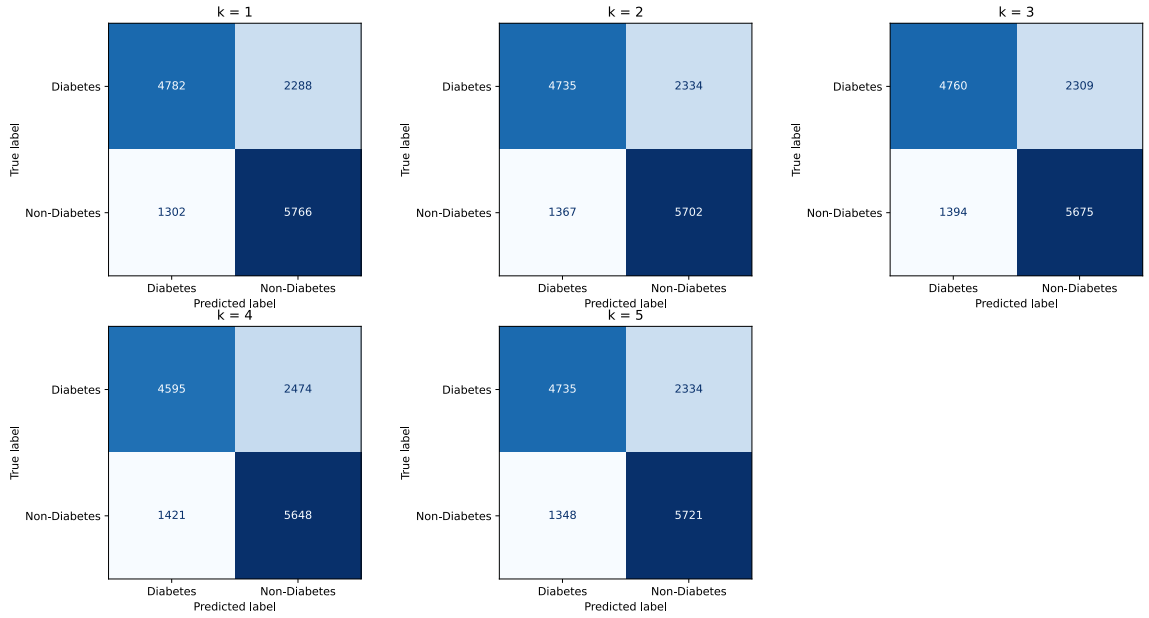



Figura 22: Matrices de confusión para cada fold del modelo de KNN con 4 componentes extraídas de PCA

