VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

MODELOS DE MACHINE LEARNING PARA LA CIENCIA DE DATOS

MACHINE LEARNING MODELS FOR DATA SCIENCE

Autor

Juan Luis Fernández Genaro

Tutor

José Luis Vicente Villardón

Grado

Estadística

Facultad

Ciencias

Departamento

Estadística

Salamanca, 2023

VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

MODELOS DE MACHINE LEARNING PARA LA CIENCIA DE DATOS

MACHINE LEARNING MODELS FOR DATA SCIENCE

Firma del alumno

Firmado por Juan Luis Fernández Genaro el día 03/07/2023 con un certificado emitido por el Ministerio de Hacienda y Función Pública.

Firma del tutor

VICENTE VILLARDON JOSE LUIS - 07839395Y

Firmado digitalmente por VICENTE VILLARDON JOSE LUIS - 07839395Y
Fecha: 2023.07.03 20:02:51 +02'00'

Juan Luis Fernández Genaro

José Luis Vicente Villardón

Grado

Estadística

Facultad

Ciencias

Departamento

Estadística

Salamanca, 2023

Certificado de los tutores TFG Grado en Estadística

D. José Luis Vicente Villardón, profesor/a del Departamento de Estadística de la Universidad de Salamanca.

HACE CONSTAR:

Que el trabajo titulado "*Modelos de Machine Learning para la Ciencia de Datos*", que se presenta, ha sido realizado por D. Juan Luis Fernández Genaro, con DNI 39490174W, y que constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Estadística en esta Universidad.

Salamanca, a fecha de firma electrónica.

Fdo.: Juan Luis Fernández Genaro

Firmado por Juan Luis Fernández Genaro el día 03/07/2023 con un certificado emitido por el Ministerio de Hacienda y Función Pública.

Fdo.: José Luis Vicente Villardón

VICENTE VILLARDON JOSE LUIS - 07839395Y

Firmado digitalmente por VICENTE VILLARDON JOSE LUIS - 07839395Y
Fecha: 2023.07.03 20:03:19 +02'00'

Agradecimientos

A mis padres, Juan Enrique y Marina, por apoyarme en todos los sentidos a lo largo de estos años, y por acometer todos los esfuerzos necesarios para permitirme disfrutar de la experiencia universitaria al completo.

A mis abuelos, los que están aún conmigo y los que no, porque sé el orgullo que les provoca verme conseguir mis objetivos.

A todos los compañeros que hoy en día llamo amigos, por acompañarme durante estos años y ser una parte fundamental de mi vida.

A Salamanca, por terminar de formarme como persona y permitirme ser la mejor versión posible de mí mismo.

Resumen

En los últimos años hemos visto como la ciencia de datos ha adquirido una importancia cada vez mayor, debido principalmente a su papel fundamental en la extracción de conocimiento y toma de decisiones en las grandes organizaciones del mundo. Debido al aumento de la cantidad y complejidad de estos datos, la aplicación de técnicas novedosas de Machine Learning se ha propuesto como una de las soluciones más útiles para abordar estos problemas. En este trabajo, hacemos un análisis teórico exhaustivo de algunas de estas técnicas, y comprobamos posteriormente su eficacia aplicándolas en una base de datos real, donde analizamos el rendimiento obtenido en una serie de cuestiones relativas al campo del aprendizaje supervisado (problema de clasificación multiclase) y no supervisado (técnicas de clustering y reducción de la dimensionalidad).

***Palabras clave**: Aprendizaje Automático, Modelo, Algoritmo, Aprendizaje Supervisado, Aprendizaje No Supervisado, Clasificación, Regresión. Árboles de Decisión, SVM, Naive-Bayes, Clustering, K-Medias, PCA, Análisis Factorial, SVD.*

Abstract

In recent years we have seen how data science has become increasingly important, mainly due to its fundamental role in knowledge extraction and decision making in large organizations around the world. Due to the increasing amount and complexity of this data, the application of novel Machine Learning techniques has been proposed as one of the most useful solutions to address these problems. In this paper, we make an exhaustive theoretical analysis of some of these techniques, and then test their effectiveness by applying them on a real database, where we analyze the performance obtained in a series of tasks related to the field of supervised learning (multiclass classification problem) and unsupervised learning (clustering and dimensionality reduction techniques).

***Keywords:** Machine Learning, Model, Algorithm, Supervised Learning, Unsupervised Learning, Classification, Regression, Decision Trees, SVM, Naive-Bayes, Clustering, K-Means, PCA, Factorial Analysis, SVD.*

Índice

Índice de Figuras

1. Introducción

Contexto

La ciencia de datos es un conjunto de principios fundamentales que sustentan y guían la extracción de información y conocimiento a partir de los datos (Fawcett & Provost, 2013)

En la actualidad, la Ciencia de Datos ha adquirido una gran relevancia debido a la enorme cantidad de información que se genera cada día en diferentes ámbitos. El análisis de datos se ha convertido en una herramienta fundamental para la toma de decisiones en empresas, instituciones y organizaciones en general

Con el aumento exponencial de los datos disponibles, la necesidad de procesar y analizarlos ha llevado a la aplicación de técnicas de Machine Learning, lo que ha permitido a los investigadores y profesionales de la Ciencia de Datos enfrentar problemas cada vez más complejos y obtener resultados más precisos.

La cantidad de información disponible para analizar no deja de crecer, y se espera que esta tendencia siga creciendo a lo largo de los próximos años, por lo que aplicar técnicas novedosas se ha vuelto más relevante que nunca. (Moreno, 2019)

En este contexto donde la información es cada vez más, y de mayor dificultad para ser gestionada, aparece el Machine Learning como una de las soluciones a este problema.

Haciendo uso de sus propiedades y del software informático disponible, podemos por lo tanto simplificar las tareas existentes, y encontrar soluciones de manera mucho más eficiente y veloz.

Objetivos

El objetivo principal de este trabajo es explorar las principales técnicas de Machine Learning aplicadas a la Ciencia de Datos. Para ello, se analizarán detalladamente los fundamentos teóricos de cada una de ellas, y su rango de aplicabilidad para cada tipo de problema existente.

Intentaremos discernir en qué situación es más o menos recomendable usar una u otra, las ventajas y desventajas que conlleva cada una de ellas, y por supuesto discutir sus limitaciones y perspectivas de cara al futuro.

Posteriormente, aplicaremos dichos algoritmos a una base de datos real, y analizaremos sus resultados.

Estructura del trabajo

Comenzaremos analizando el estado del arte actual del Machine Learning. Describiremos su historia, personajes más relevantes y orígenes. De seguido, analizaremos los diferentes tipos de aprendizaje automático existentes, y describiremos en detalle cada uno de sus algoritmos principales.

Continuaremos haciendo uso de varios de dichos algoritmos en una base de datos real, donde podremos testear y evaluar de primera mano los resultados de cada uno de ellos, y discutir su conveniencia e idoneidad en cada uno de los casos.

Finalmente, realizaremos unas conclusiones donde sintetizaremos y valoraremos los resultados obtenidos.

2. Estado del Arte

Qué es el Machine Learning: Definiciones y conceptos relevantes

El aprendizaje automático (Machine Learning) es el campo de estudio que confiere a los ordenadores la capacidad de aprender sin ser programados explícitamente. (Samuel, 1959)

Puede parecer curioso que una definición tan antigua siga siendo tan relevante hoy en día, pero así es. El concepto principal se mantiene invariable.

Necesitamos de todas formas, definir varios conceptos relevantes a esta disciplina, comencemos explicando qué es aprender en este contexto de la computación.

Se dice que un programa informático aprende de la experiencia E con respecto a alguna tarea T y alguna medida de rendimiento P si su rendimiento en T, medido por P, mejora con la experiencia E. (Mitchell, 1997)

En los sistemas informáticos, la experiencia existe en forma de datos, y la principal tarea del Machine Learning es desarrollar algoritmos de aprendizaje que construyen modelos a partir de dichos datos. Alimentando el algoritmo de aprendizaje con datos de experiencia, obtenemos un modelo que puede hacer predicciones sobre nuevas observaciones.

