Escola Universitaria Politécnica

UNIVERSIDADE DA CORUÑA

Grado en Ingeniería Electrónica Industrial y Automática

TRABAJO DE FIN DE GRADO

TFG Nº: **770G01A182**

TÍTULO: **Análisis e implementación de algoritmos de técnicas de clasificación one-class**

AUTOR: **IAGO NÚÑEZ LAMAS**

TUTOR: **ESTEBAN JOVE PÉREZ**
JOSÉ LUÍS CALVO ROLLE

FECHA: **SEPTIEMBRE DE 2020**

Fdo.: EL AUTOR

Fdo.: EL TUTOR

TÍTULO: **Análisis e implementación de algoritmos de técnicas de clasificación one-class**

ÍNDICE

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2020

AUTOR: IAGO NÚÑEZ LAMAS

Fdo.: IAGO NÚÑEZ LAMAS

Listado de figuras

Listado de tablas

TÍTULO: **Análisis e implementación de algoritmos de técnicas de clasificación one-class**

MEMORIA

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2020

AUTOR: IAGO NÚÑEZ LAMAS

Fdo.: IAGO NÚÑEZ LAMAS

Índice del documento MEMORIA

1 OBJETO

El principal objeto de este trabajo es el de realizar un análisis pormenorizado de las técnicas de clasificación one-class más empleadas en la actualidad, así como la adaptación de sus algoritmos a Matlab, para su uso en posteriores de aplicaciones. Se plantea la realización de un documento explicativo de cada una de ellas, en el que se oriente a potenciales usuarios sobre cómo emplear los algoritmos desarrollados, así como la validación de los algoritmos implementados sobre conjuntos de datos del repositorio UCI.

2 ALCANCE

En esta Sección se expone el alcance del presente Trabajo de Fin de grado, que abarca los siguientes puntos:

- Estudio general de los distintos tipos de clasificadores one-class.
- Selección de las técnicas más empleadas.
- Adaptación de dichas técnicas al entorno de programación Matlab.
- Elaboración de un documento que refleje el principio de funcionamiento y los parámetros que influyen en cada una de las técnicas. Este documento debe servir de guía a futuros usuarios para el uso de los algoritmos desarrollados.
- Validación del comportamiento de los algoritmos para detección de anomalías sobre datos reales.
- A pesar de no formar parte del alcance original del proyecto, se ha publicado una contribución científica en el congreso internacional CISIS 2020.

3 ANTECEDENTES

En esta Sección se describen los conceptos previos que sirven como punto de partida para el desarrollo posterior del trabajo.

3.1. Clasificación one-class

En la disciplina del *Machine Learning* existen diferentes tipos de problemas que se pueden abordar. Uno de ellos es la clasificación de objetos según su pertenencia a una clase. Un ejemplo de esto podría ser la clasificación de distintos tipos de frutas, por ejemplo fresas, frambuesas y moras. En ese caso, los objetos serían cada una de las piezas de fruta, mientras que las clases son el tipo de fruta, como puede verse en las Figuras 3.1 y 3.2. Dentro de los problemas de clasificación se puede distinguir entre tres categorías:

- Clasificación binaria: se distingue entre dos clases.
- Clasificación multiclase: se distingue entre tres o más clases.
- Clasificación one-class: únicamente se dispone información de una clase.

Figura 3.1 – Clasificación multiclase.

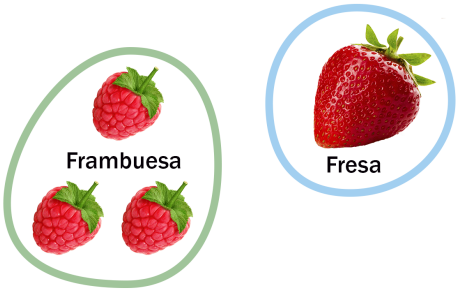

Figura 3.2 – Clasificación binaria.

Este trabajo se centrará en la clasificación one-class, en la que se realiza una clasificación de los objetos según su pertenencia o no a una clase: la clase *target* u objetivo. De esta forma, dado un objeto, este podrá pertenecer a la clase objetivo o no. En caso de que no pertenecer a ella, se denomina como *outlier*, como muestra la Figura 3.3.

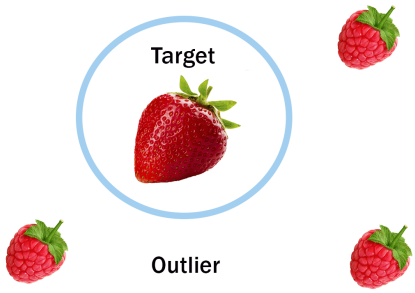

Figura 3.3 – Clasificación one-class.

La particularidad de este tipo de clasificación con respecto a la clasificación binaria o la clasificación multiclase, es que únicamente se dispone información de la clase objetivo para

extraer reglas o patrones que la caractericen, lo que de ahora en adelante se denominará como entrenamiento de un clasificador. Esto provoca que, en algunas ocasiones, si los datos objetivo y los outlier se solapan, resulte complicado encontrar un clasificador que presente un rendimiento aceptable. Por ejemplo, si solo se tiene en cuenta el parámetro del color, podría ser complicado distinguir entre fresas y frambuesas, o de la misma forma, si solo se tiene en cuenta el tamaño, ocurriría lo mismo entre moras y frambuesas.

3.2. PRTools

PRTools es una *toolbox* de Matlab dedicada al reconocimiento de patrones y desarrollada, principalmente, por Robert P.W. Duin y David Tax. En lo que ocupa al presente trabajo, se utilizará una parte de esta toolbox, denominada *Data Description Toolbox*, como base para la implementación en Matlab de los siguientes algoritmos:

- KNN.
- K-Centers.
- SVDD.
- MST.
- Gauss.
- Parzen.
- PCA.
- K-Means.
- Autoencoders.

Los resultados presentados por la *toolbox* serán:

- El modelo del clasificador: es una clase propia de la toolbox.
- El valor del *TPR*.
- El valor del *FPR*.

4 NORMAS Y REFERENCIAS

4.1. Disposiciones legales y normas aplicadas

En este trabajo se aplica el Reglamento del trabajo de fin de grado de la Escuela Universitaria Politécnica de la Universidad de A Coruña.

4.2. Referencias

[1] "Classify observations using support vector machine (svm) classifier." [Online]. Available: https://es.mathworks.com/help/stats/classreg.learning.classif.compactclassificationsvm.predict.html?lang=en

[2] "Train support vector machine (svm) classifier for one-class and binary classification." [Online]. Available: https://es.mathworks.com/help/stats/fitcsvm.html

[3] "Uci machine learning repository: data sets." [Online]. Available: https://archive.ics.uci.edu/ml/datasets.php

[4] J. Aldrich, "R.a. fisher and the making of maximum likelihood 1912-1922," *Stat Sci*, vol. 12, 09 1997.

[5] G. Cabral and A. Oliveira, "One-class classification for heart disease diagnosis," vol. 2014, 10 2014, pp. 2551–2556.

[6] P. Casale, O. Pujol, and P. Radeva, "Approximate convex hulls family for one-class classification," in *Multiple Classifier Systems*. Springer Berlin Heidelberg, 2011, pp. 106–115.

[7] ——, "Approximate polytope ensemble for one-class classification," *Pattern Recognition*, vol. 47, no. 2, pp. 854–864, feb 2014.

[8] T. Fawcett, "Introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, pp. 861–874, 06 2006.

[9] D. Fernandez-Francos, O. Fontenla-Romero, and A. Alonso-Betanzos, "One-class convex hull-based algorithm for classification in distributed environments," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–11, 2017.

[10] R. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, pp. 179–188, 01 1936.

[11] S. i and F. Herrera, "An extension on "statistical comparisons of classifiers over multiple data sets"for all pairwise comparisons," *Journal of Machine Learning Research - JMLR*, vol. 9, 12 2008.

[12] I. Jayarathne and M. Cohen, “Autoencoder-based one-class classification,” 11 2019.

[13] J.B.Kruskal, “On the shortest spanning subtree of a graph and the traveling salesman problem,” *Proceedings of the American Mathematical Society*, vol. 7, pp. 48–50, 02 1956.

[14] E. Jove, J.-L. Casteleiro-Roca, R. Casado-Vara, H. Quintián, J. A. Mendez, M. Mohamad, and J. Calvo-Rolle, “Comparative study of one-class based anomaly detection techniques for a bicomponent mixing machine monitoring,” *Cybernetics and Systems*, pp. 1–19, 07 2020.

[15] P. Juszczak, D. Tax, E. Pe¸kalska, and R. Duin, “Minimum spanning tree based one-class classifier,” *Neurocomputing*, vol. 72, pp. 1859–1869, 03 2009.

[16] D. Munroe and M. Madden, “Multi-class and single-class classification approaches to vehicle model recognition from images,” 06 2020.

[17] X. Pantazi, D. Moshou, and A. A. Tamouridou, “Automated leaf disease detection in different crop species through image features analysis and one class classifiers,” *Computers and Electronics in Agriculture*, vol. 156, pp. 96–104, 01 2019.

[18] E. Parzen, “On estimation of a probability density function and mode,” *Ann. Math. Statist.*, vol. 33, no. 3, pp. 1065–1076, 09 1962.

[19] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson, “Estimating support of a high-dimensional distribution,” *Neural Computation*, vol. 13, pp. 1443–1471, 07 2001.

[20] D. Tax, *Data Description Toolbox - dd tools 2.0.0*, Faculty EWI, Delft University of Technology. [Online]. Available: http://www.prtools.org

[21] ——, “One-class classification; concept-learning in the absence of counter-examples,” Ph.D. dissertation, Technische Universiteit Delft, 01 2001.

[22] D. Tax and R. Duin, “Support vector data description,” *Machine Learning*, vol. 54, pp. 45–66, 01 2004.

[23] B. Wan, Q. Guo, F. Fang, Y. Su, and R. Wang, “Mapping us urban extents from modis data using one-class classification method,” *Remote Sensing*, vol. 7, no. 8, p. 10143–10163, Aug 2015. [Online]. Available: http://dx.doi.org/10.3390/rs70810143

[24] S. Wold, K. Esbensen, and P. Geladi, “Principal component analysis,” *Chemometrics and Intelligent Laboratory Systems*, vol. 2, pp. 37–52, 08 1987.

4.3. Software utilizado

Los programas utilizados, por orden de uso de más a menos importante, han sido:

- Matlab: entorno utilizado para la implementación de los algoritmos y la obtención de figuras.

- Overleaf: utilizado en la redacción de la memoria.
- Visual Studio Code: para visualizaciones y representación de datos utilizando el lenguaje Python.
- Adobe Illustrator: para la creación de gráficos y esquemas.

5 DEFINICIONES Y ABREVIATURAS

En esta Sección se describirán las principales abreviaturas que se verán en a lo largo del trabajo, así como las definiciones de elementos que se consideran clave para la comprensión del mismo.

- APE: *Approximate Polytope Ensemble*
- AUC: *Area Under Curve* o área bajo la curva. Se utiliza en el análisis del rendimiento de los clasificadores.
- CH: *Convex Hull* o contorno convexo. Espacio convexo que contiene un número determinado de datos.
- ECP: *Extended Convex Polytope*. Es una versión expandida del contorno convexo y proyectada sobre un espacio de baja dimensionalidad.
- FPR: *False Positive Rate* o ratio de falsos positivos.
- I: Función indicatriz. Dado un subconjunto $A \subset X$, la función indicatriz de dicho subconjunto es una función definida en X que indica la pertenencia o no de los elementos de X al subconjunto A, de forma que para los elementos que pertenezcan la función asigna el valor 1, mientras que para los que no pertenezcan asigna el valor 0.
- KNN: *K-Nearest Neighbor* o vecino más próximo.
- MST: *Minimum Spanning Tree* o árbol de mínimo recubrimiento.
- OCSVM: *One-Class Support Vector Machine*
- $p(x)$: Función de densidad de una variable aleatoria continua. Describe la probabilidad relativa de que una variable aleatoria tome un determinado valor.
- PCA: *Principal Component Analysis*.
- ROC: *Operating Characteristic*. Representación gráfica de la sensibilidad frente a la especificidad de un clasificador.

- SCH: *Scaled Convex Hull*. Versión del CH proyectada sobre un espacio de baja dimensionalidad y expandida (o contraída).
- STD: Desviación típica.
- SVDD: *Support Vector Data Description*.
- SVM: *Support Vector Machine*.
- TPR: *True Positive Rate* o ratio de verdaderos positivos.
- UCI: University of California Irvine.
- x: Elemento del conjunto de datos de entrenamiento.
- z: Elemento del conjunto de datos de test.

6 REQUISITOS DE DISEÑO

Este trabajo tiene como objetivos el análisis y explicación de los métodos de clasificación one-class más relevantes en la actualidad. Además, se llevará a cabo una implementación de una serie de *scripts* que adapten dichas técnicas al entorno Matlab, de forma que el usuario pueda hacer uso de las mismas de una manera intuitiva. Para la implementación de los algoritmos se han fijado los siguientes requisitos:

- Para cada método debe realizarse un entrenamiento de forma que se obtenga un modelo óptimo del clasificador.
- El usuario debe tener la posibilidad de introducir el conjunto de datos deseado en formato '.mat', así como los hiperparámetros correspondientes a cada método.
- Una vez obtenido el modelo del mejor clasificador de la etapa de entrenamiento, se deberán validar los resultados con datos que no se hayan utilizado durante dicha etapa.
- Los resultados finales del entrenamiento y de la validación deben guardarse en formato *.mat* y serán presentados en la consola.

7 ANÁLISIS DE LAS SOLUCIONES

Este trabajo plantea el estudio y la adaptación de los algoritmos de las diferentes técnicas de clasificación one-class existentes. Teniendo esto en cuenta, se trata de un trabajo principalmente teórico, donde no existen diferentes alternativas que puedan ser contrapuestas para llevar a cabo una solución final. De esta forma se ha considerado que esta Sección no aplica al presente trabajo, y que el desarrollo del mismo debe encuadrarse en la Sección de Resultados Finales.

8 RESULTADOS FINALES

En esta sección se presenta el cuerpo principal del trabajo: el análisis de las técnicas y su implementación en el entorno Matlab.

8.1. Estructura de los resultados

La estructura del trabajo es la siguiente: en la Sección 8.2 se analiza y detalla el funcionamiento de cada uno de los métodos empleados para conseguir los objetivos; en la Sección 8.3 se describen las técnicas de evaluación del rendimiento de los clasificadores; en la Sección 8.4 se describe la adaptación de los algoritmos al entorno Matlab; en la Sección 8.5 se comprueba el funcionamiento de los algoritmos implementados mediante la validación con distintos conjuntos de datos, y por último, en las Secciones 8.6 y 8.7 se hará un análisis de las conclusiones y posibles vías de trabajo futuras. A mayores, en la Sección 8.8, se explica brevemente el trabajo de investigación para el congreso internacional CISIS 2020, llevado a cabo durante la realización de este TFG.

8.2. Análisis de técnicas one-class

Dentro de los métodos de clasificación one-class pueden diferenciarse tres categorías:

1. Métodos de modelado de contorno: basados en el cálculo de un contorno o límite que contiene a los datos de la clase objetivo.