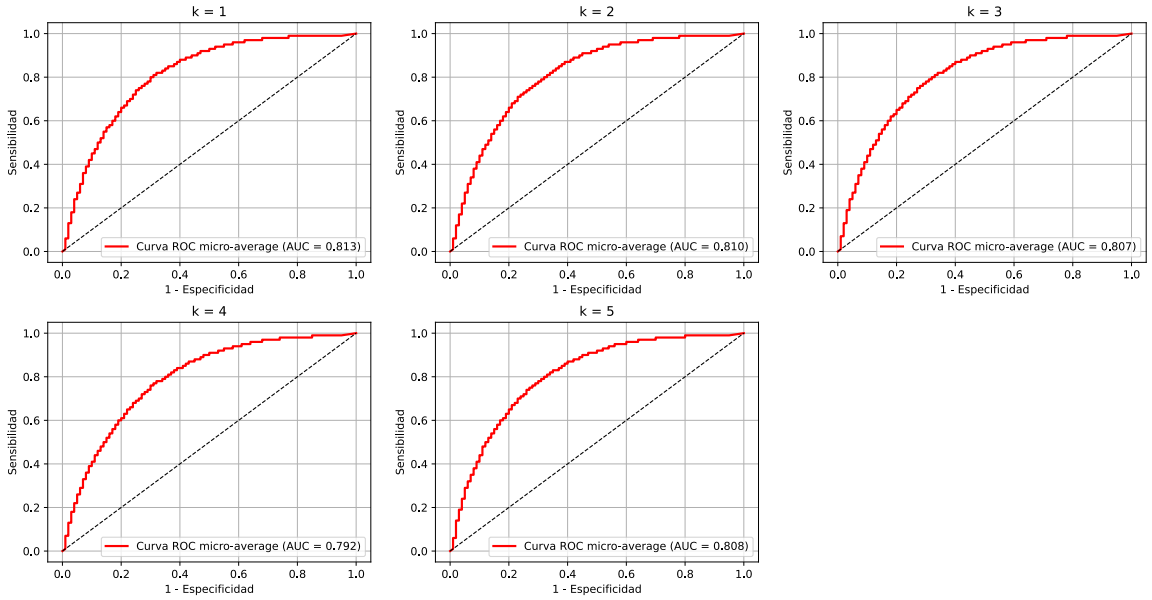



Figura 23: Curvas ROC globales para cada fold del modelo de KNN con 4 componentes extraídas de PCA

Los resultados de ambas imágenes (22 y 23) son muy parecidos a los obtenidos en la sección 4.1.1, es por eso que no se cree necesario realizar una amplia explicación sobre alguna de ellas. Para finalizar con esta sección, se recopilan los valores medios de las métricas extraídas con KNN en la tabla 6

| Model | Accuracy Avg | Precision Avg | Recall Avg | F1-Score Avg |
|---|---|---|---|---|
| KNN+Covariables Defalut | 0,7438 | 0,7480 | 0,7438 | 0,7427 |
| KNN+4 covariables ICA | 0,7369 | 0,7410 | 0,7369 | 0,7358 |
| KNN+8 covariables ICA | 0,7365 | 0,7403 | 0,7365 | 0,7355 |
| KNN+11 covariables ICA | 0,7390 | 0,7407 | 0,7390 | 0,7385 |
| KNN+4 covariables PCA | 0,7373 | 0,7420 | 0,7373 | 0,7360 |
| KNN+8 covariables PCA | 0,7391 | 0,7442 | 0,7391 | 0,7377 |
| KNN+11 covariables PCA | 0,7421 | 0,7468 | 0,7421 | 0,7408 |

Cuadro 4: Tabla con las medias de las métricas más importantes del KNN con múltiples pruebas del número de covariables

### 4.1.3. Resultados Artificial Neural Network

A continuación se muestra en la imagen 24 el diagrama de cajas de los distintos modelos de ANN