Un algoritmo de aprendizaje automático es una fórmula matemática utilizada para procesar datos y generar un modelo predictivo que permita a un sistema informático aprender de manera autónoma a partir de dichos datos. (Mitchell, 1997)

Por lo tanto, si consideramos la informática como la materia de los algoritmos, el Machine Learning es la materia de los algoritmos de aprendizaje.

Es importante definir también el concepto de modelo, el cual será usado repetidas veces a lo largo de nuestro trabajo.

Un modelo se define como una función matemática que se utiliza para predecir el valor de una variable de salida a partir de una o más variables de entrada. (Hastie, 2009)

El conjunto de datos existentes del cual haremos uso para entrenar, validar y probar nuestro modelo es llamado dataset. Más formalmente, un dataset, también conocido como conjunto de datos, es una colección organizada de datos que se utilizan para analizar y obtener información sobre un tema en particular. Este conjunto de datos puede ser de diferentes tipos y formatos, como texto, imágenes, videos, sonidos, etc.

Un dataset está a su vez formado por varios elementos. Matemáticamente:

Sea $D = \{x_1, x_2 \dots x_m\}$ un dataset que contiene m instancias, donde cada instancia está a su vez formada por d atributos, cada instancia $x_i = (x_{i1}; x_{i2} \dots ; x_{id}) \in \chi$ es un vector en el espacio muestral d-dimensional χ, donde d se denomina a la dimensionalidad de la instancia x_i y x_{ij} es el valor del atributo j-ésimo de la instancia x_i.

El proceso de utilizar algoritmos de aprendizaje automático para construir modelos a partir de datos es lo que denomina aprendizaje o entrenamiento. Es el proceso de ajustar los parámetros de un modelo predictivo para que se ajuste a los datos de entrenamiento.

Podemos entrenar nuestro modelo en distintos campos, no estamos limitados a hacerlo en un ámbito concreto. La implicación principal de que los datasets puedan contener distintos tipos de datos es que podremos hacer uso de técnicas de aprendizaje automático en múltiples formatos. Desde enfoques más clásicos aplicados desde hace varios años, como predicción de SPAM o precios bursátiles, hasta detección de patrones complejos en formatos que, intuitivamente, pueden parecer más difíciles o contraintuitivos, como sería el de las Imágenes, Audio o Texto.

Los datos utilizados en la fase de entrenamiento se denominan datos de entrenamiento, en la que cada muestra es un ejemplo de entrenamiento, y el conjunto de todos los ejemplos de entrenamiento se denomina conjunto de entrenamiento.

Dado que un modelo aprendido por una computadora corresponde a las reglas subyacentes existentes en los datos, también podemos denominarlo *hipótesis*, y las reglas subyacentes reales se denominan hechos o *verdad fundamental*.

Por tanto, el objetivo final del aprendizaje automático es encontrar o aproximarse a la verdad fundamental, a partir de unos datos existentes.

Una vez hayamos entrenado satisfactoriamente nuestro modelo, necesitaremos también validarlo.

La validación es el proceso de evaluar la precisión y eficacia de un modelo predictivo utilizando un conjunto de datos independiente que no se utilizó en el entrenamiento del modelo. Este proceso es fundamental para determinar si el modelo tiene un buen rendimiento y si puede generalizar su capacidad predictiva a nuevos datos. (Kohavi, 1995)

Para realizar dicho proceso de validación, a su vez, requeriremos de un dataset de validación.

Un dataset de validación es un conjunto de datos independiente utilizado para validar y evaluar la precisión y eficacia de un modelo predictivo. Este conjunto de datos se utiliza después del proceso de entrenamiento del modelo y se utiliza para determinar si el modelo es capaz de generalizar su capacidad predictiva a nuevos datos. (Larose, 2014)

Finalmente, tras haber entrenado y validado nuestro modelo de aprendizaje automático, solo nos quedará probar su eficacia.

La prueba es el proceso de evaluar el rendimiento y la precisión de un modelo predictivo utilizando un conjunto de datos completamente independiente que no se utilizó ni en el entrenamiento ni en la validación del modelo. Este proceso es fundamental para determinar si el modelo es capaz de generalizar su capacidad predictiva a nuevos datos y si puede ser desplegado de manera efectiva en el mundo real. (Géron, 2019)

Por último, cabe destacar un área particular del Machine Learning conocida como Deep Learning, de la cual hacen uso muchos de las herramientas de aprendizaje automático más utilizadas en la actualidad.

El aprendizaje profundo (deep learning) es una técnica de aprendizaje automático que se basa en la construcción de modelos de redes neuronales artificiales con múltiples capas de procesamiento. Estos modelos pueden aprender y representar información compleja a partir de grandes conjuntos de datos, y han demostrado ser altamente efectivos en tareas de clasificación, reconocimiento de patrones y predicción en diversos campos, como la visión por computadora, el procesamiento del lenguaje natural y la robótica. El aprendizaje profundo se basa en el ajuste iterativo de los pesos de las conexiones entre las neuronas de la red, utilizando algoritmos de optimización para minimizar la diferencia entre las predicciones del modelo y los valores reales del conjunto de datos. (LeCun, 2015))

En siguientes apartados de nuestro trabajo estudiaremos detenidamente los modelos de aprendizaje automático más comunes, y profundizaremos en varios conceptos de las definiciones previamente expuestas.

Historia del Machine Learning

Orígenes

Los inicios del aprendizaje automático se remontan a la década de 1950, cuando los científicos comenzaron a explorar la idea de que las máquinas podían aprender de manera similar a los humanos, a partir de la exposición a datos. Uno de los primeros trabajos en este campo fue el de Arthur Samuel, quien en 1959 desarrolló un programa de ajedrez que mejoraba su desempeño a medida que jugaba, utilizando técnicas de aprendizaje automático.

En esa misma época, otros científicos comenzaron a desarrollar modelos de aprendizaje automático basados en redes neuronales, inspirados en la estructura del cerebro humano. Uno de los primeros trabajos en este campo fue el de Frank Rosenblatt, quien en 1958 propuso el modelo de Perceptrón, una red neuronal capaz de clasificar patrones en datos binarios.

Sin embargo, el desarrollo del aprendizaje automático se vio obstaculizado por la falta de acceso a grandes cantidades de datos y a la capacidad de procesamiento necesaria para manejarlos. Esto llevó a un período de relativa inactividad en el campo durante las décadas de 1960 y 1970.

Década 1980 – 1990

A partir de la década de 1980, el campo del Machine Learning experimentó un renacimiento gracias al avance en la capacidad de procesamiento y al surgimiento de nuevas técnicas y algoritmos de aprendizaje automático. Uno de los hitos más importantes de esta época fue el desarrollo del algoritmo de backpropagation, una técnica de entrenamiento de redes neuronales que permitió mejorar significativamente su desempeño en la clasificación de patrones complejos.

Además, surgieron nuevas técnicas de aprendizaje supervisado, como los árboles de decisión y las máquinas de vectores de soporte (SVM), que permitieron la clasificación y regresión de datos en diferentes áreas, como la medicina, las finanzas, la seguridad y la robótica.

Años 2000

En la década de 2000, el Machine Learning experimentó un nuevo auge gracias al surgimiento de grandes cantidades de datos generados por la proliferación de la tecnología digital y la explosión de Internet. El desarrollo de nuevas técnicas de minería de datos, como el clustering y la detección de anomalías, permitió a los científicos extraer información valiosa de estos datos y utilizarla para la toma de decisiones.

Además, surgieron nuevas técnicas de aprendizaje no supervisado, como el aprendizaje por refuerzo y el aprendizaje profundo, que permitieron a los modelos de Machine Learning aprender de manera autónoma a partir de la exposición a datos y a retroalimentación.

2010-2020

En la última década, el Machine Learning se ha convertido en una de las áreas más prometedoras y en constante crecimiento dentro del mundo de la tecnología. Las empresas y organizaciones de todo el mundo están utilizando técnicas de Machine Learning para resolver una amplia variedad de problemas, desde la detección de fraudes y el diagnóstico médico hasta la recomendación de productos y servicios personalizados.