2. Métodos de estimación de densidad: en los que se supone a priori que los datos siguen una distribución concreta (normal, Poisson, etc), y se calcula su función de densidad. Cuando se reciben nuevos datos a clasificar, estos pasan a través de la función de den-

sidad, y si superan un valor de umbral establecido, se consideran pertenecientes a la clase objetivo.

3. Métodos de reconstrucción: en los cuales se extraen las propiedades más esenciales de los datos objetivo disponibles, para después inferir un modelo que caracterice a todo el conjunto. Después se utiliza este modelo para describir los nuevos datos.

En las secciones 8.2.1, 8.2.2 y 8.2.3 se analizarán varios de los métodos existentes y más relevantes dentro de cada categoría.

8.2.1. Métodos de modelado de contorno

Las técnicas de modelado de contorno proponen resolver el problema de clasificación mediante la obtención, únicamente, de los límites de la estructura geométrica del conjunto de datos. De esta forma, no es necesario una cantidad elevada de datos para llevar a cabo estos métodos, ya que, a diferencia de la densidad, la estructura geométrica no depende tanto del número de muestras disponible. Sin embargo, el escalado de los datos (en el sentido euclidiano) sí resulta problemático, debido a que la estructura del conjunto depende de las distancias entre objetos [21].

A pesar de que estos métodos no minimizan de forma directa el volumen de dicha estructura, tienden a una solución aproximada del mínimo volumen, que depende directamente de lo bueno que sea el ajuste del método al conjunto de datos dado. Una vez obtenidos los contornos del conjunto de datos de entrenamiento que, como se ha mencionado anteriormente, representan a una sola clase, los nuevos objetos se clasifican en función de un umbral que se obtiene durante el entrenamiento del clasificador, y que, en la mayoría de casos, depende de forma directa de la distancia de estos al contorno calculado.

8.2.1.1. Approximate Polytope Ensemble (APE)

Dentro de las técnicas de modelado de contorno, una posibilidad para la obtención de los límites de un conjunto de datos objetivo, es el cálculo del contorno convexo o *convex hull* [6]. Dado un conjunto finito de datos $X \subset \mathbb{R}^n$, el *convex hull* $CH(X)$ se define como el mínimo conjunto convexo que contiene a X, cumpliendo la Ecuación 8.1 [7, p. 855]:

$$CH(X) = \left\{ \sum_{i=1}^{N} \theta_i x_i \mid \theta_i \geq 0, x_i \in X; \sum_{i=1} \theta_i = 1 \right\} \tag{8.1}$$

donde:

- θ_i es cada coeficiente de x_i
- N es el número de elementos de X

En las Figuras 8.1 y 8.2 pueden verse dos ejemplos del *convex hull* en dos y tres dimensiones respectivamente.

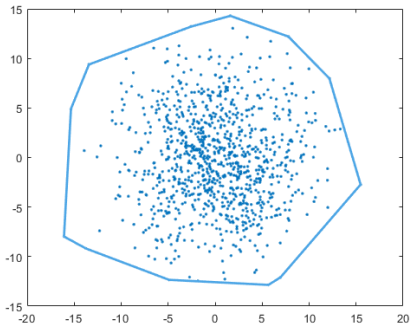

Figura 8.1 – *convex hull* en 2D.

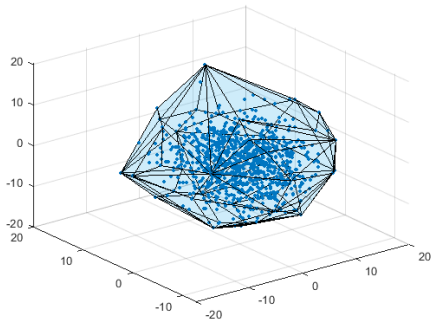

Figura 8.2 – *convex hull* en 3D.

Los principales problemas de este método son:

- El over-fitting causado por un sobreajuste del *CH* al conjunto de datos de entrenamiento, que pueda dejar fuera del límite a datos nuevos objetivo que no se hayan usado para el entrenamiento. Para que los límites del *CH* se puedan ajustar más a las necesidades del problema, y así evitar el over-fitting, se propone la utilización de versiones extendidas o reducidas del *CH* original, llamadas *extended convex polytopes* (*ECP*).
- El alto coste computacional que presentan los cálculos con conjuntos de datos de alta dimensionalidad, y que hace inviable el cálculo del *CH*. Este problema se puede resolver realizando un conjunto τ de proyecciones aleatorias del *ECP* calculado anteriormente sobre espacios de dos o tres dimensiones, de manera que es suficiente con que un nuevo objeto se sitúe fuera de una de estas proyecciones para ser considerado como *outlier*. En la Figura 8.3 se muestra un ejemplo para el conjunto de datos Iris del repositorio UCI. Dado que el conjunto de datos cuenta con 4 dimensiones, se representan 3 de ellas mediante los ejes, mientras que la cuarta se representa en la variación del color de cada punto (barra a la derecha del gráfico). Se representan en colores los puntos correspondientes a la clase Setosa, mientras que en rojo se muestra un dato de la clase Versicolor. En este caso, se realizan dos proyecciones, a partir de le las cuales se calculan los CH. Dado que el punto rojo se encuentra fuera de al menos una de las proyecciones, en este caso la 1, se descarta su pertenencia a la clase objetivo.

El método resultante de aplicar estas dos modificaciones se conoce como *Approximate convex Polytope decision Ensemble* (APE). Los vértices del *ECP* se calcularán en base al centro c, calculado como la media de todos los puntos. De esta forma, dados $c = \frac{1}{N}\sum x_i,\ \forall\ x_i \in X$, y el parámetro de expansión α, los vértices del *ECP* se definen como:

$$v_\alpha : \left\{ v + \alpha \frac{(v-c)}{\parallel v - c \parallel} \mid v \in CH(X) \right\} \tag{8.2}$$

Teniendo en cuenta que la matriz de proyecciones es generada aleatoriamente, la norma del espacio original no se conserva, por lo que un valor constante α del espacio original corresponderá a un conjunto de valores γ_i en el espacio proyectado. De esta forma, el conjunto de vértices que definen el *ECP* proyectado son de la forma:

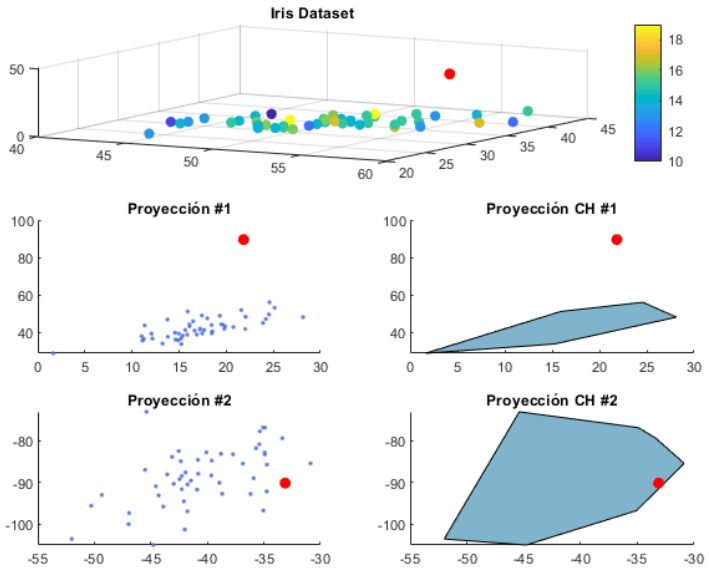

Figura 8.3 – Proyecciones del CH de la clase 1 del conjunto Iris de UCI.

$$\bar{v}^{\alpha} : \left\{ \bar{v_i} + \gamma_i \frac{(\bar{v_i} - \bar{c})}{\parallel \bar{v_i} - \bar{c} \parallel} \right\}, i = 1 \dots N \tag{8.3}$$

donde $\bar{c} = cP$ representa la proyección del centro c dada una matriz de proyeciones aleatoria P, $\bar{v_i}$ es el conjunto de vértices del *convex hull* proyectado y γ_i se define como:

$$\gamma_i = \frac{(v_i - c)^T P^T P (v_i - c)}{\parallel v_i - c \parallel} \alpha \tag{8.4}$$

donde v_i es el i-ésimo vértice del *CH* en el espacio original [7, p. 855-856]. En la Figura 8.4 puede verse un ejemplo del cálculo de los vértices del *ECP*. Como se observa, para valores de α positivos el *ECP* se expande y para valores negativos se contrae.

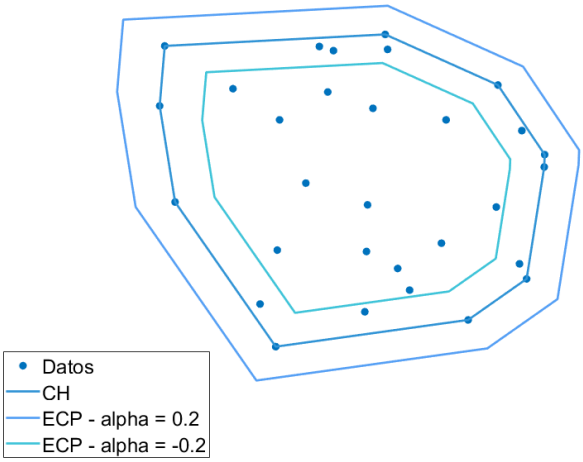

Figura 8.4 – Ejemplo del ECP en 2D.

El problema del cálculo de los vertices del *ECP* proyectado es que, dado que la norma no se conserva y que aparecen distintos parámetros γ para cada vértice proyectado, dichos parámetros pueden dar lugar a politopos no convexos, como muestra la Figura 8.5.

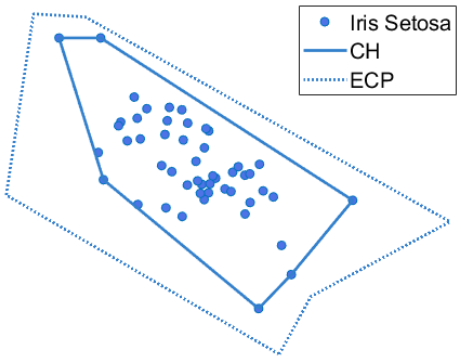

Figura 8.5 – Proyección de ECP no convexo sobre clase 1 del conjunto Iris.

Para solucionar esto, se propone la opción del uso del *Scaled Convex Polytope* o *SCH*, el cual solamente hace uso de un parámetro de expansión λ [9, p. 478-479].

Los vértices del *SCH* se definen teniendo en cuenta que $\lambda \in [0, \infty)$ de forma que $\bar{v}^{\lambda}$: $\{\lambda\bar{v}_i + (1-\lambda)\bar{c}\}\,,\ i = 1 \ldots N$.

En este método la expansión se produce después de proyectar todos los puntos del *CH*, de forma que además de utilizar solo un parámetro de expansión, también permite el uso de distintos tipos de centro. Los tipos de centro posibles son:

- La media de todos los puntos proyectados

$$\bar{c} = \frac{1}{N}\sum_i \bar{x_i}, \forall \bar{x_i} \in \bar{X} \tag{8.5}$$

- La media de los vértices del CH proyectados

$$\bar{c} = \frac{1}{N}\sum_i \bar{v_i}, \forall \bar{v_i} \in CH(\bar{X}) \tag{8.6}$$

- El centroide de los vértices proyectados

$$\bar{c} = \left(\frac{\sum x_i}{N}, \frac{\sum y_i}{N}\right), \forall\ \bar{v_i} = (x_i, y_i) \in CH(\bar{X}) \tag{8.7}$$

En la Figura 8.6 se muestra un ejemplo en dos dimensiones del *SCH*, en donde para $\lambda \in (0, 1)$ se contrae y para $\lambda \in (1, \infty)$ se expande.

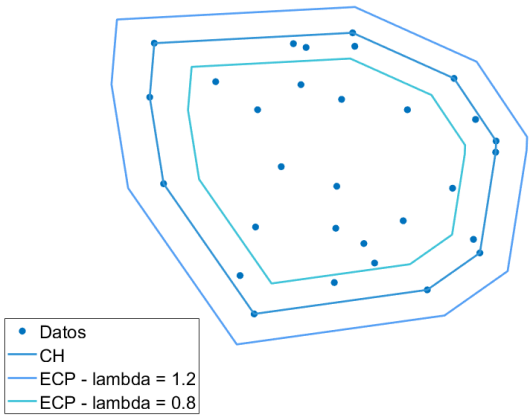

Figura 8.6 – Ejemplo del SHC en 2D.

8.2.1.2. K-Nearest Neighbors

El método *k-Nearest Neighbors* o *KNN* tiene como objetivo encontrar elementos dentro de un conjunto que están cerca unos de otros, suponiendo que éstos tienen relación entre sí. De este modo, un dato de test *z* es clasificado como perteneciente a la clase objetivo siempre y cuando la distancia a su vecino más cercano sea menor que un umbral, que será determinado en la fase de entrenamiento.

Para la estimación de la densidad local, y suponiendo un conjunto de datos d-dimensional, se centra una hiperesfera alrededor de un dato *z* de la clase objetivo. El volumen de esta hiperesfera se aumenta hasta que contiene *k* muestras del conjunto de entrenamiento [21, p. 69-70]. El cálculo de la densidad se estima mediante la fórmula:

$$p_{NN}(x) = \frac{k/N}{V_k(\parallel x - NN_k^{tr}(x) \parallel)} \tag{8.8}$$

en donde $NN_k^{tr}(x)$ representa el k-vecino más próximo de *x* en el conjunto de entrenamiento, *N* el número total de elementos del conjunto de entrenamiento y V_k el volumen de la hiperesfera que contiene al elemento *x* y a sus k-vecinos más próximos.

Suponiendo que $k = 1$, se considera que un dato de test *z* pertenece a la clase objetivo cuando su densidad local es mayor o igual a la densidad local de su primer vecino más próximo en el conjunto de entrenamiento. La ecuación 8.9 muestra la función indicatriz de un clasificador *KNN* para $k = 1$ en función de la densidad local.

$$f_{NN^{tr}}(z) = I\left(\frac{1/N}{V(\parallel z - NN^{tr}(z) \parallel)} \geq \frac{1/N}{V(\parallel NN^{tr}(z) - NN^{tr}(NN^{tr}(z)) \parallel)}\right) \tag{8.9}$$

que simplificado equivale a:

$$\begin{aligned} f_{NN^{tr}}(z) &= I\left(\frac{V(\parallel z - NN^{tr}(z) \parallel)}{V(\parallel NN^{tr}(z) - NN^{tr}(NN^{tr}(z)) \parallel)} \leq 1\right) \\ &= I\left(\frac{\parallel z - NN^{tr}(z) \parallel}{\parallel NN^{tr}(z) - NN^{tr}(NN^{tr}(z)) \parallel} \leq 1\right) \end{aligned} \tag{8.10}$$

Teniendo en cuenta lo anterior, se puede ver que el método se reduce a comparar la distancia entre la muestra *z* a su vecino más próximo $NN^{tr}(z)$ con la distancia de ese vecino a su vecino más próximo $NN^{tr}(NN^{tr}(z))$, de manera que no es necesario el cálculo explícito de la densidad. En la Figura 8.7 se puede ver un ejemplo para $k = 1$ en el que, teniendo un punto de test representado por el punto rojo, se calcula la distancia a su vecino $d_1 = \parallel z - NN^{tr}(z) \parallel$ más próximo en el conjunto de entrenamiento y, a la vez, la distancia de éste a su vecino más próximo $d_2 = \parallel NN^{tr}(z) - NN^{tr}(NN^{tr}(z)) \parallel$. La clasificación del objeto se hará de forma que:

$$f_{NN^{tr}}(z) = I\left(\frac{d_1}{d_2} \leq \varepsilon\right) \tag{8.11}$$

donde ε es el umbral fijado durante el entrenamiento. Para este ejemplo es fácil ver que, si se fija $\varepsilon = 1$, el punto de test se clasificaría como *outlier* [16].