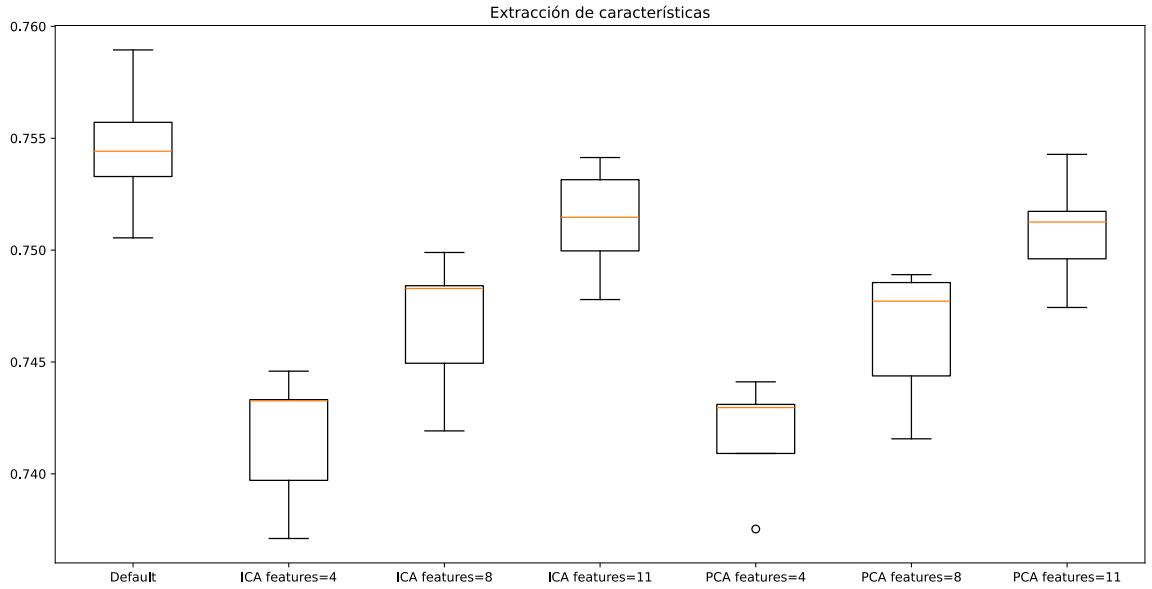



Figura 24: Diagrama de cajas ANN

En esta imagen si que se ve mucho más claramente como los valores de accuracy aumentan según el número de componentes que se introduzcan, por lo que podemos llagar a la conclusión de que en este caso lo más importante es el número de variables que se le introduzcan al modelo y no tanto el método de extracción de las mismas (PCA/ICA). Seguidamente se procede a realizar el test de K-W, cuyo resultado es:

$p\text{-}valor\ KrusW = 0{,}0002211919059142639 < 0{,}01 \rightarrow$ *Rechazamos la hipótesis: los modelos son diferentes*

Visto esto se procede a realizar el test de Turkey HSD, cuyos resultados se muestran en la siguiente tabla.

| group1 | group2 | meandiff | p-adj | lower | upper | reject |
|---|---|---|---|---|---|---|
| Default | ICA features=11 | -0.0033 | 0.5693 | -0.0091 | 0.0026 | False |
| Default | ICA features=4 | -0.0130 | 0.0000 | -0.0188 | -0.0071 | True |
| Default | ICA features=8 | -0.0079 | 0.0032 | -0.0137 | -0.0021 | True |
| Default | PCA features=11 | -0.0037 | 0.4241 | -0.0096 | 0.0021 | False |
| Default | PCA features=4 | -0.0129 | 0.0000 | -0.0187 | -0.0070 | True |
| Default | PCA features=8 | -0.0084 | 0.0017 | -0.0142 | -0.0025 | True |
| ICA features=11 | ICA features=4 | -0.0097 | 0.0002 | -0.0155 | -0.0039 | True |
| ICA features=11 | ICA features=8 | -0.0046 | 0.1954 | -0.0104 | 0.0012 | False |
| ICA features=11 | PCA features=11 | -0.0004 | 1.0000 | -0.0063 | 0.0054 | False |
| ICA features=11 | PCA features=4 | -0.0096 | 0.0003 | -0.0154 | -0.0037 | True |
| ICA features=11 | PCA features=8 | -0.0051 | 0.1203 | -0.0109 | 0.0008 | False |
| ICA features=4 | ICA features=8 | 0.0051 | 0.1185 | -0.0007 | 0.0109 | False |
| ICA features=4 | PCA features=11 | 0.0093 | 0.0005 | 0.0034 | 0.0151 | True |
| ICA features=4 | PCA features=4 | 0.0001 | 1.0000 | -0.0057 | 0.0060 | False |
| ICA features=4 | PCA features=8 | 0.0046 | 0.1927 | -0.0012 | 0.0105 | False |
| ICA features=8 | PCA features=11 | 0.0042 | 0.2940 | -0.0017 | 0.0100 | False |
| ICA features=8 | PCA features=4 | -0.0050 | 0.1359 | -0.0108 | 0.0009 | False |
| ICA features=8 | PCA features=8 | -0.0005 | 1.0000 | -0.0063 | 0.0054 | False |
| PCA features=11 | PCA features=4 | -0.0091 | 0.0005 | -0.0150 | -0.0033 | True |
| PCA features=11 | PCA features=8 | -0.0046 | 0.1900 | -0.0105 | 0.0012 | False |
| PCA features=4 | PCA features=8 | 0.0045 | 0.2182 | -0.0013 | 0.0103 | False |