Uno de los avances más importantes en el campo del Machine Learning en los últimos años ha sido el desarrollo de modelos de aprendizaje profundo, también conocidos como redes neuronales profundas. Estos modelos se basan en redes neuronales artificiales con múltiples

capas, lo que les permite aprender patrones más complejos en los datos y mejorar su desempeño en tareas de visión por computadora, procesamiento de lenguaje natural y reconocimiento de voz.

Además, el Machine Learning se ha expandido expandiendo rápidamente a nuevas áreas, como la robótica y la automoción, donde se están desarrollando sistemas autónomos capaces de aprender y tomar decisiones en tiempo real.

Actualidad y auge en popularidad

Hoy en día el Machine Learning es parte fundamental de la vida de muchas personas. En los últimos años el interés del público en sector no ha parado de crecer. Este fenómeno ha sucedido principalmente debido principalmente a la llegada al mercado de los Large Language Models (LLM), como GPT-3 (de OpenAI), LaMDA (de Google) o LLaMA (Meta AI), además de la popularización de herramientas como Dall-E, MidJourney o Whisper.

Un Large Language Model (LLM) es un modelo de aprendizaje automático que utiliza técnicas de procesamiento de lenguaje natural para aprender patrones en grandes conjuntos de datos de texto, con el objetivo de generar texto coherente y similar al producido por un humano.

Estos modelos utilizan algoritmos de redes neuronales y aprendizaje profundo para procesar y comprender el lenguaje natural en su contexto, y pueden generar texto en respuesta a un "prompt" proporcionado por el usuario. Estos modelos contienen miles de millones de parámetros y son capaces de generar texto de alta calidad en una variedad de contextos, desde chatbots hasta generación de contenido y análisis de sentimiento, y son una herramienta poderosa para procesar grandes cantidades de datos de texto y generar información útil a partir de ellos.

Los Large Language Models están comenzando a tener una implicación significativa en la sociedad. El más famoso de ellos, ChatGPT, ha batido cifras de usuarios en tiempo récord.

Pero como mencionamos previamente, los LLM no son las únicas herramientas que han captado la atención del público masivo. En el último año softwares como DALL-E o Midjourney o Whisper también han obtenido una gran cantidad de usuarios en sus servicios.

DALL-E, este es un software desarrollado por OpenAI que utiliza una combinación de técnicas de procesamiento de lenguaje natural y de visión por computadora para generar imágenes a partir de descripciones de texto. Ha conseguido ser capaz de crear imágenes altamente detalladas y realistas de objetos y situaciones que nunca antes han sido fotografiados.

Midjourney comparte varias características con DALL-E, pero está diseñado específicamente para producir arte y se centra en la generación de contenido emocionalmente resonante.

Ambos softwares han sido utilizados en una gran variedad de aplicaciones, incluyendo entre ellas la publicidad y el diseño gráfico, y su impacto en diversas industrias no dejará de crecer en los próximos años. Whisper en cambio es un sistema de reconocimiento automático de voz (ASR), que es capaz de convertir a texto todo lo que escucha, y además traducirlo simultáneamente al lenguaje requerido en tiempo real.

Como vemos, las aplicaciones del Machine Learning son infinitas, y es seguro que en los próximos años veremos una avalancha de nuevas funcionalidades que usan dichas tecnologías.

3. Algoritmos de Machine Learning

Uno de los aspectos más importantes del Machine Learning es la selección del algoritmo adecuado para el problema en cuestión que tratamos de resolver.

Existen diferentes tipos de algoritmos de Machine Learning, cada uno con sus propias características y casos de uso específicos, por lo que es importante discernir cual nos conviene más dependiendo de nuestras necesidades concretas.

En este apartado de nuestro trabajo se presentarán los principales algoritmos utilizados hoy en día, con el objetivo de ofrecer una visión general de los diferentes enfoques utilizados en el campo.

Englobaremos los algoritmos principalmente en 4 tipos principales; aprendizaje supervisado, aprendizaje no supervisado, aprendizaje por refuerzo, y aprendizaje semi-supervisado. En este trabajo, de todas formas, nos centraremos en los 2 primeros tipos, siendo los 2 últimos más propios de técnicas más complejas que involucran Deep Learning.

En cada uno de ellos definiremos sus características principales, sus casos de uso más habituales, y qué tipo de problemas tratan de resolver.

Finalmente, discutiremos las ventajas y desventajas que conllevan.

Tipología de problemas más habituales

Previamente a explicar dichos algoritmos, es necesario comprender que tipos de problemas tratamos de solventar habitualmente en el campo del machine learning, para entender de forma satisfactoria cuando es recomendable usar uno u otro.

La mayoría de los problemas que dichos modelos de aprendizaje automático tratan de solventar pueden a su vez englobarse en 2 tipos claramente diferenciados; Clasificación y Regresión.

En un problema de clasificación, el objetivo es predecir una variable de respuesta discreta, como sería, por ejemplo, la clasificación de un correo electrónico como spam o no spam.

En cambio, en un problema de regresión, el objetivo es predecir una variable de respuesta continua, como podría ser la predicción del precio de una casa en función de su ubicación y características particulares.

Mas formalmente:

La tarea de la clasificación es predecir una variable discreta, conocida como variable de respuesta o etiqueta de clase, a partir de un conjunto de variables de entrada o predictoras Por otro lado, en la regresión, el objetivo es predecir una variable de respuesta continua, como una variable de salida real. (Hastie, 2009)

Si el resultado de la predicción tiene sólo dos clases posibles, se denomina problema de clasificación binaria, en el que una clase se marca como positiva y la otra como negativa. Cuando hay más de dos clases, se convierte en un problema de clasificación múltiple.

Otros problemas generalmente tratados en el campo del Machine Learning son el agrupamiento (Clustering), la reducción de dimensiones o el procesamiento del lenguaje natural (NLP, por sus siglas en inglés, muy de moda actualmente gracias a los LLM previamente mencionados). Procedamos a definirlos.

El clustering, o agrupamiento, es una técnica de análisis de datos no supervisada que busca agrupar observaciones similares en clústeres o grupos. En el clustering, no se parte de una variable objetivo previamente definida, sino que se busca descubrir estructuras y patrones ocultos en los datos a partir de la similitud entre las observaciones. (Jain, 1999)

Los métodos de clustering pueden ser jerárquicos o no jerárquicos, y pueden utilizarse para descubrir relaciones entre variables, identificar grupos de consumidores o pacientes, o detectar anomalías en los datos, entre otras aplicaciones.

El Procesamiento de Lenguaje Natural (NLP, por sus siglas en inglés) es un campo de la informática y la inteligencia artificial que se enfoca en la interacción entre los seres humanos y los sistemas de computación a través del lenguaje natural. El NLP se encarga de desarrollar modelos y técnicas que permiten a las máquinas comprender, interpretar, y generar lenguaje humano (Manning, 1999)

Las aplicaciones del NLP son diversas e incluyen desde la traducción automática, el análisis de sentimiento, la generación de texto, hasta la extracción de información y la respuesta automática a preguntas. El NLP también se ocupa de la resolución de ambigüedades lingüísticas, el análisis gramatical, la identificación de entidades nombradas, y la clasificación de textos según su contenido.

En el caso de este trabajo, a la hora de exponer un caso práctico de uso de un algoritmo de aprendizaje automático, nos centraremos principalmente en resolver alguno de los 4 primeros tipos de problemas (Clasificación, Regresión, Reducción de dimensiones o Clustering), debido a que el Natural Language Processing (NLP) es una disciplina extremadamente compleja y que requiere de enormes cantidades de datos para su correcta optimización y utilización (es decir, hace uso de técnicas de Deep Learning).

Aprendizaje Supervisado

El aprendizaje supervisado es una técnica fundamental en el campo del aprendizaje automático que se utiliza para predecir una variable de salida (objetivo) a partir de un conjunto de variables de entrada y un conjunto de datos de entrenamiento previamente etiquetados.

En el aprendizaje supervisado, el algoritmo tiene como objetivo aprender a predecir la etiqueta correcta para nuevas instancias no vistas previamente.