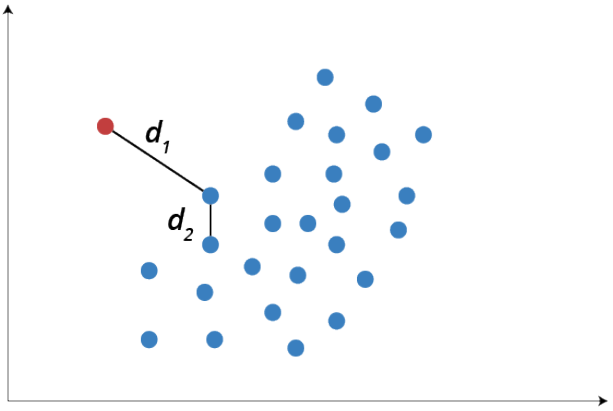

Figura 8.7 – Método Nearest Neighbor.

8.2.1.3. K-Centers

En el método K-Centers, en contraposición con KNN, se busca cubrir el conjunto de datos de entrenamiento con *k* hiperesferas (en el caso de *d* dimensiones) de igual radio. Estas hiperesferas se sitúan sobre los datos de entrenamiento de forma que se consiga minimizar la máxima distancia de todas las distancias mínimas entre los datos de entrenamiento x_i y los centros o prototipos μ_k [21, p. 68]. De esta forma, la función de error que se minimiza es como sigue:

$$\varepsilon_{K-Centers} = \max_i(\min_k \parallel x_i - \mu_k \parallel^2) \tag{8.12}$$

La Ecuación 8.12 se puede desglosar como dos procesos: en el primero se calculan, según el número de clusters *k*, los elementos más cercanos a cada centro μ_k, de manera que se hace una partición de los datos en el número de clusters establecido. En la Figura 8.8 se muestra un ejemplo del cálculo de las distancias entre uno de los centros μ_i (cuadrado rojo) y cada elemento x_i (puntos azules), de modo que los más cercanos se asignarán a ese centro. Para el otro centro el proceso se desarrolla de forma análoga.
El segundo de los procesos se encarga de posicionar de forma óptima cada centro, de manera que se calculará, de entre los elementos asignados a él, el que presenta la distancia mayor, y se tratará de minimizar esa distancia, como se muestra en la Figura 8.9. Se puede ver que los primeros centros, iniciados aleatoriamente (cuadrados rojos), se sitúan a una distancia máxima de uno de los elementos (línea roja) y que, para minimizar esa distancia (y en posteriores iteraciones las del resto de puntos de mayor distancia), se debe escoger el punto rodeado por un círculo rojo como el centro óptimo.

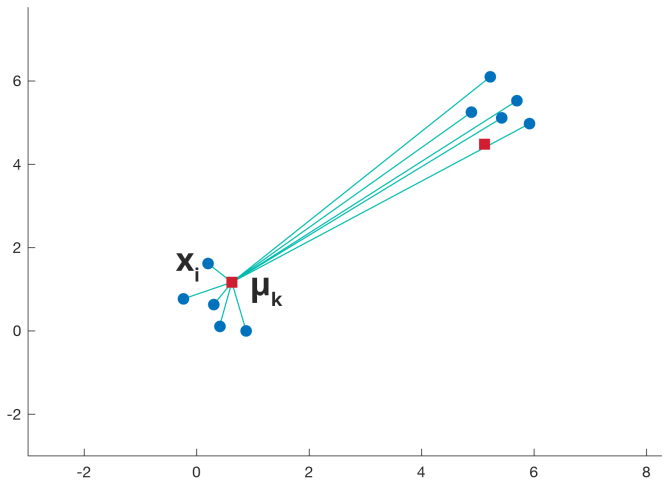

Figura 8.8 – Asignación de centros.

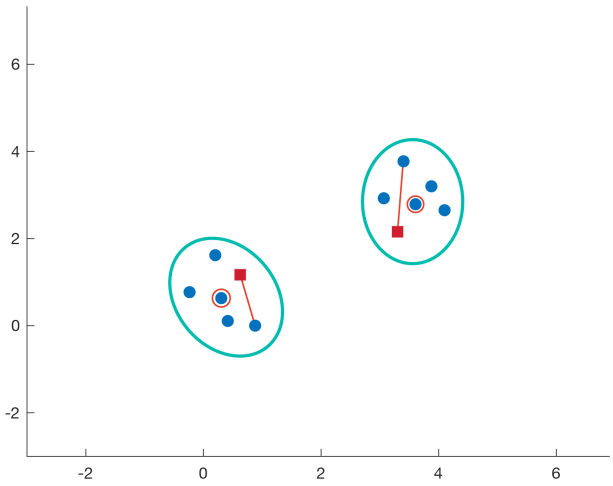

Figura 8.9 – Cálculo del centro óptimo.

Los centros son inicializados de manera aleatoria de forma que, para evitar los mínimos locales y llegar a la solución óptima, se repetirá esta inicialización varias veces, y se utilizará la mejor solución de todas según el mínimo error. El radio se determina según la máxima distancia a los datos que cada hiperesfera debe contener.

Después del entrenamiento, es posible calcular la distancia de un dato de test *z* al conjunto de datos objetivo, que quedará definida por la Ecuación 8.13

$$d_{K-Centers} = \min_{k} \parallel z - \mu_k \parallel^2 \tag{8.13}$$

8.2.1.4. Support Vector Machines

Las máquinas de vectores de soporte o *SVM* son un conjunto de algoritmos de aprendizaje no supervisado utilizados ampliamente para problemas de clasificación y regresión. En este trabajo se abordará la interpretación de Schölkopf [19] para el caso de la clasificación one-class.

Dado un conjunto de muestras $\{x_i\}, x_i \in \mathbb{R}^n$, cada punto del conjunto se transforma mediante una función de mapeado o función kernel $\phi : \mathbb{R}^n \to \mathcal{K}$ del espacio característico $\mathbb{R}^n$, a un espacio kernel de mayor dimensión $\mathcal{K}$, generado por el kernel $k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$. El nuevo espacio kernel sirve para simplificar el contorno que divide los datos, que en el espacio característico tiene una forma irregular (Figura 8.10), convirtiendo esta división en el cálculo de un hiperplano que separa los datos objetivo de los *outlier* en el espacio kernel (Figura 8.11).

De esta forma, para llevar a cabo la clasificación, el algoritmo *OCSVM* encuentra el hiperplano en el espacio kernel que separa los datos del origen con el máximo margen. En caso de que ese hiperplano no exista, se hace uso de una variable ξ_i, para permitir que algunos *outliers* se sitúen dentro del margen, y un parámetro $\nu \in [0, 1]$, que controla el coste que suponen esas anomalías. En general, ν proporciona un límite superior en la fracción de *outliers*.

Este hiperplano introducirá una superficie no lineal en el espacio característico.

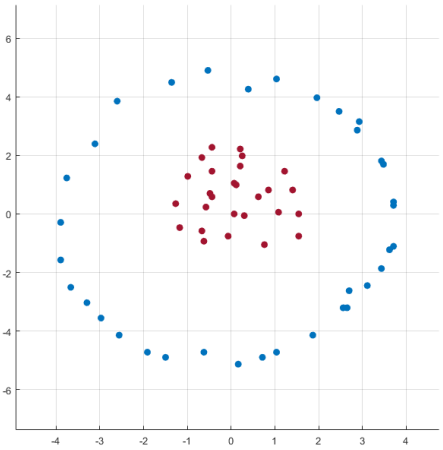

Figura 8.10 – Conjunto de datos en el espacio característico.

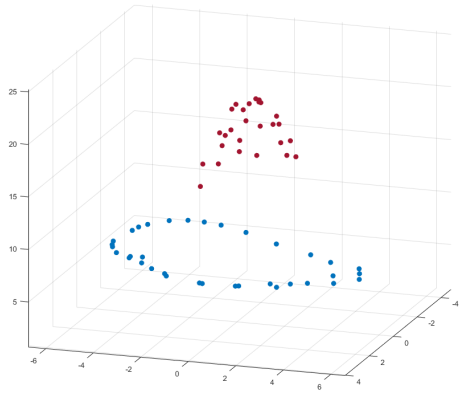

Figura 8.11 – Conjunto de datos en el espacio kernel.

Para el cálculo del hiperplano, este algoritmo resuelve la siguiente programación cuadrática:

$$\begin{aligned} &\min_{w,\xi,\rho} \left(\frac{1}{2} \parallel w \parallel + \frac{1}{\nu m} \sum_{i=1}^{m} \xi_i - \rho \right) \\ &con\ la\ restricción\ w \cdot \phi(x_i) \geq \rho - \xi_i, \\ &i = 1, ..., m;\ \xi_i \geq 0 \end{aligned} \tag{8.14}$$

donde ω es el vector normal al hiperplano y ρ es el margen.

8.2.1.5. Support Vector Data Description

El método *Support Vector Data Description* o *SVDD* es el último de los métodos de modelado de contorno analizados en este documento, y fue descrito por primera vez por Tax y Duin [22]. La idea principal parte de una adaptación del concepto de *SVM* a la clasificación one-class. La diferencia de este algoritmo con *SVM* es que para calcular el límite de decisión, en lugar de utilizar un hiperplano, se hace uso de una hiperesfera. La esfera está caracterizada por su centro *a* y el radio *R*, y se pone como condición que todos los datos objetivo del conjunto de entrenamiento queden contenidos dentro de ella. Al imponer esta condición se deduce que el error empírico es 0. El error estructural se define como:

$$\varepsilon_{struct}(R, a) = R^2 \tag{8.15}$$

que se debe minimizar siguiendo la restricción

$$\| x_i - a \|^2 \leq R^2;\ \forall i \tag{8.16}$$

Para poder tener en cuenta la posibilidad de que existan *outliers* en el conjunto de entrenamiento, es necesario introducir una variable ξ que permita tener en cuenta esta variabilidad. Esto hace que el error empírico no sea necesariamente 0, lo que se traduce en que el nuevo error tenga una parte empírica y otra estructural [21, p.22]:

$$\varepsilon_{struct}(R, a, \xi) = R^2 + C \sum_{i=1}^{N} \xi_i \tag{8.17}$$

que debe ser minimizado teniendo en cuenta la nueva restricción

$$\| x_i - a \|^2 \leq R^2 + \xi_i;\ \xi_i \geq 0, \forall i \tag{8.18}$$

donde *C* es un parámetro que permite la compensación entre el volumen de la descripción y el error. Una vez resuelto el problema de minimización se obtendrá un modelo del límite de decisión con el cual se podrá ver si un nuevo objeto *z* pertenece o no a la clase objetivo. El objeto será considerado perteneciente a la clase objetivo cuando su distancia al centro a de la hiperesfera sea menor o igual que el radio. La Ecuación 8.19 muestra la función indicatriz de un clasificador SVDD.

$$f_{SVDD} = I\left(\| z - a \|^2 \geq R^2\right) \tag{8.19}$$

El uso de esta técnica como se ha descrito hasta ahora puede aportar malos resultados, ya que el modelo de límite en forma de hiperesfera resulta poco flexible, llevando a una malo rendimiento en conjuntos de datos que no siguen una estructura esférica. Para ello, se plantea la posibilidad de introducir diferentes kernels que permiten el mapeo de los datos en una nueva representación, de forma equivalente a como se ha visto en *SVM*, lo que hace que se ajusten mejor a la hiperesfera. En el caso de la función *svdd* de la *toolbox PRTools*, el kernel utilizado es el *RBF* o *Radial Basis Function* definido como:

$$K(x_i, x_j) = exp\left(-\frac{\| x_i - x_j \|^2}{2\sigma^2}\right) \tag{8.20}$$

El kernel *RFB* tiene en cuenta únicamente las distancias entre objetos del conjunto de entrenamiento. El parámetro σ permite cambiar el número de vectores de soporte, lo que hace que el límite se ajuste más a los datos, alejándose así de la forma esférica original. Se distinguen tres casos dependiendo del valor de σ [21, p. 33]:

- Para valores de σ cercanos a la mínima distancia entre vecinos, todos los objetos se

comportan como vectores de soporte, albergando cada uno de ellos una distribución Gaussiana, lo que es equivalente a la descripción de Parzen con un valor pequeño del ancho de ventana.

- Para valores de σ del orden de la máxima distancia entre vecinos, los vectores de soporte se reducen, de modo que la forma final del límite del clasificador es similar a la esfera original obtenida sin aplicar el kernel.

- Para valores intermedios de σ, los vectores de soporte serán los objetos del borde del conjunto que se diferencien más del resto en términos de distancia.

En la Sección 8.4.2 se verá un ejemplo de la influencia de este parámetro sobre el límite de decisión del clasificador.

8.2.1.6. Minimum Spanning Tree Data Description

Este método se basa en el modelado de la estructura intrínseca del conjunto objetivo mediante el uso de un árbol recubridor mínimo o *minimum spanning tree* (*MST*). Este árbol es un tipo de grafo totalmente conexo y no dirigido que une todos los puntos del conjunto mediante aristas a las que se asocia un peso. Para que sea un *MST* es necesario que no haya ningún bucle y que las distancias entre ejes $e_{ij} = (x_i, x_j)$ (parejas de puntos que se unen mediante una arista) sean mínimas.

Sean $\{x_i, x_j\} \in X \subset \mathbb{R}^n$ dos puntos de la clase objetivo. Si estos dos puntos describen a dos objetos similares en la realidad, también lo hacen en el espacio de representación $\mathbb{R}^n$. Asumiendo que los puntos vecinos a estos también pertenecen a la clase objetivo, se llega a que es posible encontrar una transformación entre los dos puntos, para la cual todos los puntos que recorran ese camino también pertenezcan a la clase objetivo.

Dado un conjunto objetivo X compuesto por *n* muestras $x_i \in \mathbb{R}^N \ i = 1, 2, ..., n$, y sea $G = (X, E)$ un grafo totalmente conexo y no dirigido definido en X, y $E = \{e_{ij}\}$ el conjunto de todos los vértices $e_{ij} = (x_i, x_j)$. Para cada eje se asocia un peso w_{ij} que es igual a la longitud del mismo $w_{ij} = \| e_{ij} \| = \| x_i - x_j \|$ [15]. Como se ha mencionado anteriormente, el objetivo es encontrar un subgrafo $\mathcal{G}$ sin bucles tal que la suma total de los pesos, o la suma de las longitudes de los vértices, sea mínima. Dado que los pesos son simétricos (es un grafo no dirigido), es suficiente con considerar los pesos para los cuales $i > j$ o viceversa. Para encontrar este subgrafo o *MST* existen diversos algoritmos, como por ejemplo el de Kruskal [13].