Cuadro 5: Comparación de Tukey HSD entre el modelo de ANN con diferentes componentes

En esta tabla se puede ver que el mejor modelo esta entrenado con 11 componentes (sacadas de ICA o PCA) o las variables que venían en el dataset por defecto. Para saber cual es el mejor modelo se deberá saber los tiempos de entrenamiento de cada uno y el que tarde menos (sea computacionalmente menos costoso) debera ser el seleccionado finalmente. Como este tema se desarrollara en la sección 4.2, se selecciona uno de los tres modelos aleatorios y se muestran las matrices de confusión y las curvas ROC de cada fold.

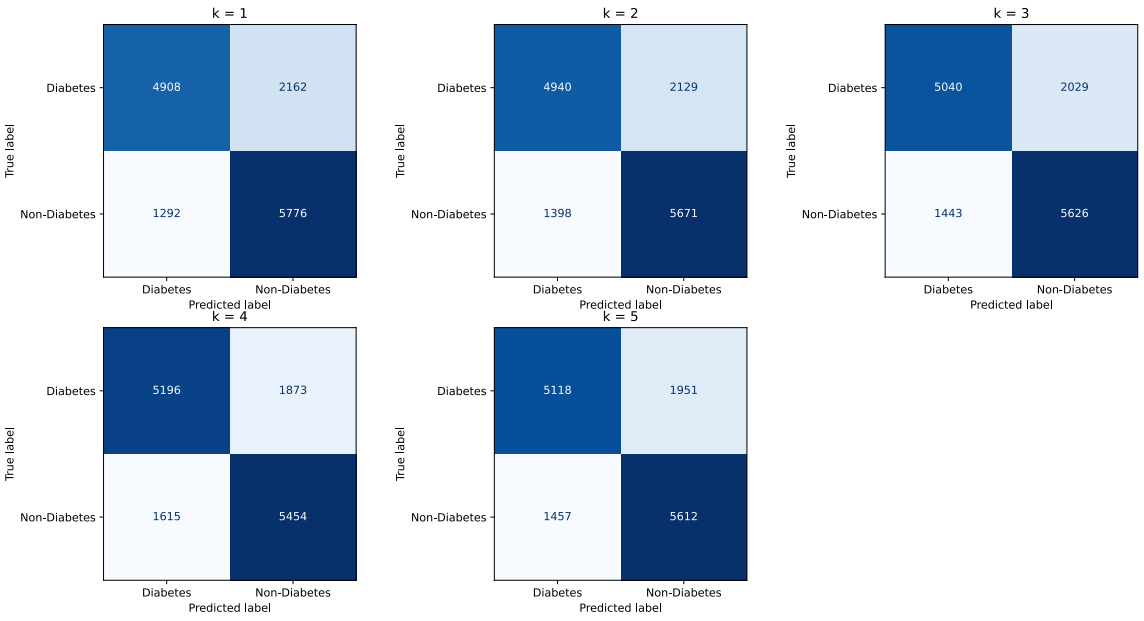



Figura 25: Matrices de confusión para cada fold del modelo de ANN con los componentes por defecto