Mas formalmente:

El aprendizaje supervisado es un tipo de aprendizaje automático en el que el objetivo es aprender una función de mapeo de entrada-salida a partir de ejemplos de entrenamiento previamente etiquetados. En este proceso, se proporciona al algoritmo una entrada y una salida correspondiente, y el algoritmo aprende a producir la salida correcta para nuevas entradas. (Mitchell, 1997)

El aprendizaje supervisado se ha utilizado con éxito en una amplia gama de aplicaciones, entre otras:

- Reconocimiento de voz: donde se utiliza para convertir las señales de voz en texto.
- Reconocimiento de imágenes: donde se utiliza para identificar objetos y personas en imágenes.
- Clasificación de texto: donde se utiliza para etiquetar automáticamente textos según su tema o contenido.
- Detección de fraude: donde se utiliza para identificar transacciones fraudulentas en tarjetas de crédito o sistemas financieros.
- Diagnóstico médico: donde se utiliza para clasificar imágenes médicas según la presencia de enfermedades.

Usualmente, los problemas que tratan de resolver los algoritmos de aprendizaje supervisado son de *clasificación* y *regresión.*

Entre los principales algoritmos que utilizan técnicas de aprendizaje supervisado encontramos:

Regresión mínimos cuadrados

La regresión de mínimos cuadrados es un método utilizado en el aprendizaje automático para modelar la relación entre una variable dependiente y y una o más variables independientes $x_1, x_2, \dots, x_.$

El objetivo es encontrar la mejor función lineal que minimice la suma de los cuadrados de las diferencias entre los valores reales de y y los valores predichos por el modelo. Esta función lineal se conoce como la línea de regresión y se representa por la ecuación:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n +$$

Donde:

- y es la variable dependiente que queremos predecir.

-$x_1, x_2, \dots, x_.$ son las variables independientes que utilizamos para hacer la predicción.

- $\beta_0, \beta_1, \beta_2, \dots, \beta_n$ son los coeficientes de regresión que representan las pendientes de las variables independientes.

- ϵ es el término de error, que representa la diferencia entre el valor real de y con el valor predicho por el modelo.

El objetivo de la regresión de mínimos cuadrados es encontrar los valores óptimos para los coeficientes de regresión que minimicen la suma de los cuadrados de los residuos (diferencias entre los valores reales y los valores predichos). Esto se puede lograr mediante el método de mínimos cuadrados, que busca minimizar la siguiente función de costo:

$$\text{Costo}(\beta_0, \beta_1, \beta_2, \dots, \beta_n) = \sum_{i=1}^{m} \left(y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_n x_{in})\right)^2$$

Donde:

- m es el número de observaciones en el conjunto de datos de entrenamiento.

- y_i son los valores reales de la variable dependiente en la i-ésima observación.

- $x_{i1}, x_{i2}, \dots, x_{in}$ son los valores de las variables independientes en la i-ésima observación.

La regresión de mínimos cuadrados encuentra los valores óptimos de los coeficientes de regresión al minimizar el costo mediante técnicas de optimización, como el método de descenso de gradiente o la solución analítica utilizando álgebra lineal.

Una vez que se obtienen los coeficientes óptimos, se puede utilizar el modelo de regresión de mínimos cuadrados para predecir el valor de y para nuevas observaciones utilizando la ecuación de la línea de regresión.

Regresión logística

La regresión logística es un algoritmo de aprendizaje supervisado utilizado en el campo del machine learning para modelar y predecir la probabilidad de que una variable dependiente binaria o categórica se encuentre en una clase determinada. A diferencia de la regresión de mínimos cuadrados, que se utiliza para variables dependientes continuas, la regresión logística es especialmente adecuada para problemas de clasificación.

La regresión logística utiliza la función logística, también conocida como función sigmoide, para estimar la probabilidad de pertenencia a una clase. La función sigmoide se define como:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Donde z es una combinación lineal de las variables independientes ponderadas por los coeficientes de regresión, similar a la regresión de mínimos cuadrados. La ecuación de la regresión logística se puede expresar como:

$$P(y = 1 | x_1, x_2, \dots, x_n) = \sigma(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)$$

Donde:

$$P(y = 1 | x_1, x_2, \dots, x_n)$$

representa la probabilidad de que la variable dependiente y sea igual a 1, dado los valores de las variables independientes $x_1, x_2, \dots, x_n$.

$$\beta_0, \beta_1, \beta_2, \ldots, \beta_n$$

Son los coeficientes de regresión que determinan la influencia de las variables independientes en la probabilidad de pertenencia a la clase 1.

El proceso de entrenamiento de la regresión logística implica encontrar los valores óptimos para los coeficientes de regresión que maximicen la verosimilitud de los datos observados. Una vez que se obtienen los coeficientes óptimos, se puede utilizar el modelo de regresión logística para hacer predicciones clasificando nuevas observaciones en función de la probabilidad estimada.

Árboles de decisión

Los árboles de decisión son algoritmos de aprendizaje automático que utilizan una estructura de árbol para tomar decisiones basadas en reglas lógicas. Son ampliamente utilizados en el campo del aprendizaje automático debido a su simplicidad y capacidad para manejar tanto problemas de clasificación como de regresión.

Un árbol de decisión consta de nodos que representan atributos o características, y arcos que representan las reglas de decisión basadas en esos atributos. El nodo raíz del árbol representa la característica más importante o el atributo que tiene el mayor poder de discriminación. Los nodos internos representan las pruebas en los atributos, y los nodos hoja representan las clases o los valores de salida.

Para tomar una decisión en un árbol de decisión, se sigue un camino desde el nodo raíz hasta un nodo hoja, siguiendo las reglas de decisión en cada nodo interno. Cada regla de decisión se basa en una prueba del valor del atributo correspondiente. Por ejemplo, en un problema de clasificación binaria, la prueba podría ser "si el atributo X es mayor que un umbral, ir al subárbol izquierdo, de lo contrario, ir al subárbol derecho".

El proceso de construcción de un árbol de decisión implica dividir el conjunto de datos de entrenamiento en subconjuntos más pequeños basados en los atributos, de modo que los subconjuntos resultantes sean lo más homogéneos posible en términos de la clase o el valor objetivo. Esta división se realiza de manera recursiva hasta que se cumplan ciertos criterios de detención, como alcanzar un nivel máximo de profundidad del árbol.

Los árboles de decisión tienen varias ventajas, como su interpretabilidad, la capacidad de manejar tanto datos numéricos como categóricos, y su capacidad para capturar relaciones no lineales entre las características y la variable objetivo. Sin embargo, también pueden ser propensos al sobreajuste si no se controla adecuadamente.

Una vez construido el árbol de decisión, se puede utilizar para hacer predicciones clasificando nuevas observaciones o estimando valores de salida. Cada observación sigue el camino correspondiente en el árbol, y la clase o el valor objetivo se determina según el nodo hoja alcanzado.