Teniendo el conjunto de datos objetivo mostrado en la Figura 8.12, se quiere construir un clasificador basado en *MST*. Para ello se lleva a cabo el proceso descrito anteriormente, y se obtiene con ello el grafo deseado, como muestra la Figura 8.13.

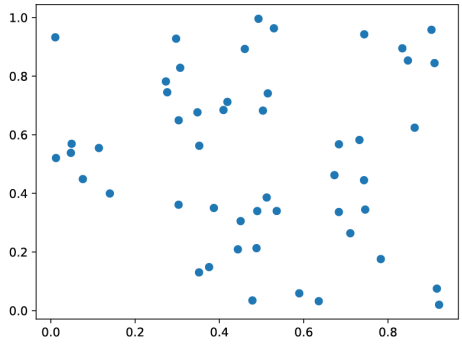

Figura 8.12 – Conjunto de datos objetivo.

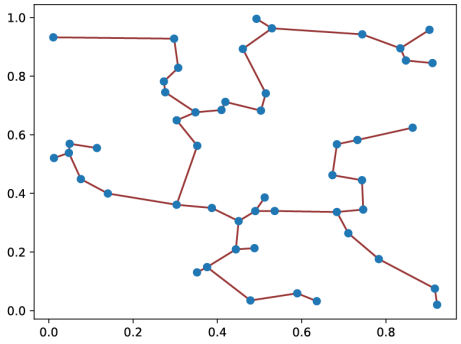

Figura 8.13 – MST del conjunto de datos.

Una vez se tiene la descripción del conjunto de datos objetivo mediante el *MST*, y dado un dato de test z, el proceso de clasificación se lleva a cabo comparando la distancia d a la proyección z_p de éste sobre la arista del *MST* más cercana, como muestra la Figura 8.14. De esta forma se comparará después la distancia con un umbral θ, calculado durante la fase de entrenamiento del clasificador. Este umbral se calcula en base a la función cuantil de la distribución de pesos, y por lo tanto, en base a un conjunto de distancias entre objetos vecinos en el *MST*.

Se tendrá que para valores $\| z - z_p \| < \theta$ el punto se clasificará como perteneciente a la clase objetivo, mientras que de lo contrario se considerará como *outlier*.

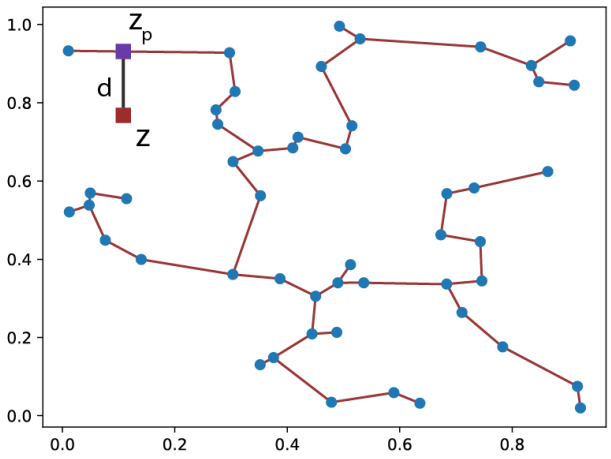

Figura 8.14 – Proceso de test.

8.2.2. Métodos de estimación de densidad

En este conjunto de métodos, se lleva a cabo una estimación de la densidad de los datos de entrenamiento, para lo cual se supone a priori que los datos se ajustan a una distribución

(normal, Poisson, etc). Una vez calculada la densidad del conjunto objetivo durante el entrenamiento, se fija un umbral de esa densidad, mediante el cual los nuevos datos se clasificarán como pertenecientes a la clase objetivo (si superan el umbral), o como *outliers*.

8.2.2.1. Gauss

El método de distribución Gaussiana es el más simple de los métodos de cálculo de densidad. La densidad de probabilidad de un conjunto $X \subset \mathbb{R}^n$ viene dada por la Ecuación 8.21.[21, p. 65]

$$p_{\mathcal{N}}(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{8.21}$$

donde $x \in X$, μ es la media y Σ la matriz de covarianzas.
La función objetivo de este algoritmo es, por lo tanto, el cálculo de los parámetros μ y Σ. Según el método de máxima verosimilitud [4], se tiene que la media y la covarianza de las muestras son un buen estimador de la media y covarianza de la población. Por lo tanto, una vez calculada la función de distribución $p_{\mathcal{N}}(x;\mu,\Sigma)$, se puede fijar un umbral para determinar si un dato es *outlier* o no. Ese umbral está representado por el parámetro *FracReg*, que representa la fracción de datos objetivo rechazados durante la fase de entrenamiento. Esto se ejemplifica en la Figura 8.15.

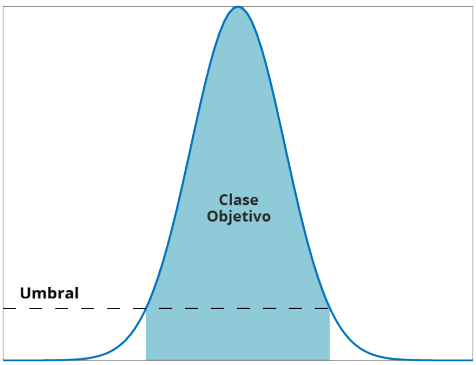

Figura 8.15 – Distribucion Gaussiana.

En la Figura 8.16 se puede ver cómo afecta al límite de decisión la variación del parámetro *FracReg* para un conjunto de dos dimensiones.

A pesar de ser un método muy sencillo, puede presentar un problema de cálculo: la inversión de la matriz de covarianza. Para esto se introduce el parámetro de regularización, que añade una pequeña constante a la diagonal de la matriz. En este caso se seguirá la Ecuación 8.22.

$$\Sigma' = (1-\lambda)\Sigma + \lambda I \tag{8.22}$$

donde Σ' es la nueva matriz de covarianza, λ el parámetro de regularización e I la matriz

identidad.

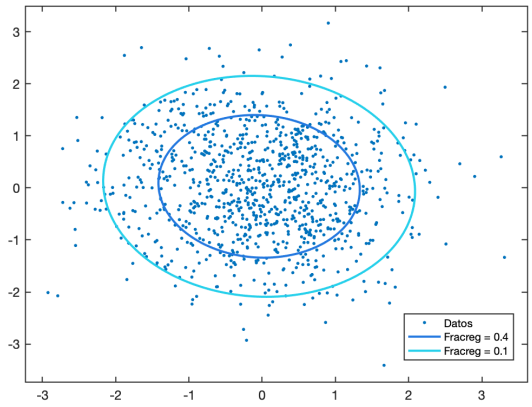

Figura 8.16 – Influencia del parámetro FracReg.

8.2.2.2. Parzen

Otro de los métodos de estimación de densidad es el método Parzen [18], que es una extensión del método de combinación de Gaussianas. La estimación de la densidad se lleva a cabo mediante una combinación de kernels Gaussianos centrados en objetos individuales de entrenamiento [21, p. 67], con matrices de covarianzas diagonales $\Sigma_i = hI$. La Ecuación 8.23 muestra el cálculo de la función de estimación de densidad de Parzen.

$$p_p(x) = \frac{1}{N} \sum_{i=1}^{N} p_{\mathcal{N}}(x; x_i, hI) \tag{8.23}$$

En este método, además de los parámetros vistos en el método anterior, se añade el ancho (h) de la ventana de Parzen. En la Figura 8.17 se muestra un ejemplo del cálculo de la función de densidad para cada punto x_i, dado un ancho de ventana r. El proceso es similar al de *KNN*. Haciendo una analogía con la Física, mientras que en *KNN* se mantiene constante la masa, en Parzen se mantiene constante el volumen. De este modo, para cada hipesfera de radio r, igual al ancho de ventana, se estima la densidad de forma análoga a lo visto en el método 8.2.2.1, obteniendo así una estimación para cada una de las *N* hiperesferas. La densidad final del conjunto será la suma de todas.

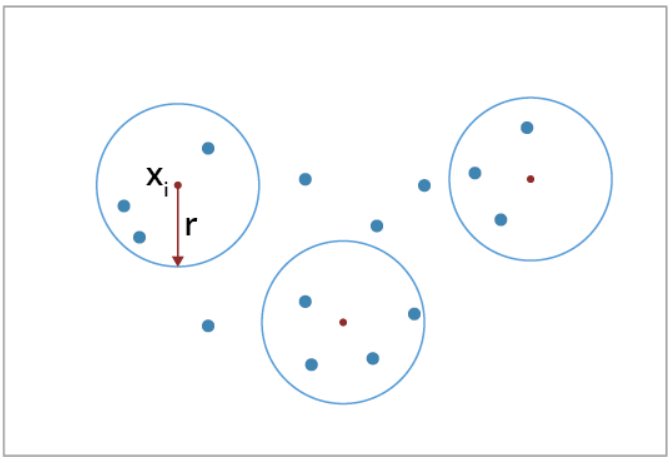

Figura 8.17 – Método Parzen.

En la Figura 8.18 se muestra un ejemplo del límite de decisión del clasificador Parzen en dos dimensiones, para un conjunto de datos con forma de banana.

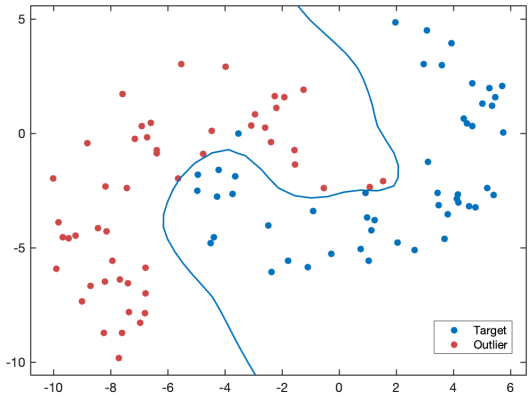

Figura 8.18 – Límite de decisión en clasificador Parzen.

8.2.3. Métodos de reconstrucción

Los métodos de reconstrucción surgen con el objetivo de modelar los datos sin tener ninguna información previa sobre ellos, por lo que los algoritmos englobados en esta categoría pertenecen a priori al aprendizaje no supervisado. En esta sección se analizará la adaptación de algunos de estos algoritmos a la clasificación one-class.

8.2.3.1. Principal Component Analysis

Principal Component Analysis o PCA es un método estadístico utilizado en el análisis de datos multivariables. Su aplicación se extiende a una gran cantidad de ramas del conocimiento como herramienta para la reducción dimensional o para clustering. También se utiliza, dentro del ámbito del machine learning, en problemas de clasificación y detección de anomalías [24].

Aunque tengan un objetivo distinto, las aplicaciones de PCA se centran en encontrar algún

tipo de relación entre los datos. Para ello se busca el subespacio ortonormal que refleje la mayor variación entre las distintas variables [21, p.76]. En este método se calcula los autovectores de la matriz de covarianza de los datos de entrenamiento, cuyos autovalores son máximos, para después construir con ellos una base $\mathcal{W}$ del subespacio generado, sobre la que se proyectarán los datos. Dado un objeto *z*, para comprobar si pertenece al conjunto de datos objetivo, se calculará el error de reconstrucción. Para ello se calculará primero su proyección en el subespacio:

$$z_p = \mathcal{W}(\mathcal{W}^T\mathcal{W})^{-1}\mathcal{W}^T z \tag{8.24}$$

A continuación, con el objeto proyectado se calcula el error de reconstrucción. En la *toolbox PRTools* el error de reconstrucción se calcula de la forma:

$$f(z) = \| \, z - z_p \, \|^2 \tag{8.25}$$

Por defecto se utilizan los *k* autovectores con los mayores autovalores, aunque existe la posibilidad de utilizar los autovectores con los autovalores más pequeños. A continuación se podrá ver un ejemplo del cálculo de las componentes principales del conjunto de datos de dos dimensiones mostrado en la Figura 8.19.

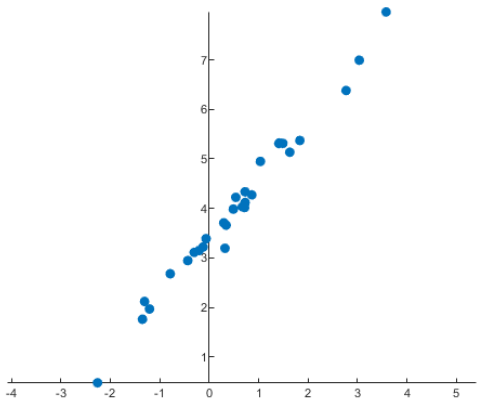

Figura 8.19 – Conjunto de datos 2D.

El primer paso es calcular la matriz de covarianza del conjunto, que en este caso resulta:

$$C = \begin{pmatrix} 1{,}6901 & 2{,}0186 \\ 2{,}0186 & 2{,}4543 \end{pmatrix}$$

Después se deben calcular los autovectores y autovalores de dicha matriz:

$$V = \begin{pmatrix} -0{,}7701 & 0{,}6380 \\ 0{,}6380 & 0{,}7701 \end{pmatrix} \quad D = \begin{pmatrix} 0{,}0177 & 0 \\ 0 & 4{,}1266 \end{pmatrix}$$

Con los datos anteriores es posible construir las componentes principales del conjunto de datos, y calcular la variación para cada una de ellas. La Figura 8.20 muestra las componentes como dos líneas rojas sobre el conjunto de datos. En este caso la componente en la que se produce la mayor variación es la que se sitúa en el primer cuadrante. Para conjuntos de datos de mayor dimensión se tendrán solo en cuenta las dos componentes con mayor variación.

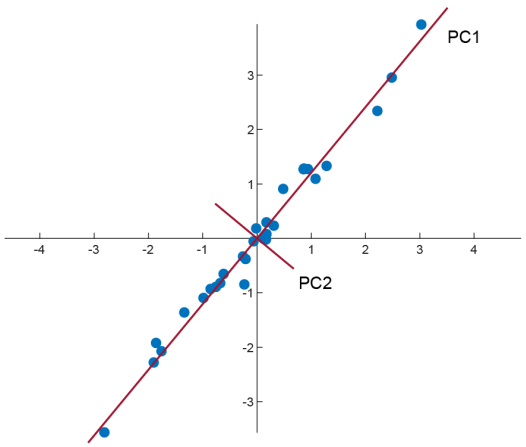

Figura 8.20 – Componentes principales.

El porcentaje de variación de los datos en cada componente se calcula dividiendo el autovalor correspondiente por la suma de todos los autovalores. En este ejemplo la variación de la primera componente corresponde al 99.57 %, mientras que la de la segunda componente es del 0,43 %.