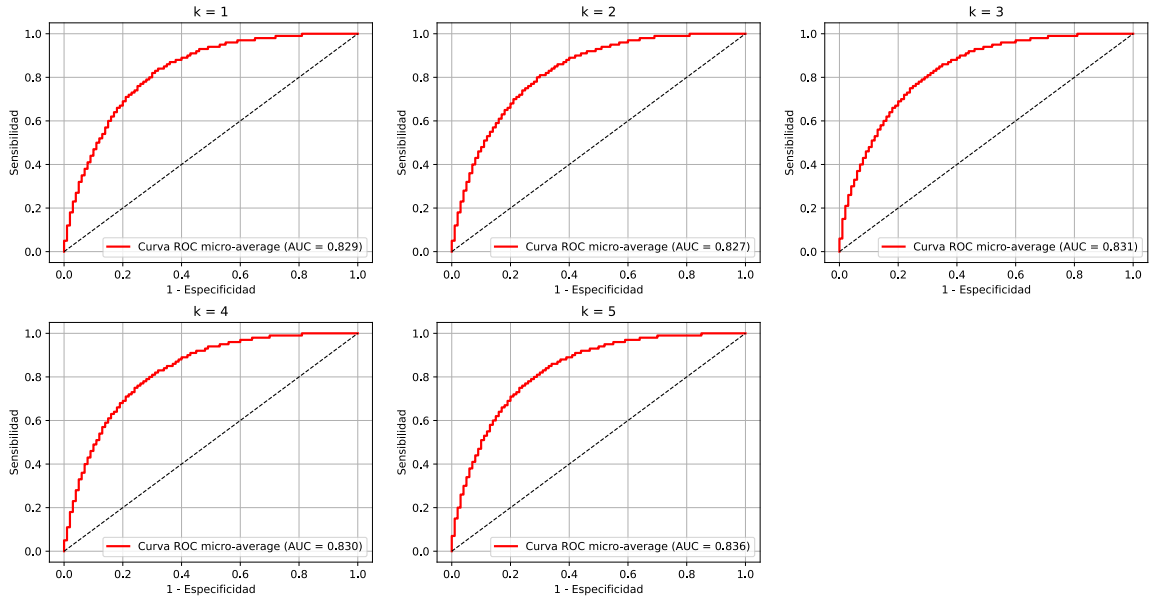



Figura 26: Curvas ROC globales para cada fold del modelo de ANN con los componentes por defecto

Por último, como en todas las subsecciones anteriores se dibuja una tabla con los valores medios de la exactitud, precisión, f1-score y sensibilidad.

| Model | Accuracy Avg | Precision Avg | Recall Avg | F1-Score Avg |
|---|---|---|---|---|
| ANN+Covariables Defalut | 0,7546 | 0,7566 | 0,7546 | 0,7541 |
| ANN+4 covariables ICA | 0,7416 | 0,7443 | 0,7416 | 0,7409 |
| ANN+8 covariables ICA | 0,7484 | 0,7489 | 0,7484 | 0,7483 |
| ANN+11 covariables ICA | 0,7513 | 0,7537 | 0,7513 | 0,7507 |
| ANN+4 covariables PCA | 0,7417 | 0,7445 | 0,7417 | 0,7410 |
| ANN+8 covariables PCA | 0,7462 | 0,7494 | 0,7462 | 0,7454 |
| ANN+11 covariables PCA | 0,7509 | 0,7541 | 0,7509 | 0,7501 |

Cuadro 6: Tabla con las medias de las métricas más importantes del ANN con múltiples pruebas del número de covariables

En el modelo de ANN+Covariables default están las 4 métricas más altas de los 21 modelos que se probaron en este trabajo, pero se tiene que valorar si la diferencia con otros modelos o con el mismo pero un número diferente de covariables es la suficiente para seleccionar este modelo como el más óptimo, o por otro lado si la diferencia de tipos es muy grande se selecciona otro modelo. Este sera uno de los puntos que se discutirán en la conclusión 5.

## 4.2. Resultados: medición de tiempos

En esta sección se muestran los tiempos del entrenamiento de cada modelo. Se dividen en tablas donde cada una será un modelo de IA diferente. Cada fila dentro de la tabla indica el número de componentes usado y que método de reducción de la dimensionalidad se usó, en cambio las columnas son el modo de ejecución del entrenamiento . El timeit solo se ejecuta una única vez y lo que se desea cronometrar es el tiempo de entrenamiento de los 5 folds, de esta manera no se cronometran el escalado, ni el PCA/ICA, para que de esta forma estos modelos no se vean claramente perjudicados en la comparación de tiempos.