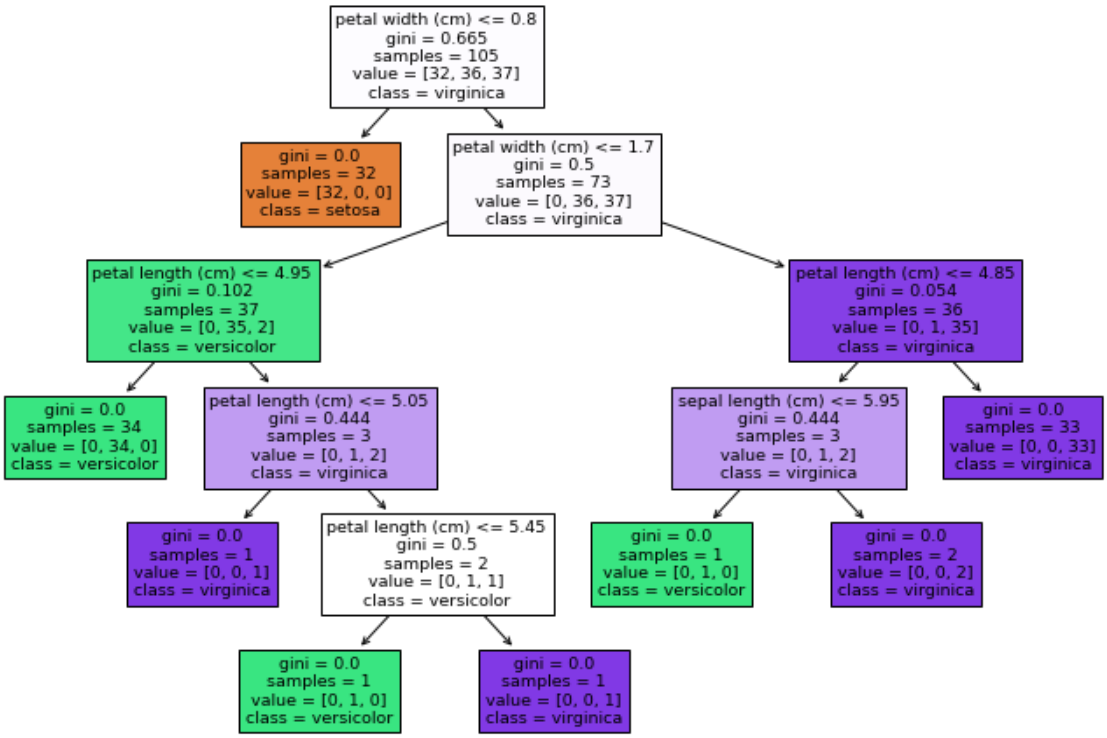

Figura 1: Árbol de decisión correspondiente al dataset Iris. Fuente: Elaboración propia.

Máquinas de Vector Soporte (SVM)

Las Máquinas de Vectores de Soporte (SVM) son un poderoso algoritmo de aprendizaje automático utilizado tanto para problemas de clasificación como de regresión. La idea fundamental detrás de las SVM es encontrar el hiperplano óptimo que mejor separe las muestras de diferentes clases o que mejor se ajuste a los datos en el caso de la regresión.

Dado un conjunto de datos de entrenamiento, donde cada muestra tiene un conjunto de características y una etiqueta de clase asociada, el objetivo de las SVM es encontrar el hiperplano de separación que maximice el margen entre las muestras de diferentes clases. El margen se define como la distancia perpendicular más cercana entre el hiperplano y las muestras de entrenamiento.

En el caso de problemas de clasificación linealmente separables, el hiperplano óptimo se puede representar por la ecuación:

$$w \cdot x + b = 0$$

Donde w es el vector de pesos normal al hiperplano, w es el vector de características de una muestra y b es el sesgo o término de sesgo. Las muestras se clasifican según qué lado del hiperplano se encuentren.

Sin embargo, en la mayoría de los casos, los conjuntos de datos no son linealmente separables. En esos casos, se utilizan herramientas matemáticas para mapear las muestras a un espacio dimensional superior donde puedan ser separables linealmente. Esto se logra mediante el uso

de funciones kernel, que calculan el producto escalar entre las muestras en el espacio de características ampliado sin necesidad de realizar la transformación explícita.

Al encontrar el hiperplano de separación óptimo, las SVM también pueden manejar eficientemente casos en los que existen muestras atípicas o ruido en los datos. Las muestras más cercanas al hiperplano, llamadas vectores de soporte, son las más importantes para determinar la ubicación y orientación del hiperplano, y son fundamentales para hacer predicciones precisas en nuevas muestras.

Las SVM se pueden extender para resolver problemas de clasificación no lineal mediante la utilización de kernel no lineales, como el kernel polinomial o el kernel gaussiano (RBF). Estos kernels permiten modelar relaciones más complejas entre las características y las etiquetas de clase.

Además de la clasificación, las SVM también se utilizan en problemas de regresión, donde el objetivo es encontrar una función que se ajuste a los datos con el menor error posible. La formulación de la regresión SVM se basa en encontrar un hiperplano que pase cerca de la mayoría de las muestras, en lugar de separarlas.

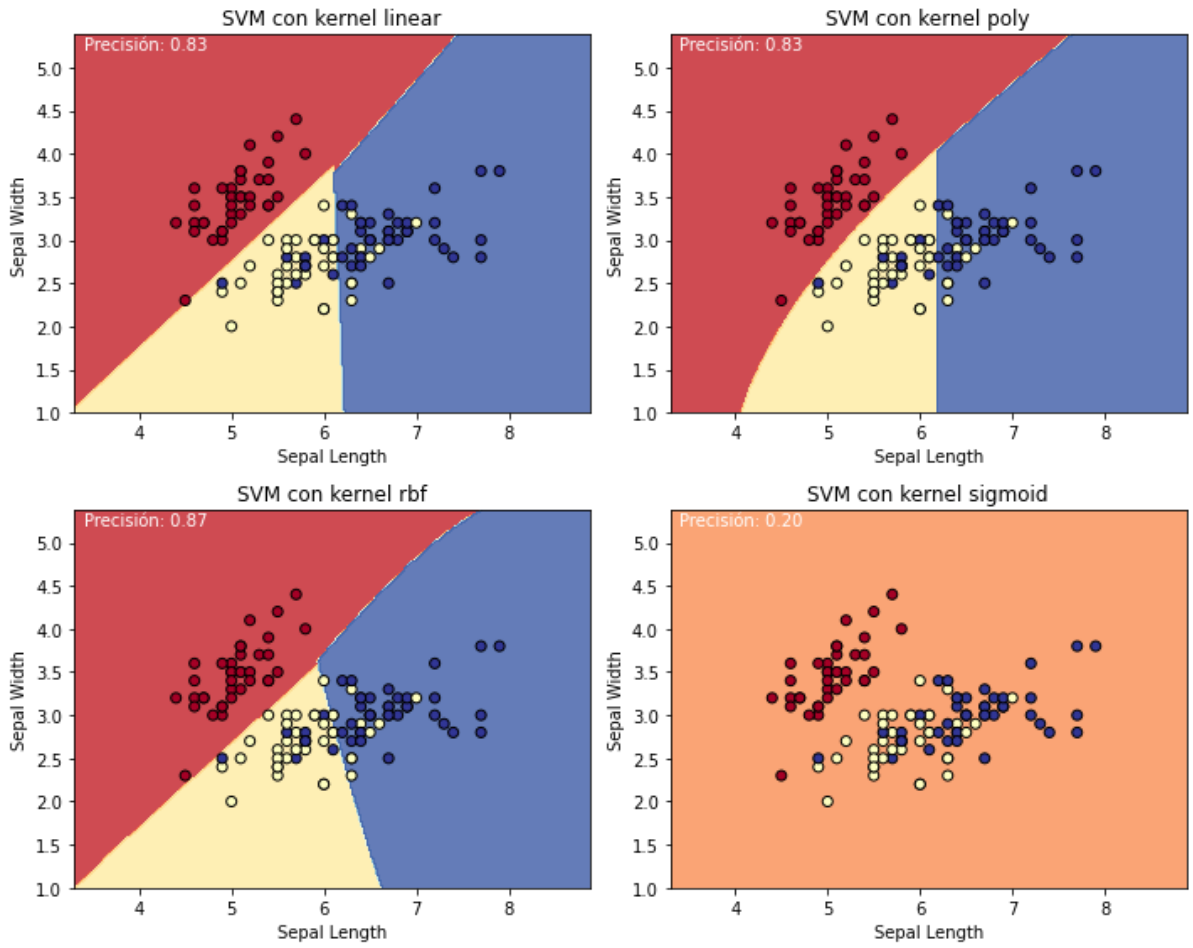

Figura 2: Divisiones espaciales generadas a través de diferentes kernels de un SVM en 2 dimensiones en el dataset Iris. Fuente: Elaboración propia.

Naive Bayes

Naive Bayes es un algoritmo de aprendizaje automático supervisado basado en el teorema de Bayes. Se utiliza para problemas de clasificación basándose en la probabilidad condicional de que un objeto pertenezca a una determinada clase dadas sus características o atributos.

El clasificador Naive Bayes asume que las características son independientes entre sí, dado que son condicionalmente independientes de la clase dada la clase. Esta suposición simplifica el cálculo de las probabilidades y permite una rápida estimación de la probabilidad de pertenencia a una clase.