Para comprobar si un punto de test *z* (cuadrado rojo) pertenece a la clase objetivo del conjunto de datos del ejemplo, se calcula el cuadrado de la distancia d de ese punto a su proyección sobre la componente principal z_p (cuadrado morado), como muestra la Figura 8.21. Si ese valor es menor o igual que el del límite establecido al entrenar el clasificador, se tendrá que el punto pertenece a la clase objetivo, de lo contrario se considerará como *outlier*.

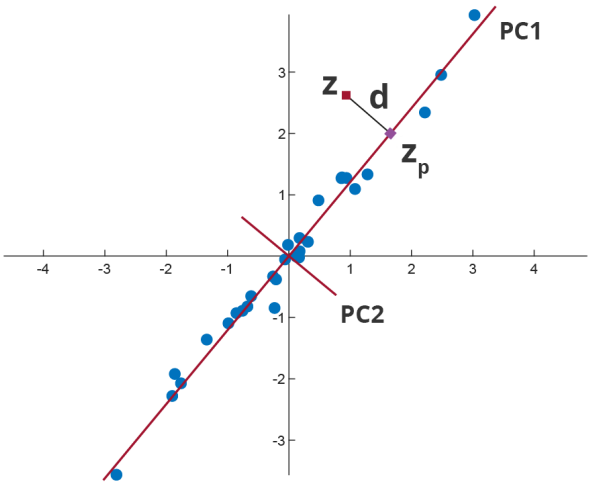

Figura 8.21 – Evaluación de dato de test.

8.2.3.2. K-Means

El método de clustering K-Means es similar a K-Centers, con diferencias en la función de error que se minimiza. En K-Means se calculan la media de las distancias de los objetos a los prototipos, de forma que resulta más robusto frente a variaciones grandes en los *outliers*. A diferencia de K-Centers, este método no fija los centros en los objetos de entrenamiento [21, p.73]. En este método se presupone que los datos están agrupados en *clusters*, y que se pueden caracterizar por una serie de objetos prototipo μ_k. Estos prototipos crean una partición de todo el espacio característico.

Este algoritmo posiciona los objetos prototipo de forma que se minimiza el siguiente error:

$$\varepsilon_{k-means} = \sum_i \left(\min_k \parallel x_i - \mu_k \parallel^2 \right) \tag{8.26}$$

Dado un objeto *z*, su distancia al conjunto objetivo se define como la distancia cuadrática del objeto al prototipo más cercano:

$$d_{K-Means} = \min_k \parallel z - \mu_k \parallel^2 \tag{8.27}$$

En la Figura 8.22 se puede ver la diferencia entre un clasificador K-Means (línea continua) y uno K-Centers (línea discontinua).

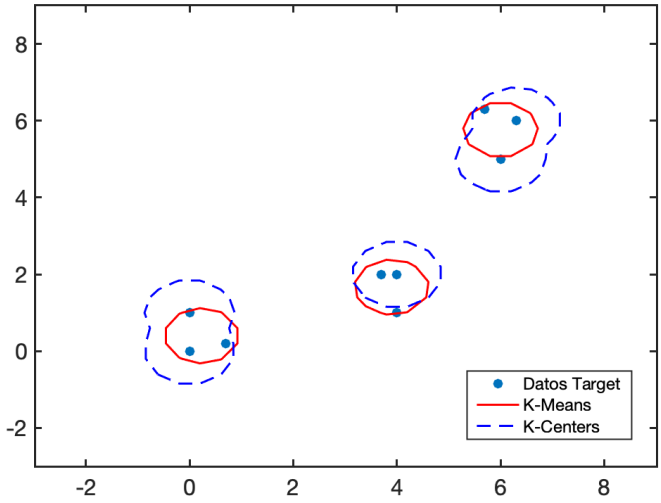

Figura 8.22 – Clasificador K-Means vs K-Centers.

8.2.3.3. Autoencoders

Los *autoencoders* son un tipo de red neuronal no supervisada que se basa en la reducción dimensional o compresión de la información, que posteriormente se descomprime para recrear los datos originales, de modo que la representación final sea lo más cercana posible a la original [12]. Durante este proceso, gracias al propio diseño de la red, también se consigue reducir el ruido entre los datos durante la etapa de aprendizaje.

En la Figura 8.23 se puede ver la arquitectura de una red autoencoder, en la que se diferencian dos etapas:

- Etapa de codificación: se compone por la capa de entrada, en la que se introducen los datos; una o más capas ocultas de reducción dimensional y una última capa de cuello de botella, en la que se tiene una representación comprimida de los datos originales.

- Etapa de decodificación: en la que, a partir de la capa de cuello de botella, se descomprime la información pasándola por una o más capas ocultas, para mostrarse en la capa de salida con la misma dimensión que a la entrada de la red.

La capa oculta de cuello de botella cuenta con un número de neuronas h_{auto} ocultas [21, p.78].

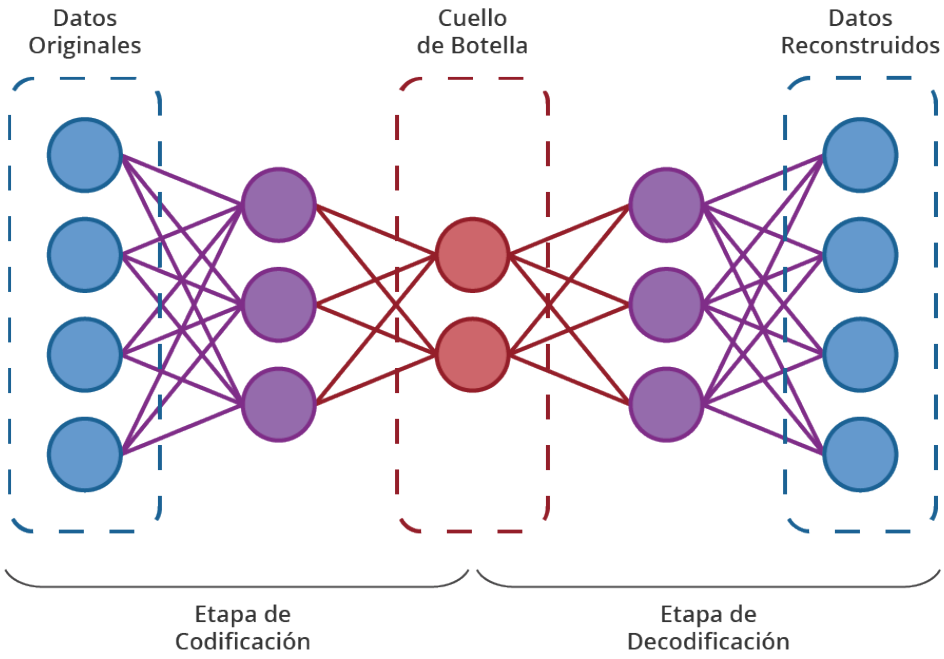

Figura 8.23 – Arquitectura red neuronal autoencoder.

La función de activación de todas las capas es una sigmoide. Durante el entrenamiento se minimizará el error cuadrático medio entre la representación original y su reconstrucción, y se fijará un umbral de dicho error. Después del proceso de reconstrucción, el error que presenten los datos *outlier* debe ser superior al presentado por los datos objetivo. Dado un nuevo dato de test *z*, el autoencoder lo clasificará como *outlier* si su error de reconstrucción supera el umbral establecido en el entrenamiento. El error de reconstrucción se calcula como muestra la Ecuación 8.28.

$$\varepsilon_{auto} = \| f_{auto}(z; w) - z \|^2 \tag{8.28}$$

donde $f_{auto}(z; w)$ representa la salida reconstruida por la red.

8.3. Análisis del comportamiento de un clasificador

En esta sección se llevará a cabo un análisis de las distintas técnicas de evaluación del rendimiento de los clasificadores, y de qué métricas son las más adecuadas para la clasificación one-class.

A la hora de evaluar un clasificador existen distintos parámetros que pueden ser útiles para hacerse una idea del rendimiento del mismo. En el caso de la clasificación one-class, el clasificador debe distinguir únicamente si un objeto pertenece a una clase: la clase objetivo. Teniendo en cuenta esto, y dado un dato nuevo de test, existen cuatro posibilidades a la hora de su clasificación:

- Verdadero positivo: dato perteneciente a la clase objetivo que se clasifica como tal.

- Falso negativo: dato perteneciente a la clase objetivo que se clasifica como *outlier*.
- Verdadero negativo: dato *outlier* que se clasifica como tal.
- Falso negativo: dato perteneciente a la clase target que se clasifica como *outlier*.

Estas posibilidades se recogen en la llamada matriz de confusión, ilustrada en la Figura 8.24.

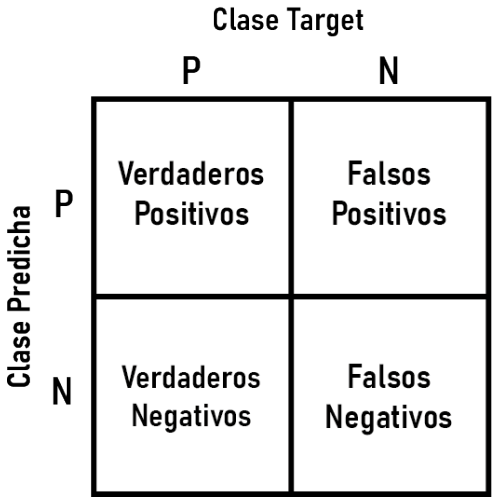

Figura 8.24 – Matriz de confusión.

Algunas métricas comunes que se pueden deducir de la matriz de confusión son las siguientes:

- *True Positive Rate*: Mide la relación entre verdaderos positivos y el total de datos pertenecientes a la clase positiva. También es conocido como *recall* o *sensitivity*.

$$True\ Positive\ Rate = \frac{TP}{P} \tag{8.29}$$

- *False Positive Rate*: Mide la relación entre falsos positivos y el total de datos pertenecientes a la clase negativa. También es conocido como *false alarm rate*.

$$False\ Positive\ Rate = \frac{FP}{N} \tag{8.30}$$

- *Precision*: Representa la fracción de elementos positivos clasificados correctamente. No tiene en cuenta los elementos de la clase negativa predichos por el clasificador. También se conoce como *positive predictive value* o PPV.

$$Precision = \frac{TP}{TP + FP} \tag{8.31}$$

- *Accuracy*: Determina cuántos elementos se han clasificado correctamente en total, te-

niendo en cuenta ambas clases.

$$Accuracy = \frac{TP + TN}{P + N} \tag{8.32}$$

- *Specificity*: Representa la fracción de elementos negativos clasificados correctamente. No tiene en cuenta los elementos de la clase positiva.

$$Specificity = \frac{TN}{FP + TN} = 1 - fp\,rate \tag{8.33}$$

- *Negative Predictive Value*: Representa la fracción de de los elementos predichos como negativos que realmente lo son.

$$NPV = \frac{TN}{TN + FN} \tag{8.34}$$

- F_1 *Score*: Esta métrica es una combinación de la precision y el recall en forma de media armónica.

$$F_1 = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision} \tag{8.35}$$

8.3.1. Análisis ROC

Las métricas *precision*, *accuracy* y *specificity* tienen un problema: varían dependiendo de la distribución de las clases, lo cual es especialmente importante en la clasificación one-class, ya que los datos de situaciones anómalas (*outliers*) pueden ser muy pocos. En la Tabla 8.1 se muestra un ejemplo de un mismo clasificador entrenado sobre dos conjuntos de datos A y B. De este ejemplo se puede ver que, al cambiar la distribución de la clase objetivo, se produce un cambio en las métricas de rendimiento del clasificador.

Conjunto	TP	FN	TN	FP	TPR	FPR	Prec.	Acc.
A	10	5	15	1	0.66	0.06	0.91	0.81
B	100	50	15	1	0.66	0.06	0.99	0.69

Tabla 8.1 – Rendimiento de un mismo clasificador dependiendo de la distribución de la clase.

Por lo tanto, para evitar esa variación se utiliza la curva *ROC* (Receiving Operating Characteristic), la cual relaciona el tp_r con el fp_r, que no dependen de la distribución de las clases [8]. En la Figura 8.25 se muestra una curva *ROC* calculada. Cada punto de la curva representa un valor distinto del par (tp_r, fp_r), para diferentes valores del umbral de rechazo de datos target en el entrenamiento del clasificador, de los cuales se obtienen distintos valores de la matriz de confusión.

Para manejar de forma más sencilla la curva *ROC* es útil convertirla a un valor escalar. Para esto, se calcula el *AUC* o Area Under the Curve de la curva *ROC*. Este valor representa

la probabilidad de que el clasificador asigne un valor positivo aleatorio sea clasificado como tal.

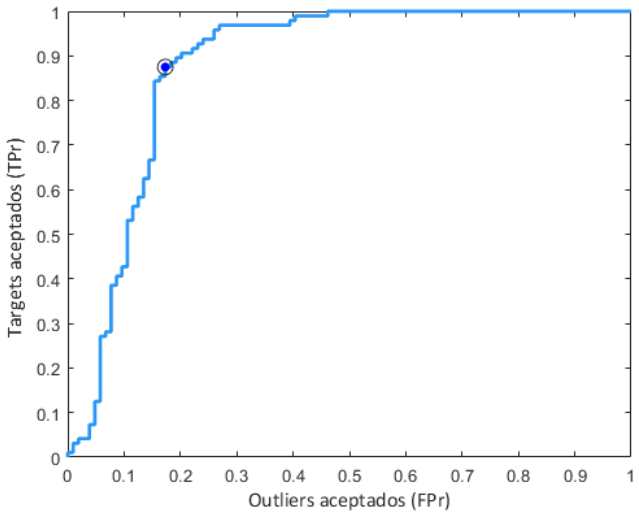

Figura 8.25 – Curva ROC.

Cuando el umbral de *outliers* considerado durante el entrenamiento no varía, la curva *ROC* tiene únicamente un punto, por lo que el *AUC* es el área del trapezoide formado por los puntos $\{(0,0),(fpr,tpr),(1,1),(0,1)\}$. En la Figura 8.26 se muestra el *AUC* para un clasificador discreto con $fp_r = 0,05,\ tp_r = 0,95$. El rendimiento del clasificador es mejor cuanto mayor sea el *AUC*.

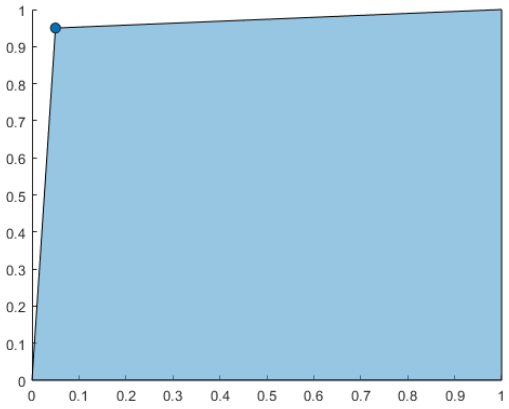

Figura 8.26 – Area Under ROC Curve.

8.4. Adaptación de algoritmos

En esta sección se detallará el método de entrenamiento y validación de los clasificadores, así como las adaptaciones de los algoritmos descritos anteriormente, al entorno de programación Matlab. Se especificarán los hiperparámetros de cada clasificador, así como los datos proporcionados a la salida de cada uno de los algoritmos.