Se empezará comentando os tiempos del modelo de RFC, que se puede ver en la siguiente tabla 7.

| Modelo | Secuencial (s) | Multihilo (s) | Multiproceso (s) | Tiempo n_jobs (s) |
|---|---|---|---|---|
| RFC Variables default | 78,47 | 42,48 | 37,64 | 43,51 |
| RFC ICA 4 componentes | 235,35 | 63,79 | 87,80 | 34,25 |
| RFC ICA 8 componentes | 267,11 | 72,39 | 88,69 | 67,63 |
| RFC ICA 11 componentes | 333,64 | 90,33 | 118,21 | 48,11 |
| RFC PCA 4 componentes | 230,94 | 60,84 | 88,24 | 33,51 |
| RFC PCA 8 componentes | 243,38 | 67,13 | 85,24 | 66,30 |
| RFC PCA 11 componentes | 330,29 | 89,78 | 117,94 | 48,33 |

Cuadro 7: Tiempos de ejecución del modelo RFC en diferentes modos de procesamiento

Se puede ve que tanto multihilo como multiproceso mejoran en tiempo al secuencial, método que supuestamente debería ser el más lento con amplia diferencia. También se puede ver que cuando aumenta el número de componentes tanto en ICA como en PCA el tiempo también aumenta de manera exponencial en cualquier caso.

A continuación se muestra la tabla de tiempos de los modelos de KNN 8.

| Modelo | Secuencial (s) | Multihilo (s) | Multiproceso (s) | Tiempo n_jobs (s) |
|---|---|---|---|---|
| KNN Variables default | 0,10 | 0,05 | 17,71 | 0,11 |
| KNN ICA 4 componentes | 0,25 | 0,25 | 17,73 | 0,27 |
| KNN ICA 8 componentes | 0,51 | 0,57 | 20,56 | 0,46 |
| KNN ICA 11 componentes | 0,63 | 0,57 | 17,75 | 0,61 |
| KNN PCA 4 componentes | 0,27 | 0,5 | 17,69 | 0,28 |
| KNN PCA 8 componentes | 0,48 | 0,45 | 20,57 | 0,48 |
| KNN PCA 11 componentes | 0,58 | 0,58 | 17,78 | 0,62 |

Cuadro 8: Tiempos de ejecución del modelo KNN en diferentes modos de procesamiento

A nivel de tiempos este es el modelo más “singular”, pues al realizar un entrenamiento tan rápido resulta muy difícil asegurar con un cierto nivel de fiabilidad cual de los 3 procesos -secuencial, multihilo y n_jobs- es el más rápido, pues todos tienen el mejor tiempo con alguna configuración. Para saber cual es el mejor se deberían hacer una gran cantidad de pruebas y sacar una media de tiempos para cada modelo.
Respecto a lo que se comento en la primera linea, este modelo se considera el más “extraño” por los resultados temporales del multiproceso. Aunque tiene algo de sentido que este modelo sea el más lento ya que al tratarse de un proceso tan rápido el tiempo que tarda el equipo en la creación y gestión de procesos puede ser diferencial, pero no hasta tal punto de existir una diferencia de 17 segundos. Esta parte tiene difícil explicación ya que aunque en un principio se pensó en un problema de sobrecarga de procesos, esta hipótesis se descartó pues en los otros modelos se uso el mismo código base y no se tienen tiempos tan dispares al resto. Es por esto que no se logró encontrar ninguna explicación/solución a este comportamiento.