Dado un conjunto de datos de entrenamiento que contiene objetos con características y etiquetas de clase conocidas, el clasificador Naive Bayes estima las probabilidades de pertenencia a cada clase utilizando el teorema de Bayes. La fórmula del teorema de Bayes es:

$$P(C_k|x_1, x_2, \dots, x_n) = \frac{P(C_k) \cdot P(x_1, x_2, \dots, x_n|C_k)}{P(x_1, x_2, \dots, x_n)}$$

Donde:

- $P(C_k|x_1, x_2, \dots, x_n)$ es la probabilidad de que el objeto pertenezca a la clase C_k dado los valores de sus características $x_1, x_2, \dots, x_n$.
- $P(C_k)$ es la probabilidad a priori de la clase C_k que se estima contando la frecuencia de cada clase en el conjunto de datos de entrenamiento.
- $P(x_1, x_2, \dots, x_n|C_k)$ es la probabilidad de las características $x_1, x_2, \dots, x_n$ dado que el objeto pertenece a la clase C_k. Esta probabilidad se estima asumiendo independencia condicional entre las características y calculando la frecuencia de cada combinación de características y clase.
- $P(x_1, x_2, \dots, x_n)$ es la probabilidad marginal de las características, que se utiliza como factor de normalización.

Una vez que se estiman las probabilidades de pertenencia a cada clase para un objeto dado, se selecciona la clase con la probabilidad más alta como la clasificación predicha para ese objeto.

Métodos Ensemble

Los métodos ensemble son técnicas de aprendizaje automático que combinan múltiples modelos predictivos para obtener una predicción más precisa y robusta. Estos métodos se basan en la idea de que la combinación de varios modelos puede superar las limitaciones individuales de cada uno de ellos, y producir una predicción más confiable y generalizable.

Existen diferentes enfoques para construir métodos ensemble, pero dos de los más populares son el promedio de modelos y el ensamblado por votación.

El promedio de modelos implica entrenar varios modelos independientes sobre el mismo conjunto de datos de entrenamiento y luego promediar sus predicciones para obtener una predicción final. Esto se aplica tanto a problemas de clasificación como de regresión. Por ejemplo, en el caso de la regresión, se promedian las predicciones numéricas de los modelos individuales, mientras que, en la clasificación, se puede utilizar el promedio de las probabilidades de pertenencia a cada clase.

El ensamblado por votación implica entrenar varios modelos independientes y dejar que voten para tomar una decisión final. En el caso de la clasificación, cada modelo emite su propia predicción de clase, y la clase final se determina mediante una votación mayoritaria o ponderada. En el caso de la regresión, se puede utilizar el promedio de las predicciones numéricas de los modelos individuales.

Los métodos ensemble también pueden utilizar técnicas de remuestreo, como el bagging y el boosting. El bagging (bootstrap aggregating) implica entrenar múltiples modelos independientes utilizando subconjuntos aleatorios de datos de entrenamiento con reemplazo, y luego promediar o combinar sus predicciones. Esto ayuda a reducir la varianza y mejorar la generalización del modelo. El boosting implica entrenar modelos secuenciales, donde cada modelo se enfoca en los errores del modelo anterior. Los modelos se ponderan según su rendimiento y se combinan para obtener una predicción final.

Algunos ejemplos populares de métodos ensemble son el Random Forest y el Gradient Boosting. El Random Forest utiliza el bagging junto con árboles de decisión, mientras que el Gradient Boosting utiliza el boosting para construir un modelo fuerte a partir de modelos débiles, como árboles de decisión.

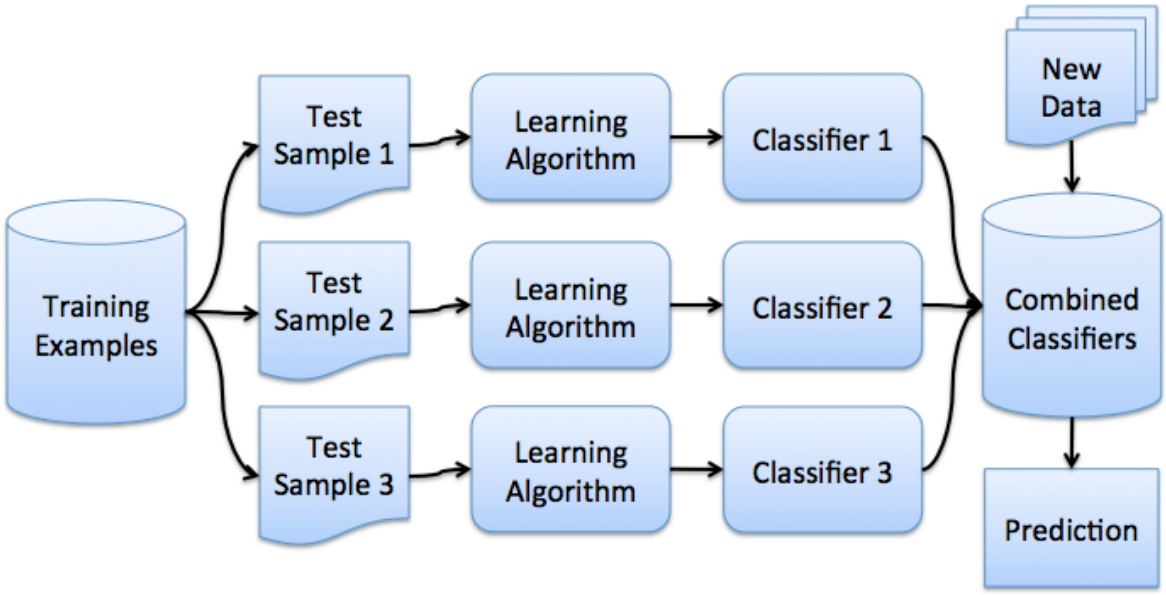

Figura 3: Representación gráfica del funcionamiento de los métodos ensemble. Fuente: Departamento de Ciencias de la Computación de la Universidad de Sevilla.

Ventajas y Desventajas del Aprendizaje Supervisado

Una vez presentados los algoritmos más utilizados, es importante exponer también las ventajas y las limitaciones que conllevan estos mismos. Entre las ventajas encontramos:

Precisión: El aprendizaje supervisado puede producir modelos altamente precisos para problemas de clasificación y regresión. Esto se debe a que el modelo se entrena con datos etiquetados que permiten una evaluación precisa del rendimiento.

Interpretabilidad: Los modelos de aprendizaje supervisado son a menudo interpretables, lo que significa que es posible entender cómo se tomó una decisión específica. Esto puede ser particularmente importante en áreas como la medicina o el derecho, donde se requiere transparencia y explicabilidad.

Generalización: Los modelos de aprendizaje supervisado pueden generalizar bien a nuevos datos, siempre que los datos de entrenamiento sean representativos de los datos de prueba.

Gran cantidad de algoritmos disponibles: Existe una amplia variedad de algoritmos de aprendizaje supervisado disponibles, lo que permite a los usuarios seleccionar el algoritmo más adecuado para su problema específico.

Dentro de las desventajas encontramos:

Necesidad de datos etiquetados: El aprendizaje supervisado requiere que los datos de entrenamiento estén etiquetados, lo que puede ser costoso y laborioso.

Sensibilidad al sesgo de selección de muestra: El aprendizaje supervisado puede estar sujeto a un sesgo de selección de muestra si la muestra de entrenamiento no es representativa de la población en general. Esto puede dar lugar a modelos que no generalizan bien en nuevos datos.

Limitaciones en la capacidad de capturar patrones complejos: Algunos modelos de aprendizaje supervisado pueden tener dificultades para capturar patrones complejos en los datos, lo que puede resultar en un rendimiento subóptimo.

Riesgo de sobreajuste: El aprendizaje supervisado puede estar sujeto al riesgo de sobreajuste, lo que significa que el modelo se ajusta demasiado a los datos de entrenamiento y no se generaliza bien a nuevos datos.