La Figura 8.27 presenta la estructura general del programa implementado en Matlab. Se puede ver que todos los métodos serán llamados desde una función global (Anexo 11), la cual se encargará también de normalizar los datos de entrada y de guardar y presentar los datos finales. Esta función global tendrá como único parámetro de entrada el conjunto de datos.

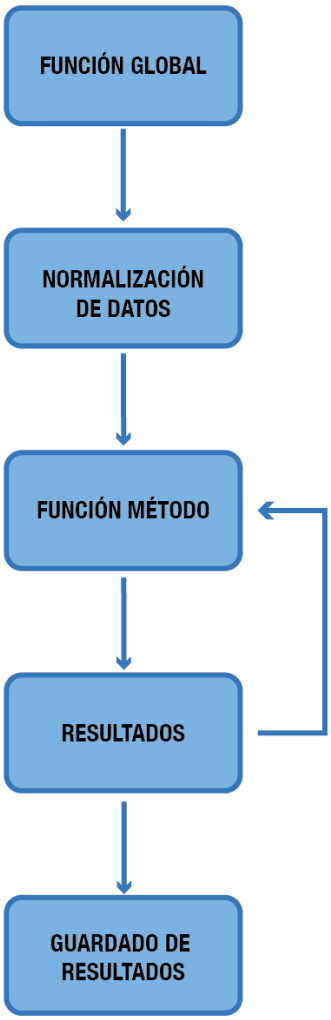

Figura 8.27 – Estructura general del programa de Matlab.

La ejecución del programa se realiza, siguiendo la Figura 8.27 mencinada anteriormente, en los siguientes pasos:

1. Se realiza la llamada a la función global, pasando como argumento el nombre del archivo, con extensión *.mat*, que contiene el conjunto de datos.
2. Dentro de la función global se realiza la llamada a una función que realiza la normalización de los datos.
3. Para cada método se realiza la llamada a su función principal que, tras entrenar y validar el clasificador, devuelve los resultados.

4. Por último, una vez obtenidos todos los resultados en el paso anterior, se guardan en un archivo *.mat* y se presentan en pantalla los valores de AUC y tiempo de entrenamiento.

8.4.1. Entrenamiento y validación

El objetivo del entrenamiento es encontrar el clasificador con mayor rendimiento posible. Para ello, se llevará a cabo un barrido de los parámetros de cada modelo (fracción de *outliers*, número de centros, neuronas de la capa oculta, etc), de forma que se encuentre la combinación de éstos que genere el mejor clasificador.

Para comprobar el rendimiento del clasificador durante la etapa de entrenamiento existen dos posibilidades: que se cuente con datos *outlier* reales o no. En caso de que no se cuente con datos *outlier*, será necesario generarlos de manera artificial. El proceso de evaluación del clasificador se dividirá en dos etapas:

- **Etapa de entrenamiento**: donde se realizará el entrenamiento del clasificador mediante validación cruzada con *kfold*. En esta etapa se dividirá el 90 % de datos objetivo en *k* subconjuntos, y se realizarán k iteraciones, de forma que cada subconjunto forme parte del entrenamiento y del test, como muestra la Figura 8.28. Esta etapa se repite para un barrido de los parámetros de cada clasificador, de forma que se obtiene el clasificador con mejor rendimiento. Por ejemplo, dado un conjunto con 100 datos objetivo y 10 datos *outlier*, el proceso de entrenamiento funcionará como sigue:

  1. Se separa el $90\,\%$ de los datos objetivo y *outlier*, de forma que quedan 90 datos objetivo y 9 datos *outlier*.
  2. Los datos objetivo se dividen en $k = 10$ subconjuntos, de forma que cada subconjunto contendrá 9 elementos.
  3. Para cada iteración, se utilizarán $k - 1$ subconjuntos de datos objetivo para el entrenamiento, lo que en este caso suma un total de 81 elementos.
  4. En cada iteración se utilizará el subconjunto restante de datos objetivo para el test, en este ejemplo serán 9 elementos, junto con los 9 datos *outlier* del principio.
  5. Se repetirán los pasos anteriores (a excepción del primero) k veces.

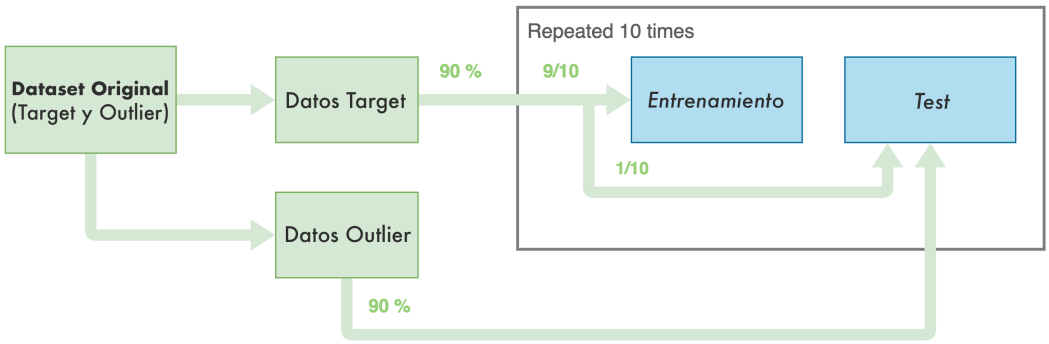

Figura 8.28 – Proceso de entrenamiento.

- **Etapa de validación**: en esta etapa se valida el clasificador con el que se ha obtenido el mejor rendimiento en la etapa de entrenamiento. En esta fase se vuelve a entrenar el clasificador, y se valida con el 10 % de datos objetivo y *outlier* que no se han usado para el entrenamiento, obteniendo así una buena representación del rendimiento del clasificador. Continuando el ejemplo de la anterior etapa, en esta los pasos a seguir serán:

  1. Con el mejor modelo del clasificador obtenido en la etapa anterior, se vuelve a entrenar utilizando el $90\,\%$ de los datos objetivo (90 elementos).
  2. Con los 10 datos objetivo y el dato *outlier* separado al principio, se procede a validar el nuevo clasificador.

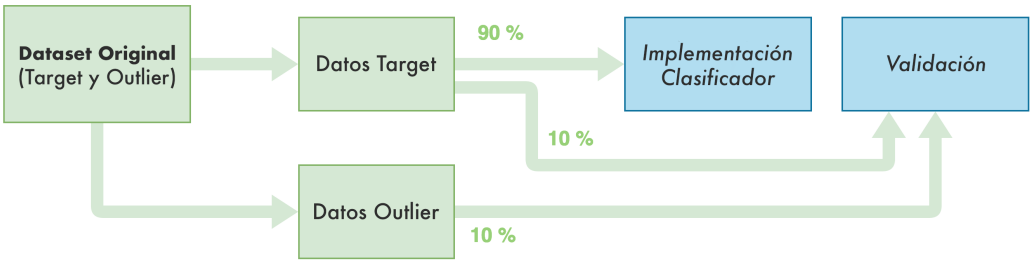

Figura 8.29 – Proceso de validación.

De esta forma se asegura que el clasificador tiene un rendimiento adecuado independientemente del conjunto de datos y se evita el *overfitting*.

8.4.2. Adaptación de los métodos

En este apartado se procederá a describir la adaptación llevada a cabo para cada método. Para los métodos que están englobados en la *toolbox PRTools* [20] se definirán únicamente sus hiperparámetros, ya que su adaptación se ha limitado a la fase de entrenamiento y validación de los mismos. Para el resto de métodos se detallarán los algoritmos implementados. A continuación se describirán los elementos comunes a todos los métodos. Todos ellos tendrán como punto de partida los siguientes parámetros de entrada:

- **El conjunto de datos** sobre el que se quiere construir el clasificador. Antes de la entrada al algoritmo de entrenamiento del clasificador, existe la posibilidad de normalizar los datos. Para ello se ha decidido utilizar dos métodos de normalización:

  - **Normalizado en el intervalo (0,1):** sea $x_i \in X \subset \mathbf{R}^n$ un punto del conjunto de datos de entrenamiento, para normalizarlo entre 0 y uno se realiza la siguiente operación:

$$x_{i_{norm}} = \frac{x_i - min\{x_i\}}{max\{x_i\} - min\{x_i\}} \cdot (0{,}99995 - 0{,}00005) + 0{,}00005 \tag{8.36}$$

- **Normalizado *z-score*:** mediante la función *zscore* de *Matlab*, se normaliza el conjunto de datos *X*, de forma que la media del conjunto normalizado sea nula y su desviación típica sea 1. De este modo, se realiza, para cada columna del conjunto de datos, y dado un elemento $x_{ij} \in X \subset R^n$, la siguiente operación:

$$x_{ij_{norm}} = \frac{x_{ij} - \mu_{ij}}{\sigma_{ij}} \tag{8.37}$$

  donde μ_{ij} y σ_{ij} son, respectivamente, la media y la desviación típica de cada columna.

De este modo, el conjunto de datos puede entrar al algoritmo de entrenamiento del clasificador de tres formas:

1. Sin normalizar
2. Normalizado en el intervalo (0,1)
3. Normalizado con *z-score*

- **Fracreg**. Otro de los parámetros de entrada, común a todos los algoritmos (excepto *APE*), es la fracción de datos objetivo rechazados durante el entrenamiento (*Fracreg*), del cual se hará un barrido para los valores $[0 : 5 : 10]$ en porcentaje. Generalmente, cuanto mayor sea el valor de este parámetro, mayor será el ajuste del límite de decisión al conjunto de entrenamiento. Las Figuras 8.30 y 8.31 muestran, respectivamente, un ejemplo de la variación de este parámetro en los clasificadores *KNN* y Parzen. En azul se muestra el límite de decisión para $Fracreg = 20\,\%$ mientras que en rojo el valor es de $Fracreg = 1\,\%$.

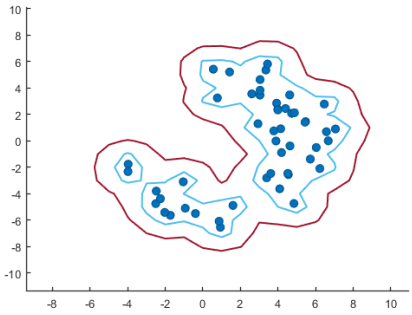

Figura 8.30 – Clasificador *KNN*

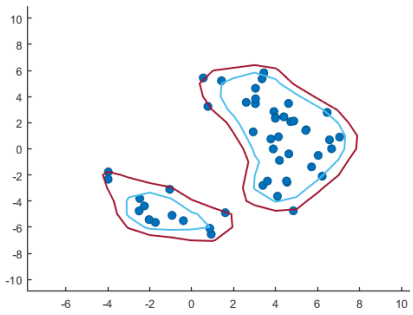

Figura 8.31 – Clasificador Parzen

- El valor ***k*** de las iteraciones para la validación *kfold*.

Por último, todos los métodos proporcionan a su salida los siguientes datos:

- **Resultados del entrenamiento**: valores de de la matriz de confusión, el *AUC*, la *precision*, el *accuracy*, el clasificador de cada combinación de los hiperparámetros, el tiempo de entrenamiento de cada uno y el tiempo de ejecución global para ese método.

- **Resultados de la validación**: el clasificador final y el valor del AUC.

A continuación se analizarán de forma pormenorizada los parámetros y algoritmos propios de cada método.

8.4.2.1. APE

Este método consta de dos algoritmos principales, uno de entrenamiento y otro de test. El algoritmo de entrenamiento tiene como entradas:

- El conjunto de datos
- τ matrices de proyección aleatorias

y como salida, el modelo del clasificador, que consta de:

- τ matrices de proyección aleatorias
- Los vértices del *CH*

El funcionamiento de este algoritmo consiste en la proyección del conjunto de datos y el cálculo del *CH* y de su centro, el cual se puede calcular según la media de todos los puntos, la media de los vértices del *CH* o la mediante la función *polygeom*. Finalmente se añaden a la estructura del modelo, además de estos datos, la matriz de proyección y los vértices proyectados.

El algoritmo de test funciona de la siguiente manera: dado un punto $z \in \mathbb{R}^n$, el modelo M y el parámetro α, el objetivo es averiguar si el punto pertenece al espacio generado por el *CH*. Para ello se procede a recorrer el conjunto de proyecciones calculadas anteriormente, comprobando en cada una si el punto está dentro. Si en una de ellas el punto queda fuera, se tiene que no pertenece al *CH* y por lo tanto se termina el bucle.

Para el uso del método implementado en Matlab se realizará primero un barrido de los siguientes parámetros:

- Tipo de centro: Media de todos los datos, media de los vértices del *CH* o el centroide.
- Tipo de extensión: *ECP* (Extended Convex Polytope) o *SCH* (Scaled *convex hull*).
- Parámetros de expansión alpha (para el *ECP*) y lambda (para el *SCH*).
- Método de normalización: Sin normalizar, normalizado entre 0 y 1 o zscore.
- Número de proyecciones.

La salida de la función de entrenamiento aporta los siguientes parámetros:

- AUC
- STD (Desviación típica)

- Ratios de falsos positivos y verdaderos positivos.
- Verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.
- Precisión y exactitud.

Se medirá el tiempo de entrenamiento para cada iteración del barrido paramétrico, así como el tiempo de ejecución global (incluyendo el entrenamiento y la validación). Estos tiempos se incluirán en las salidas de la función global, como se ha mencionado anteriormente.

En el Anexo 12 se muestra el código de la función principal de este método, en la cual se lleva a cabo el entrenamiento y la validación mediante los algoritmos descritos anteriormente. El resto de métodos descritos a continuación cuentan con una función equivalente, adaptada a sus respectivos hiperparámetros de entrada.

8.4.2.2. SVM

En este método se lleva a cabo la adaptación de las *Support Vector Machines* a la clasificación one-class. Para ello se implementa un algoritmo en el cual se realizan, dentro de cada *kfold*, los siguientes pasos:

1. Mediante la función *fitcsvm* de Matlab [2], se entrena un modelo de clasificador *SVM* sobre los datos de entrenamiento.
2. Se utiliza el modelo anterior para predecir las etiquetas de los datos de test, utilizando la función *predict* [1].
3. Se calculan los valores de las distintas métricas (AUC, precision, etc).

Los parámetros de entrada al algoritmo son:

- El conjunto de datos
- Fracción de datos objetivo rechazados (*Fracreg*). En este caso, la función *fitcsvm* se encarga de calcular el margen de error, tal que el número especificado de datos objetivo rechazados tenga una puntuación negativa.

8.4.2.3. Métodos de *PRTools*

En todos los algoritmos englobados en esta *toolbox*, el clasificador final se guardará con el propio formato proporcionado por ella. A continuación se describirán los hiperparámetros de cada uno de estos métodos.