Por último pasamos a la tabla de tiempos de los modelos ANN 9.

| Modelo | Secuencial (s) | Multihilo (s) | Multiproceso (s) |
|---|---|---|---|
| ANN Variables default | 55,85 | 32,53 | 43,97 |
| ANN ICA 4 componentes | 54,21 | 32,30 | 36,04 |
| ANN ICA 8 componentes | 39,63 | 16,85 | 35,19 |
| ANN ICA 11 componentes | 58,30 | 42,43 | 34,00 |
| ANN PCA 4 componentes | 37,53 | 22,50 | 34,36 |
| ANN PCA 8 componentes | 32,05 | 20,31 | 33,60 |
| ANN PCA 11 componentes | 54,75 | 34,00 | 40,61 |

Cuadro 9: Tiempos de ejecución del modelo ANN en diferentes modos de procesamiento

De esta tabla se pueden extraer las siguientes conclusiones

1. En todos los casos el modo multihilo es el proceso más eficiente, pues tiene los mejores tiempos con una amplia diferencia (varios segundos).

2. El multiproceso no se establece como el 2º mejor método, pues por sorprendentemente que parezca en algunos casos es peor que el modo secuencial (PCA 8 componentes).

3. al contrario que en la tabla 7, los tiempos no crecen según el número de componentes, pues el entrenamiento más rápido, independientemente del modo de ejecución, se produce con 8 componentes en ICA y PCA.

## 4.3. Resultados: clustering

En esta sección se analizan los resultados obtenidos por los algoritmos de clustering en función de las métricas que se han considerado: V-Measure e índice de Rand ajustado.

### 4.3.1. Resultados: k-means

| Fecha y hora | Modelo | V-Measure | Índice de Rand ajustado |
|---|---|---|---|
| 30/04/2025 19:01 | kmeans | 0,0799679 | 0,08982246 |

Los valores obtenidos, tanto de V-Measure como de ARI son muy bajos, lo que sugiere que el agrupamiento con k-means no logra identificar bien las verdaderas clases del dataset, aún partiendo del número de clústers realmente existentes en el conjunto de datos.

### 4.3.2. Resultados: DBSCAN

| Fecha y hora | Modelo | Epsilon | V-Measure | Índice de Rand ajustado |
|---|---|---|---|---|
| 30/04/2025 19:08 | dbscan | 2 | 0,004617421 | 0,001004287 |
| 30/04/2025 19:09 | dbscan | 2,1 | 0,002308316 | 0,000252743 |
| 30/04/2025 19:09 | dbscan | 2,2 | 0,000944013 | 5,10454E-05 |
| 30/04/2025 19:10 | dbscan | 2,3 | 0,00032216 | 8,78609E-06 |
| 30/04/2025 19:10 | dbscan | 2,4 | 0,000167524 | 2,37901E-06 |
| 30/04/2025 19:11 | dbscan | 2,5 | 0,00010858 | 9,45285E-07 |
| 30/04/2025 19:12 | dbscan | 2,6 | 5,03365E-05 | 2,34139E-07 |
| 30/04/2025 19:13 | dbscan | 2,7 | 8,12742E-05 | 6,02217E-08 |
| 30/04/2025 19:15 | dbscan | 2,8 | 9,08724E-06 | -3,5147E-08 |
| 30/04/2025 19:17 | dbscan | 2,9 | 1,4997E-06 | -3,99887E-08 |

La mayoría de las ejecuciones muestran medidas de V-Measure e índice de Rand ajustado prácticamente nulos, lo que indica una muy mala calidad de agrupamiento. Esto lleva a pensar que el método del DBSCAN no es adecuado para clasificar este conjunto de datos.
El mejor resultado se alcanza con $epsilon = 2,7$, con un V-Measure de aproximadamente 0,8127 y un índice de Rand ajustado de 0,0622. Este valor de índice de Rand ajustado sigue siendo bajo, lo que implica que aunque hay cierta homogeneidad en los clusters, no se alinean bien con las etiquetas reales.