Desafíos del aprendizaje supervisado

Sesgo en los datos: El sesgo en los datos es uno de los desafíos más importantes en el aprendizaje supervisado. Si los datos de entrenamiento están sesgados, el modelo también estará sesgado. Los sesgos pueden ser de diferentes tipos, como sesgo de selección de muestra, sesgo de atributo y sesgo de etiqueta.

Selección de características: La selección de características es un proceso crítico en el aprendizaje supervisado. Si se seleccionan características irrelevantes o ruidosas, el rendimiento del modelo puede verse afectado negativamente.

Escalabilidad: El aprendizaje supervisado requiere grandes cantidades de datos y computación para entrenar modelos precisos. La escalabilidad es un desafío importante para el aprendizaje supervisado, especialmente en aplicaciones de Big Data.

Aprendizaje No Supervisado

El aprendizaje no supervisado es una rama del aprendizaje automático que se enfoca en descubrir patrones y estructuras en datos sin etiquetas o información de salida predefinida. A diferencia del aprendizaje supervisado, donde se proporcionan ejemplos de entrada y salida para entrenar el modelo, en el aprendizaje no supervisado, el objetivo es encontrar patrones y relaciones en los datos sin un conocimiento previo.

Formalmente:

El aprendizaje no supervisado es un tipo de aprendizaje automático que tiene como objetivo encontrar patrones en los datos sin la ayuda de ejemplos etiquetados. En lugar de tratar de predecir una salida específica, el objetivo del aprendizaje no supervisado es descubrir estructuras en los datos y agruparlos en categorías o clases similares. (Bengio, 2013)

Este objetivo es logrado mediante la identificación de patrones en los datos y la creación de representaciones internas que resuman la información en los datos sin la necesidad de una etiqueta de salida específica.

Hoy en día el aprendizaje no supervisado es utilizado en una gran cantidad de industrias, con fines muy diversos. Algunos de ellos podrían ser:

- Segmentación de clientes: Podemos identificar grupos de clientes con comportamientos similares, lo que puede ayudar a los especialistas en marketing a personalizar la publicidad y las promociones para cada grupo.

- Categorización de documentos o artículos: Un algoritmo de clustering puede identificar automáticamente los temas principales en un conjunto de artículos de noticias, como política, deportes y tecnología, lo que permite la categorización y organización de estos.

- Visión computacional: Podemos llegar a utilizar algoritmos de aprendizaje no supervisados para tareas de percepción visual, como el reconocimiento de objetos, lo cual se aplica a es útil en técnicas de visión por computadora.

- Detección de anomalías: El aprendizaje no supervisado también se utiliza para detectar anomalías en los datos, lo que puede ser útil en áreas como la detección de fraudes o la detección de fallas en maquinaria industrial. Por ejemplo, en una fábrica, los sensores pueden recolectar datos sobre el rendimiento de las máquinas. Al aplicar técnicas de aprendizaje no supervisado a estos datos, se pueden identificar patrones normales de rendimiento y detectar cualquier anomalía que pueda indicar un problema en la máquina.

La mayoría de las aplicaciones del aprendizaje no supervisado pueden a su vez agruparse en 3 categorías principales: clustering, reglas de asociación y técnicas de reducción de la dimensionalidad. En las siguientes páginas las definiremos y estudiaremos en detalle los algoritmos más comunes usados en cada una de ellas.

Clustering

Los algoritmos de clustering son una técnica de aprendizaje no supervisado utilizada para agrupar conjuntos de datos similares en función de sus características.

El objetivo del clustering es encontrar patrones y estructuras en los datos que puedan ser utilizados para comprender mejor el conjunto de datos y/o tomar decisiones informadas.

Los algoritmos de clustering dividen un conjunto de datos en grupos o clusters de manera que los datos dentro de un cluster sean similares entre sí y diferentes de los datos en otros clusters. Los algoritmos de agrupación en clústeres se pueden clasificar en varios tipos: exclusivos/superpuestos, jerárquicos y probabilísticos.

Exclusivos/Superpuestos

Los clústeres exclusivos se refieren a conjuntos de datos que se agrupan en grupos distintos y no comparten elementos entre ellos. En otras palabras, cada punto de datos solo se asigna a un solo clúster y no hay superposición entre los diferentes clústeres.

Por otro lado, los clústeres superpuestos se refieren a conjuntos de datos que se agrupan en grupos que pueden compartir algunos elementos en común. En otras palabras, algunos puntos de datos pueden pertenecer a más de un clúster al mismo tiempo, lo que crea superposición entre los diferentes clústeres.

Es importante tener en cuenta que la presencia de clústeres superpuestos o exclusivos depende del algoritmo de clustering utilizado y del conjunto de datos específico. Algunos algoritmos de clustering, pertenecientes a esta categoría son:

K-Means

El algoritmo de K-medias es un algoritmo de clustering utilizado en aprendizaje automático y análisis de datos. Su objetivo es agrupar un conjunto de datos en K clústeres distintos basados en sus características.

El proceso de agrupación se realiza mediante la minimización de una función objetivo conocida como "suma de los cuadrados de las distancias" (SSD) entre los puntos de datos y los centroides de los clústeres. El algoritmo funciona de la siguiente manera:

- Seleccionar el número K de clústeres que se desean crear y los centroides iniciales para cada clúster.

- Asignar cada punto de datos al clúster más cercano según la distancia euclidiana entre los datos y los centroides.
- Calcular los nuevos centroides para cada clúster como el punto medio de todos los datos asignados al clúster.
- Repetir los pasos 2 y 3 hasta que los centroides de los clústeres ya no cambien significativamente o se alcance un número máximo de iteraciones.

Una vez que se completa el proceso de agrupación, se pueden realizar análisis adicionales sobre los clústeres identificados y sus características, como la distribución de los datos y la variación dentro y entre los clústeres.

Es importante tener en cuenta que el algoritmo de K-medias puede ser sensible a la elección inicial de los centroides y puede no funcionar bien en conjuntos de datos con formas irregulares o con características no lineales. Además, también puede haber problemas con los clústeres superpuestos o excluyentes.

Jerárquicos

A diferencia del algoritmo de K-medias, la agrupación en clusters jerárquica no requiere que se especifique previamente el número de clústeres que se desean crear.

En la agrupación en clusters jerárquica, los datos se agrupan en una jerarquía de clústeres, a menudo representada como un árbol o dendograma. El proceso de agrupación comienza con cada punto de datos en su propio clúster y continúa combinando los clústeres existentes en clústeres más grandes según su similitud, hasta que todos los puntos de datos se agrupen en un solo clúster.

Normalmente se utilizan cuatro métodos diferentes para medir la similitud:

- Enlace de Ward: este método establece que la distancia entre dos clústeres se define por el incremento en la suma de cuadrados después de fusionar los clústeres.
- Enlace promedio: este método está definido por el promedio de distancia entre dos puntos en cada clúster.
- Enlace completo (o máximo): este método está definido por la distancia máxima entre dos puntos en cada clúster.
- Enlace único (o mínimo): este método viene definido por la distancia mínima entre dos puntos en cada clúster.

Un espacio vectorial se define como una distancia d_{ij} entre 2 vectores $x^{(i)}$ y $x^{(j)}$ si cumple:

$d_{ij} \geq 0$. Además, si $d(u, v) = 0, u = v$.

$d_{ij} = d_{ji}$ o propiedad simétrica.

$d_{ij} + d_{jk} \geq d_{ik}$ o propiedad triangular.

A su vez, existen varias métricas para calcular estas distancias:

- Distancia euclidiana

$$d_{ij} = \sqrt{\sum_{k=1}^{p} \left(x_{ik} - x_{jk}\right)^2}$$

- Distancia de Manhattan

$$d_{ij} = \sum_{k=1}^{p} \left|x_{ik} - x_{jk}\right|$$

- Coeficiente de correlación

$$d_{ij} = 1 - \frac{\sum_{k=1}^{p}(x_{ik} - \bar{x_i})\left(x_{jk} - \bar{x_j}\right)}{\sqrt{\sum_{k=1}^{p}(x_{ik} - \bar{x_i})^2 \sqrt{\sum_{k=1}^{p}\left(x_{jk} - \bar{x_j}\right)^2}}}$$

- Distancia de Mahalanobis

$$d_{ij} = \left(x^{(i)} - x^{(j)}\right)^T \Gamma^{-1} \left(x^{(i)} - x^{(j)}\right)$$

Existen dos enfoques principales para la agrupación en clusters jerárquica: el enfoque aglomerativo y el enfoque divisivo. En el enfoque aglomerativo, cada punto de datos comienza en su propio clúster y se combinan los clústeres más cercanos entre sí hasta que se obtiene un solo clúster que contiene todos los datos.