- **KNN:** El umbral se fijará en función del método de distancias seleccionado y del parámetro *fracreg*, de forma que, ordenando las distancias de menor a mayor, se descarten el $(1 - Fracreg) \cdot 100\,\%$ de las distancias más pequeñas. Los parámetros de entrada son:
  - El conjunto de datos

- Fracción de datos objetivo rechazados (*Fracreg*)
- El número de vecinos (*k*)
- El método de distancia elegido para el cálculo del umbral:
  - *kappa*: calcula las distancias del k-vecino más próximo al resto de puntos del conjunto de entrenamiento
  - *delta*: calcula la media de la distancia a todos los k-vecinos
  - *gamma*: calcula la media cuadrática de las distancias de todos los k-vecinos

Si no se introduce el número de vecinos la propia *toolbox* se ocupa de optimizarlo.

- **K-Centers:** Los parámetros de entrada son:
  - El conjunto de datos
  - Fracción de datos objetivo rechazados (*Fracreg*)
  - El número de centros (*k*)
  - El número de repeticiones de selección de centros aleatorios.

En las Figuras 8.32, 8.33 y 8.34, se puede ver cómo varía el límite de decisión de un clasificador K-Centers al variar el número de centros *k*. Cuanto mayor es el número de centros, mayor es el ajuste del límite a los datos.

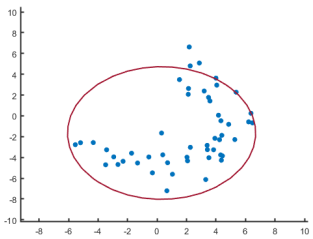

Figura 8.32 – Límite con $k = 1$

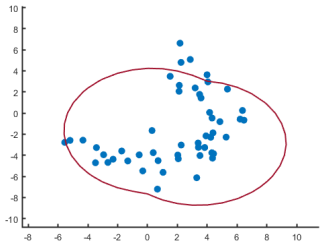

Figura 8.33 – Límite con $k = 2$

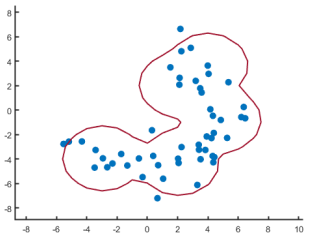

Figura 8.34 – Límite con $k = 5$

- **SVDD:** Los parámetros de entrada son:
  - El conjunto de datos
  - Fracción de datos objetivo rechazados (*Fracreg*)
  - El parámetro σ

En las Figuras 8.35, 8.36 y 8.37 se muestra la influencia del parámetro σ en el límite de decisión del clasificador.

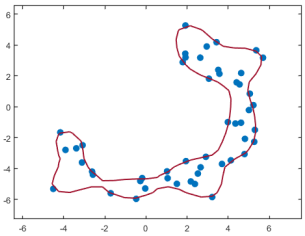

Figura 8.35 – $\sigma = 2$

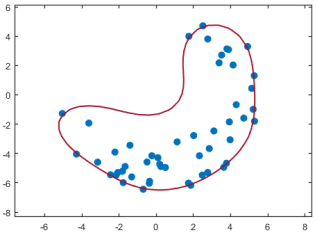

Figura 8.36 – $\sigma = 5$

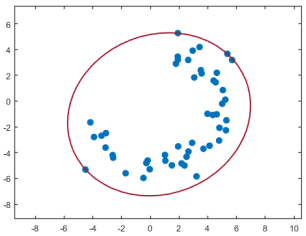

Figura 8.37 – $\sigma = 25$

- **MST:** Los parámetros de entrada son:
  - El conjunto de datos
  - Fracción de datos objetivo rechazados (*Fracreg*)
- **Gauss:** Los parámetros de entrada son:
  - El conjunto de datos
  - Fracción de datos objetivo rechazados (*Fracreg*)
  - Parámetro de regularización (*R*)

  El parámetro de regularización introduce un pequeño valor en la diagonal de la matriz de covarianzas para evitar problemas en el cálculo de su inversa. Generalmente no tiene un efecto resaltable en el límite de decisión, por lo que es recomendable utilizar el valor por defecto de la *toolbox*.
- **Parzen:** Los parámetros de entrada son:
  - El conjunto de datos
  - Fracción de datos objetivo rechazados (*Fracreg*)
  - Ancho de ventana de Parzen

  Si el ancho de la ventana es desconocido no es necesario hacer barrido, ya que la propia *toolbox* se encarga de optimizarlo.
- **PCA:** Los parámetros de entrada son:
  - El conjunto de datos
  - Fracción de datos objetivo rechazados (*Fracreg*)
  - El número de componentes principales
- **K-Means:** El cálculo de los prototipos óptimos se lleva a cabo de manera automática por la *toolbox*, utilizando un algoritmo similar al de esperanza-maximización. Los parámetros de entrada son. Los parámetros de entrada son:
  - El conjunto de datos

- Fracción de datos objetivo rechazados (*Fracreg*)
- El número de clusters (*k*)

- **Autoencoders:** Los parámetros de entrada son:
  - El conjunto de datos
  - Fracción de datos objetivo rechazados (*Fracreg*)
  - Número de neuronas de la capa de cuello de botella h_{auto}. Este parámetro será barrido de forma automática con $1 \leq h_{auto} \leq dim - 1$, donde *dim* es el número de variables del conjunto de datos.

8.5. Validación de los algoritmos propuestos

En esta sección se verán los resultados de la validación de los algoritmos implementados en Matlab sobre cinco conjuntos de datos reales, obtenidos del repositorio UCI [3]. Estos conjuntos tendrán distinto número de muestras, de variables y de porcentaje de *outliers*, de forma que se realice una validación que ofrezca una visión general del funcionamiento de los algoritmos en distintos tipos de conjuntos de datos y aplicaciones. Para esto se han seleccionado los conjuntos de datos mostrados en la Tabla 8.2.

Nombre	Nº Muestras	Nº Variables	*Outliers* (%)
Iris	150	4	100 (66.6 %)
Ionosphere	351	33	126 (36.0 %)
Vowels	1456	12	50 (3.43 %)
Cardio	1831	21	176 (9.61 %)
Annthyroid	7200	6	534 (7.42 %)

Tabla 8.2 – Conjuntos de datos para validación

A continuación se presentan los resultados obtenidos para cada uno de los conjuntos de datos descritos anteriormente, de los que se proporcionarán los resultados del AUC de la curva ROC y el tiempo de ejecución total del entrenamiento y la validación.

8.5.1. Iris

El conjunto de datos *Iris*, de R.A. Fisher [10], es uno de los más utilizados a la hora de evaluar el funcionamiento de algoritmos de aprendizaje automático. Este conjunto consiste en datos que caracterizan a tres tipos diferentes de plantas de la familia Iris. Se etiquetarán como *outliers* los pertenecientes a la clase 'setosa', mientras que el resto se considerarán pertenecientes a la clase objetivo. La Tabla 8.3 muestra los resultados obtenidos para este conjunto de datos.

Iris		
Método	**AUC (%)**	**Tiempo ejecución (min)**
APE	100	0.1480
AUTOENCODER	100	0.1119
GAUSS	100	0.0088
K-CENTERS	100	1.4769
K-MEANS	100	0.1659
KNN	100	0.0610
MST	100	0.0122
PARZEN	80.00	0.0350
PCA	100	0.0600
SVDD	100	0.0436
SVM	100	0.0172

Tabla 8.3 – Resultados validación con conjunto de datos Iris

8.5.2. Ionosphere

El conjunto de datos *Ionosphere* contiene mediciones de los electrones libres en la ionosfera, cuyos valores son binarios (clase 'buena' o 'mala'), de forma que se consideran *ouliers* los pertenecientes a la clase 'mala'. Este es el conjunto de mayor dimensionalidad de entre los que se han utilizado para esta validación. Se puede ver que, debido a esto, el tiempo de ejecución en el autoencoder es bastante elevado en comparación al resto. La mejor clasificación se obtiene con el método APE. La Tabla 8.4 muestra los resultados obtenidos para este conjunto de datos.

Ionosphere		
Método	**AUC (%)**	**Tiempo ejecución (min)**
APE	100	0.2428
AUTOENCODER	89.02	87.93
GAUSS	82.95	0.0168
K-CENTERS	86.74	1.726
K-MEANS	84.47	0.1734
KNN	84.47	0.0689
MST	86.74	0.0280
PARZEN	61.36	0.1401
PCA	89.02	0.7144
SVDD	84.47	0.0566
SVM	86.36	0.0232

Tabla 8.4 – Resultados validación con conjunto de datos Ionosphere

8.5.3. Vowels

El conjunto de datos *Vowels* contiene datos que representan la pronunciación de dos vocales japonesas por parte de 9 personas. En este caso se toman como datos *outlier* la parte correspondiente al primer sujeto, mientras que el resto se consideran pertenecientes a la clase objetivo. La Tabla 8.5 muestra los resultados obtenidos para este conjunto de datos.

Vowels		
Método	**AUC (%)**	**Tiempo ejecución (min)**
APE	50.00	0.2973
AUTOENCODER	64.64	0.9136
GAUSS	63.92	0.0126
K-CENTERS	66.42	3.666
K-MEANS	94.28	0.2191
KNN	93.92	0.2641
MST	97.14	0.1934
PARZEN	51.42	0.3885
PCA	95.35	0.2307
SVDD	84.28	1.998
SVM	65.35	0.0790

Tabla 8.5 – Resultados validación con conjunto de datos Vowels

8.5.4. Cardio

El conjunto de datos *Cardio* consiste en datos de medidas del pulso de un feto y de contracciones uterinas. En el conjunto original existen tres clases: patológica; normal y sospechosa, que se han adaptado de forma que la sospechosa se descarta, la patológica se considera como *outlier* y la normal como clase objetivo. La Tabla 8.6 muestra los resultados obtenidos para este conjunto de datos.

Cardio		
Método	**AUC (%)**	**Tiempo ejecución (min)**
APE	55.57	0.3323
AUTOENCODER	90.17	5.206
GAUSS	92.21	0.0096
K-CENTERS	96.36	3.644
K-MEANS	93.72	0.2277
KNN	93.93	0.3501
MST	92.51	0.3508
PARZEN	52.42	0.4266
PCA	90.39	0.4012
SVDD	89.09	7.051
SVM	94.33	0.0780

Tabla 8.6 – Resultados validación con conjunto de datos Cardio

8.5.5. Annthyroid

El conjunto *Annthyroid* representa datos de análisis de pacientes de los que se sospecha que puedan sufrir de hipotiroidismo. De esta forma, en el conjunto original consta de tres clases: *normal*, *hiperfunción* y *funcionamiento anormal*. En este caso, la clase normal se considerará como clase objetivo y los datos pertenecientes a las otras dos se considerarán ourliers. Esta es la clase que cuenta con un número de muestras más grande, lo que ha resultado en un tiempo de ejecución muy elevado para los algoritmos *K-Centers* y *SVDD*, que son muy sensibles al número de muestras. La Tabla 8.7 muestra los resultados obtenidos para este conjunto de datos.

Annthyroid		
Método	**AUC (%)**	**Tiempo ejecución (min)**
APE	54.71	2.505
AUTOENCODER	64.66	3.097
GAUSS	68.28	0.0531
K-CENTERS	68.29	364.3
K-MEANS	67.98	1.106
KNN	73.94	16.72
MST	72.46	6.085
PARZEN	72.09	18.68
PCA	87.08	0.2467
SVDD	66.75	700.6
SVM	78.74	2.146

Tabla 8.7 – Resultados validación con conjunto de datos Annthyroid

8.6. Conclusiones

Esta Sección expone las conclusiones a las que se ha llegado una vez realizado el trabajo. A lo largo del trabajo se ha podido observar que existen tres vías a la hora de abordar el problema de la clasificación one-class, que a su vez contienen multitud de métodos diferentes. La existencia de gran variedad de técnicas hace difícil para los usuarios finales la comprensión profunda de cada uno de los métodos. De esta problemática surge la necesidad de este trabajo, en el que se ha buscado realizar un análisis de las diferentes técnicas, desde una perspectiva en la que se supone que el lector no está necesariamente familiarizado con ellas, así como una simplificación en cuanto al uso, facilitando la adaptación de todos los algoritmos al entorno Matlab. De esta forma, se asegura que el usuario final tiene una comprensión básica de cómo funciona cada método, así como una herramienta, de uso sencillo, con la que aplicarlos a problemas reales.

Uno de los grandes obstáculos presentados a la hora de realizar este trabajo, ha sido la obtención de fuentes documentales, ya que resultaba imprescindible encontrar una base teórica profunda sobre la que fundamentar el análisis de cada técnica, y no el simple resumen que se suele encontrar en la mayoría de artículos.

En este trabajo, se ha podido comprobar también la presencia de la clasificación one-class en muchos ámbitos diferentes como son: la industria, la medicina, las ciencias naturales o las ciencias sociales [14, 5, 17, 23]. La capacidad de reconocer patrones anómalos en un conjunto de datos supone un avance muy grande en investigación dentro de estos campos.

En la Sección 8.5 se ha demostrado la utilidad de la implementación en Matlab de una amplia variedad de algoritmos, ya que, como se puede observar, dependiendo del conjunto de datos, se obtienen mejores resultados con alguno en específico, para lo cual es conveniente contar con la posibilidad de entrenarlos todos sobre un mismo conjunto, de forma que se pueda comparar su rendimiento y escoger el que mejor se adecúe.

Por último, y a título personal, considero que durante este trabajo he podido adquirir capacidades como la búsqueda de información; la capacidad de análisis; la programación optimizada y estructurada en Matlab o el tratamiento y representación de datos, las cuales me serán de utilidad en mi futuro profesional. Cabe mencionar también que, gracias al Departamento de Ingeniería Industrial, se me ha brindado la posibilidad de ser el autor, junto al Profesor Esteban Jove, de un artículo de investigación en el congreso internacional CISIS 2020 (Anexo 10), sobre nuevas técnicas de clasificación one-class, lo cual me ha permitido aprender sobre el funcionamiento de la investigación en el ámbito académico.

8.7. Trabajos futuros

En esta Sección se tratarán las posibles vías de desarrollo futuro en el contexto de este trabajo, así como aspectos que podrían complementarlo pero que se escapan del alcance del mismo. A continuación se detallará una lista de los trabajos futuros que se proponen:

- **Implementación de los algoritmos en el lenguaje Python.** Dado el amplio uso de este

lenguaje en el ámbito del *Machine Learning*, así como su característica de poseer una licencia de código abierto, resulta idóneo para esta aplicación.

- **Ampliación de los algoritmos analizados en el presente trabajo.** A pesar de haber cubierto los algoritmos principales de clasificación one-class, existen otros que no se han analizado, así como nuevas técnicas que merecen ser consideradas.
- **Programación de una interfaz GUI.** Con el fin de mejorar la experiencia de uso de la implementación de los algoritmos en Matlab.
- **Análisis estadístico.** En cuanto a los aspectos que podrían complementar este trabajo, sería interesante llevar a cabo un análisis estadístico [11], que permitiese comparar el rendimiento de los diferentes métodos de clasificación de una forma rigurosa.
- **Técnicas de *clustering*.** Puede resultar interesante la investigación sobre el uso de técnicas de *clustering* para agrupar datos, lo que a priori puede mejorar el rendimiento de alguno de los clasificadores. Esto se ha desarrollado en el artículo mencionado anteriormente para el caso del algoritmo *APE*. En la Sección 8.8 se expone de forma resumida el tema desarrollado en dicho artículo.