# 5. Conclusiones

El desarrollo de este proyecto permitió aplicar de manera práctica técnicas de reducción de dimensionalidad aplicadas a una clasificación supervisada, así como técnicas de clustering para realizar una clasificación no supervisada.

Como se puede ver en los resultados de reducción de dimensionalidad 4.1, las métricas son muy parecidas entre todos los modelos independientemente del número de componentes que se utiliza el entrenamiento, aunque se esperaba que según aumentase el número de componentes la exactitud debería aumentar, cosa que no pasa en la realidad. También se puede comprobar que a nivel de resultados es indiferente que técnica de reducción de dimensionalidad se utilice (PCA o ICA), como se puede ver claramente en la imagen 24.
Respecto a los tiempos de ejecución qu tienen estos modelos se puede destacar el buen rendimiento que tiene el modo multihilo y del parámetro n_jobs pues en la mayoría de casos han sido los modos más eficientes.A pesar de eso, no existe un modo que sea infinitamente mejor que los otros en este caso, por lo que lo más correcto es seleccionar un modo de ejecución concreto para cada modelo en lugar de seleccionar una única estrategia para todos.

1. Modelo RFC: Mejor modo de ejecución es el **n_jobs**.
2. Modelo ANN: Mejor modo de ejecución es el **multihilo**.
3. Modelo KNN: Mejor modo de ejecución es el **secuencial**. En este caso se elige el secuencial porque a simple vista no se puede elegir uno mejor que el resto, por lo que selecciona el que menos complejidad requiere a nivel de código.

El lado negativo de esta sección lo deja el modo de multiproceso, pues en ninguno de los tres modelos se obtuvieron los tiempos esperados. Este modo de ejecución suele ser más eficiente de todos, que en este caso no lo sea deja entrever un posible problema en el correcto funcionamiento del codigo o del equipo.

En cuanto al uso de técnicas de clustering, los resultados no han sido para nada satisfactorios, como hemos visto en la sección 3 y en el análisis de los resultados obtenidos. Estos malos resultados se han dado para los dos algoritmos diferentes que se han probado (uno de ellos probado hasta 10 veces modificando uno de los hiperparámetros). Este hecho, sumado a que la representación obtenida con el t-SNE es poco clara y no permite observar dos clases claramente diferenciadas, nos lleva a pensar que el conjunto de datos no se presta bien para trabajar con técnicas de clustering, debido a que las dos clases que se pretenden identificar se encuentran prácticamente superpuestas. Para corroborar esta afirmación se pueden aplicar otras técnicas de clustering como, por ejemplo, el clustering jerárquico, si bien, como se ha expuesto, se intuye que los resultados no van a ser satisfactorios sea cual sea el algoritmo de clustering que se utilice.

A nivel general, podemos decir que ninguno de los modelos estudiados (tanto de clasificación supervisada como supervisada) tiene mucha aplicabilidad al ámbito de aplicación (detección de diabetes), pues incluso en el mejor de los casos la exactitud no supera un 76 %, que no es una predicción fiable en un ámbito como el médico.

## Bibliografía


[1] ISO Vim. «International vocabulary of basic and general terms in metrology (VIM)». En: *International Organization* 2004 (2004), págs. 09-14.

[2] *Métricas para la evaluación del error (AA1 Tema 3)*.

[3] Naomi Altman y Martin Krzywinski. «Ensemble methods: bagging and random forests». En: *Nature Methods* 14.10 (2017), págs. 933-935.

[4] *Sckit Learn Random Forest Classifier*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`.

[5] *Modelos no lineales de aprendizaje supervisado (AA1 Tema 5)*.

[6] *Sckit Learn KNeighborsClassifier*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`.

[7] *Técnicas de reducción de la dimensión (AA2 Tema 1)*.

[8] Jörg Sander et al. «Density-based clustering in spatial databases: The algorithm gdbscan and its applications». En: *Data mining and knowledge discovery* 2 (1998), págs. 169-194.