8.8. Artículo de investigación CISIS 2020

Durante la realización de este TFG he tenido la posibilidad de colaborar con el Departamento de Ingeniería Industrial en la realización de un artículo para el congreso internacional CISIS 2020.

La motivación de este artículo parte de un problema práctico real: el bajo rendimiento al entrenar un clasificador basado en métodos de contorno para una planta industrial que controla el nivel de agua de un depósito, la cual funciona en varios puntos de operación. El hecho de que existan distintos puntos de operación, hace que los datos se dividan en grupos. Si se aplica directamente un método de contorno a este conjunto de datos, se obtiene un clasificador cuyo borde de decisión contiene a todos los grupos, y a su vez, a todos los datos *outlier* que puedan existir entre ellos.

La solución propuesta en este artículo es el uso de técnicas de *clustering*, para posteriormente aplicar a cada uno de los grupos resultantes el método de clasificación APE. Como se puede ver en el Anexo 10, los resultados obtenidos han sido destacables, de forma que se consigue mejorar considerablemente el rendimiento de los clasificadores basados en técnicas de modelado de contorno, para este tipo de conjuntos de datos.

TÍTULO: **Análisis e implementación de algoritmos de técnicas de clasificación one-class**

ANEXOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2020

AUTOR: EL ALUMNO

Fdo.: IAGO NÚÑEZ LAMAS

Índice del documento ANEXOS

9 DOCUMENTACIÓN DE PARTIDA

Escuela Universitaria Politécnica Ferrol
UNIVERSIDADE DA CORUÑA

Intranet eup

ESCUELA UNIVERSITARIA POLITÉCNICA

ASIGNACIÓN DE TRABAJO FIN DE GRADO

En virtud de la solicitud efectuada por:
En virtud da solicitude efectuada por:

APELLIDOS, NOMBRE: *Núñez Lamas, Yago*
APELIDOS E NOME:

DNI: [redacted] **Fecha de Solicitud:** OCT2019
DNI: *Fecha de Solicitude:*

Alumno de esta escuela en la titulación de Grado en Ingeniería en Electrónica Industrial y Automática, se le comunica que la Comisión de Proyectos ha decidido asignarle el siguiente Trabajo Fin de Grado:

O alumno de esta escola na titulación de Grado en Enxeñería en Electrónica Industrial e Automática, comunícaselle que a Comisión de Proxectos ha decidido asignarlle o seguinte Traballo Fin de Grado:

Título T.F.G:Análisis e implementación de algoritmos de técnicas de clasificación one-class

Número TFG: *770G01A182*

TUTOR:(Titor) *Jove Pérez,Esteban*

COTUTOR/CODIRECTOR: *José Luis Calvo Rolle*

La descripción y objetivos del Trabajo son los que figuran en el reverso de este documento:

A descrición e obxectivos do proxecto son os que figuran no reverso deste documento.

Ferrol a Miercoles, 6 de Noviembre del 2019

DESCRIPCIÓN Y OBJETIVO:OBJETO:

El principal objeto de este trabajo es el de realizar un análisis pormenorizado de las técnicas de clasificación one-class más empleadas en la actualidad, así como la adaptación de sus algoritmos a Matlab, para su uso en posteriores aplicaciones. Se plantea la realización de un documento explicativo de cada una de ellas, en el que se oriente a potenciales usuarios sobre cómo emplear los algoritmos desarrollados.

ALCANCE:

Estudio general de los distintos tipos de clasificadores one-class.
Selección de las técnicas más empleadas.
Adaptación de dichas técnicas al entorno de programación Matlab.
Elaboración de un documento que refleje el principio de funcionamiento y los parámetros que influyen en cada una de las técnicas. Este documento debe servir de guía a futuros usuarios para el uso de los algoritmos desarrollados.
Validación del comportamiento de los algoritmos para detección de anomalías sobre datos reales.

10 PUBLICACIÓN EN CONGRESO INTERNACIONAL

Núñez I. et al. (2021) Hybrid Approximate Convex Hull One-Class Classifier for an Industrial Plant. In: Herrero Á., Cambra C., Urda D., Sedano J., Quintián H., Corchado E. (eds) 13th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2020). Advances in Intelligent Systems and Computing, vol 1267. Springer, Cham. https://doi.org/10.1007/978-3-030-57805-3_27

11 FUNCIÓN GLOBAL MATLAB

```
%%Función que realiza la carga y tratamiento de los datos, de forma que
%posteriormente se ejecuten los métodos de clasificación one-class
%Entrada:
%Archivo = Nombre del archivo del conjunto de datos OC.
%- Contiene la matriz de datos "DatosOneClass" y la de índices "Indices"
%
%Parámetros:
%-DatosNorm = Datos de clase objetivo para entrenamiento
%-DatosNormVal = Datos de clase objetivo para validación
%-DatosOut = Datos outlier para entrenamiento
%-DatosOutVal = Datos outlier para validación
%-Dat= cell con datos normalizados y separados para entrenamiento y
%validación
%-----------------------------
%Autor: Iago Núñez
%Universidade da Coruña, 2020

function global_occ(archivo)
%%Carga de datos
%Se carga el archivo y se separan los targets de los outliers
    load(archivo, 'DatosOneClass', 'Indices');

%%Normalización de los datos

%Normalización entre 0 y 1
Datos_zo = normalizar(DatosOneClass,'zo'); %Normalizacion de los datos target

%Normalización zscore
Datos_zscore = normalizar(DatosOneClass,'zscore'); %Normalizacion de los datos target


%%Se separan los datos entre entrenamiento y validación

%Se inicializa la cell donde se guardarán todos los datos
dat={};

%Se separan los datos sin normalizar, normalizados entre 0 y 1 y zscore
[dat.DNormVal{1,1}, dat.DOutVal{1,1}, dat.DNorm{1,1}, dat.DOut{1,1}] = separarDatos(DatosOneClass,Indices);
[dat.DNormVal{2,1}, dat.DOutVal{2,1}, dat.DNorm{2,1}, dat.DOut{2,1}] = separarDatos(Datos_zo,Indices);
[dat.DNormVal{3,1}, dat.DOutVal{3,1}, dat.DNorm{3,1}, dat.DOut{3,1}] = separarDatos(Datos_zscore,Indices);
dat.normType{1,1}="noNorm";
dat.normType{2,1}="zo";
dat.normType{3,1}="zscore";

%%Llamada a los métodos

resAPE = PRINCIPAL_APE(dat,archivo);
resAUTOENC = PRINCIPAL_AUTOENC(dat,archivo);
resGAUSS = PRINCIPAL_GAUSS(dat,archivo);
resKCENTERS = PRINCIPAL_KCENTERS(dat,archivo);
resKMEANS = PRINCIPAL_KMEANS(dat,archivo);
resKNN = PRINCIPAL_KNN(dat,archivo);
resPARZEN = PRINCIPAL_PARZEN(dat,archivo);
resPCA = PRINCIPAL_PCA(dat,archivo);
resSVDD = PRINCIPAL_SVDD(dat,archivo);
resMST = PRINCIPAL_MinSpanningTree(dat,archivo);
resSVM = PRINCIPAL_SVM(dat,archivo);


%%Tabla con resultados finales de la validación

Metodo=["APE","AUTOENC","GAUSS","KCENTERS","KMEANS","KNN","PARZEN","PCA","SVDD","MST","SVM"]';

AUC = [resAPE.validacion.AUC, resAUTOENC.validacion.AUC, resGAUSS.validacion.AUC...
    resKCENTERS.validacion.AUC, resKMEANS.validacion.AUC, resKNN.validacion.AUC, resPARZEN.validacion.AUC...
    resPCA.validacion.AUC, resSVDD.validacion.AUC, resMST.validacion.AUC, resSVM.validacion.AUC]';
Tiempo_entrenamiento =[resAPE.totalTime, resAUTOENC.totalTime, resGAUSS.totalTime ...
    resKCENTERS.totalTime, resKMEANS.totalTime, resKNN.totalTime, resPARZEN.totalTime ...
    resPCA.totalTime, resSVDD.totalTime, resMST.totalTime, resSVM.totalTime]';
```

```
T = table(Metodo,AUC,Tiempo_entrenamiento) %#ok<NOPRT>

cd ResultadosFinales
nombre=strcat("Resultados_Validacion_",archivo);
save(nombre,'T');
cd ..

end
```

12 FUNCIÓN PRINCIPAL DEL MÉTODO APE

```
%% Función principal del método APE

% Entradas:
%-dat: cell con los datos OC separados y con sus distintas normalizaciones
%-nombreArchivo: nombre del conjunto de datos utilizado
% Salida:
%-Resultados: cell con los resultados de entrenamiento y validación
% ------------------------------
% Autor: Iago Núñez
% Universidade da Coruña, 2020

function [Resultados] = PRINCIPAL_APE(dat,nombreArchivo)
%% Argumentos del entrenamiento

%kfold
NKFOLD=10;
REP=1;
%Parámetro de expansion
alpha=1;
%Dimensiones del espacio de proyeccion
dimensions=2;
%Metodo
method = 1;
%Tipo de centro
Center_type=3;
n_Center_type=size(Center_type,2);
%Tipo de extension
Extension_type=1;
n_Extension_type=size(Extension_type,2);
%numero de proyecciones
NPros=[5 10 50 100 500];
n_NPros=size(NPros,2);
%Valor de Lambda
Lambda=[0.8 1 1.2 1.4 1.6 1.8 2];
n_Lambda=size(Lambda,2);
%Tipo de Normalizacion [1,2,3]=noNorm, (0,1), zscore
Normalizacion=1;
n_Normalizacion=size(Normalizacion,2);

%Num=numero de iteracciones
Num=n_Center_type*n_Extension_type*n_NPros*n_Lambda*n_Normalizacion;
%% Generamos N_pros proyecciones aleatorias.
Projections = randn(size(dat.DNorm{1},1), 2, max(NPros));
%% %Proyecciones es una matriz de m*2*nPros, siendo m numero de
%% %variables de entrada
j=1;
%% Comenzamos las iteracciones
startTrain = tic;
for center_type=Center_type
    for extension_type=Extension_type
        for lambda=Lambda
            for NORMALIZACION=Normalizacion


                for nPros=NPros
                    startIteracion = tic;
                    [auc,desv,t_train,t_latencia, tpRatio,fpRatio, trueP, falseP, trueN, falseN,...
                        precission,accuracy] = APE_func(cell2mat(dat.DNorm(NORMALIZACION)), ...
                        cell2mat(dat.DOut(NORMALIZACION)), alpha, nPros,center_type, ...
                        extension_type, lambda, dimensions, NKFOLD, REP,Projections);
```

```
61
62                          Resultados.alpha{j,1}=alpha;
63                          Resultados.lambda{j,1}=lambda;
64                          Resultados.npros{j,1}=nPros;
65                          Resultados.method{j,1}='Convexo';
66
67
68                          if center_type==1
69                              Resultados.center_type{j,1}='Medio';
70                          elseif center_type==2
71                              Resultados.center_type{j,1}='Media_Vertices';
72                          else
73                              Resultados.center_type{j,1}='Centroid';
74                          end
75
76                          if NORMALIZACION==1
77                             Resultados.normType{j,1}='noNorm';
78                          elseif NORMALIZACION==2
79                              Resultados.normType{j,1}='zo';
80                          else
81                              Resultados.normType{j,1}='zscore';
82                          end
83
84                          Resultados.auc{j,1}=auc;
85                          Resultados.desv{j,1}=desv;
86                          Resultados.t_train{j,1}=t_train;
87                          Resultados.t_latencia{j,1}=t_latencia;
88                          Resultados.tpRatio{j,1}=tpRatio;
89                          Resultados.fpRatio{j,1}=fpRatio;
90                          Resultados.tp{j,1}=trueP;
91                          Resultados.fp{j,1}=falseP;
92                          Resultados.tn{j,1}=trueN;
93                          Resultados.fn{j,1}=falseN;
94                          Resultados.precission{j,1} = precission;
95                          Resultados.accuracy{j,1} = accuracy;
96                          finIteracion = toc(startIteracion);
97                          fprintf('APE %s: Paso #%i de %i completado. Tiempo consumido: %f minutos. Hora: %s \n',nombreArchivo , j, Num, finIteracion
98                          j=j+1;
99                  end
100             end
101         end
102     end
103 end
104 finTrain = toc(startTrain);
105 Resultados.totalTime = finTrain/60;
106 %%Guardar los parametros del mejor modelo para validar despues
107
108     %Primero se busca el indice del mejor AUC
109         [~,indAUC] = max(cell2mat(Resultados.auc));
110         ValData = {};
111     %Se guardan en una estructura todos los parametros del mejor modelo y
112     %tambien los datos tanto de validacion como de entrenamiento y test
113         ValData.alpha = Resultados.alpha(indAUC);
114         ValData.lambda = Resultados.lambda(indAUC);
115         ValData.npros = Resultados.npros(indAUC);
116         ValData.method = 'Convexo';
117         ValData.center_type = Resultados.center_type(indAUC);
118         ValData.extension_type = Extension_type;
119         ValData.normType = Resultados.normType(indAUC);
120
121 %%validacion del modelo obtenido
122
123     Resultados.validacion = validacionAPE(ValData, dat, method, dimensions);
124
125 %%Guardamos los resultados
126
127     ArchivoResultados = strcat('ResultadosAPE',num2str(nombreArchivo));
128     cd Resultados
129     save(ArchivoResultados,'Resultados')
130     cd ..
131
132 return
```

TÍTULO: **Análisis e implementación de algoritmos de técnicas de clasificación one-class**

PLANOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2020

AUTOR: EL ALUMNO

Fdo.: IAGO NÚÑEZ LAMAS

12.1. Planos

Esta sección no aplica debido al carácter teórico del trabajo.

TÍTULO: **Análisis e implementación de algoritmos de técnicas de clasificación one-class**

PLIEGO DE CONDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2020

AUTOR: EL ALUMNO

Fdo.: IAGO NÚÑEZ LAMAS

Índice del documento PLIEGO DE CONDICIONES

13 Pliego de condiciones

Esta sección no aplica debido al carácter teórico del trabajo.

TÍTULO: **Análisis e implementación de algoritmos de técnicas de clasificación one-class**

MEDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2020

AUTOR: EL ALUMNO

Fdo.: IAGO NÚÑEZ LAMAS

Índice del documento MEDICIONES

14 Mediciones

Esta sección no aplica debido al carácter teórico del trabajo.

TÍTULO: **Análisis e implementación de algoritmos de técnicas de clasificación one-class**

PRESUPUESTO

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2020

AUTOR: EL ALUMNO

Fdo.: IAGO NÚÑEZ LAMAS

Índice del documento PRESUPUESTO

15 PRESUPUESTO

Dado el carácter teórico del trabajo, el presupuesto se limita a la mano de obra y el precio del software utilizado. De esta forma el presupuesto se muestra en la Tabla 15.1.

Elemento	**Horas**	**Precio**
Mano de obra	550	30 €/h
Licencia Matlab	-	800 €/año
Total		17.300 €

Tabla 15.1 – Tabla de presupuesto