En el enfoque divisivo, todos los puntos de datos comienzan en un solo clúster y se dividen en clústeres más pequeños a medida que se avanza en la jerarquía.

Una vez que se completa el proceso de agrupación, se puede determinar el número óptimo de clústeres al cortar el dendograma en una altura que represente un equilibrio entre la homogeneidad dentro de los clústeres y la heterogeneidad entre los clústeres.

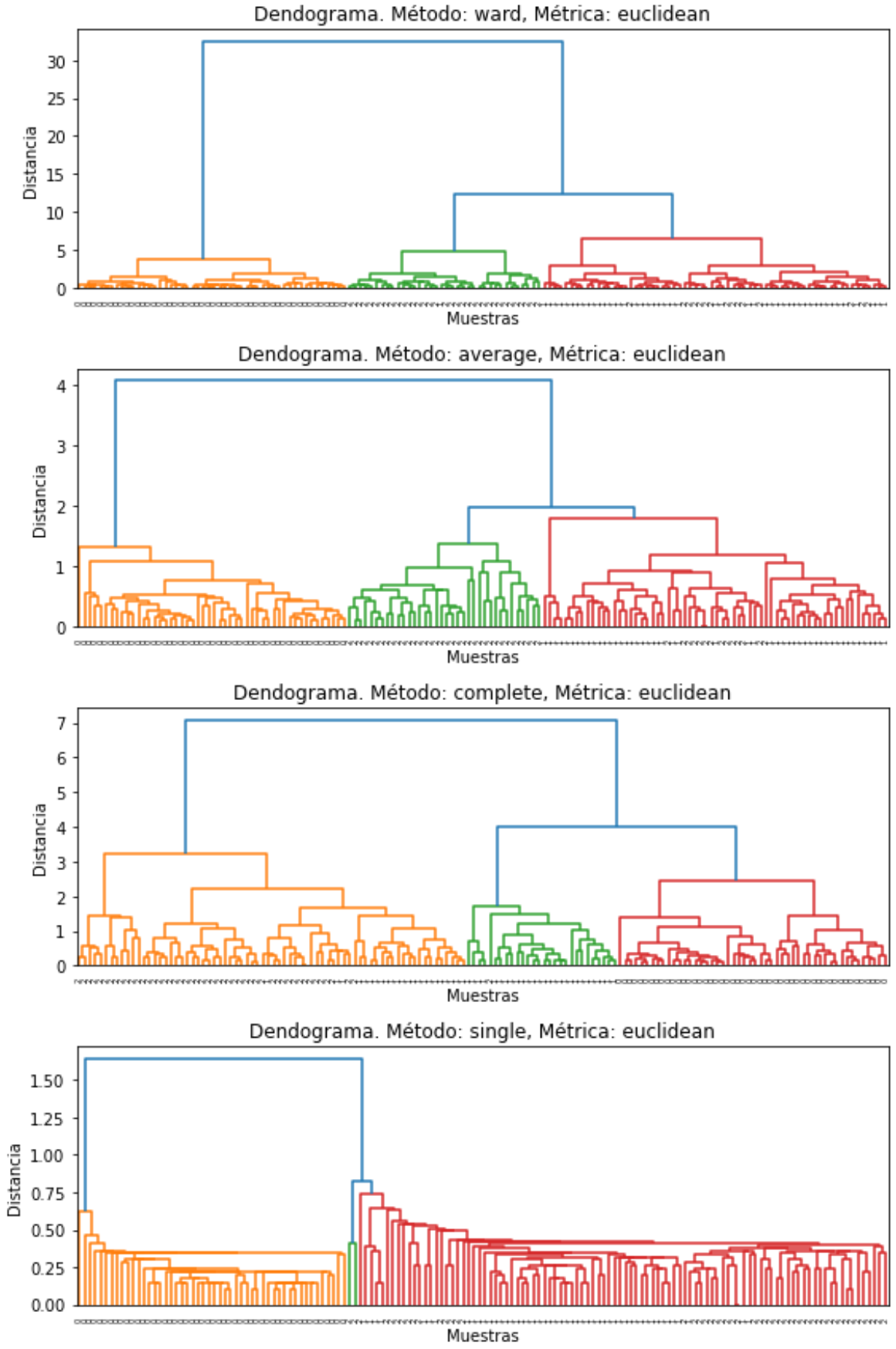

Figura 4: Diferentes dendogramas generados en el dataset Iris a través de distintos métodos de enlace. Fuente: Elaboración propia.

Probabilístico

La agrupación de clústeres probabilística es un enfoque de clustering que se basa en la asignación de probabilidades a la pertenencia de cada punto de datos a cada uno de los clústeres. A diferencia de los métodos de clustering tradicionales, donde cada punto de datos se asigna a un único clúster, la agrupación de clústeres probabilística permite que un punto de datos pertenezca a múltiples clústeres con diferentes grados de pertenencia.

En la agrupación de clústeres probabilística, se modela la distribución de probabilidad conjunta de los puntos de datos y las etiquetas de clúster utilizando un modelo de mezcla Gaussiana. Este modelo asume que los datos provienen de una combinación de varias distribuciones gaussianas y que cada punto de datos se genera a partir de una distribución específica.

El modelo de mezcla Gaussiana se entrena utilizando un algoritmo de estimación de máxima verosimilitud para determinar los parámetros óptimos que describen las distribuciones gaussianas. Una vez que se ha entrenado el modelo, se utiliza la probabilidad de pertenencia a cada clúster para asignar cada punto de datos a uno o varios clústeres.

La agrupación de clústeres probabilística es útil en situaciones en las que los datos no se ajustan a una distribución de probabilidad específica o cuando los puntos de datos pueden pertenecer a más de un clúster. También se utiliza a menudo en aplicaciones de clasificación de texto y análisis de imágenes, donde la pertenencia de los puntos de datos a diferentes categorías puede ser ambigua o incierta.

Reglas de asociación

La idea principal detrás de las reglas de asociación es encontrar relaciones entre diferentes elementos en un conjunto de datos, identificando los conjuntos de elementos que aparecen juntos con más frecuencia de lo esperado por casualidad. Estas relaciones se expresan en forma de reglas, donde se indica la probabilidad de que, si un elemento A aparece, entonces también aparecerá otro elemento B.

La técnica de reglas de asociación se basa en dos conceptos principales: soporte y confianza. El soporte se refiere a la frecuencia con la que un conjunto de elementos aparecen juntos en el conjunto de datos, mientras que la confianza mide la frecuencia con la que un elemento B aparece en las transacciones que contienen el elemento A.

Algoritmos a priori

El algoritmo A priori se utiliza para encontrar conjuntos de elementos que aparecen juntos con una frecuencia mayor a un umbral mínimo predefinido. A partir de estos conjuntos, se pueden generar reglas de asociación que expresan la probabilidad de la aparición de un elemento dado, dada la presencia de otro elemento.

Se utilizan dentro de conjuntos de datos transaccionales para identificar conjuntos de elementos frecuentes, o colecciones de elementos, para identificar la probabilidad de consumir un producto dado el consumo de otro producto.

Reducción de la dimensionalidad

Las técnicas de reducción de la dimensionalidad en el aprendizaje no supervisado son un conjunto de herramientas y algoritmos que se utilizan para procesar y analizar conjuntos de datos de alta dimensionalidad y reducir su complejidad mediante la identificación de patrones y relaciones significativas entre las variables.

Mas formalmente:

La reducción de la dimensionalidad es una técnica que busca transformar un conjunto de datos complejos y altamente correlacionados en un conjunto de datos de menor dimensión que mantenga la mayor parte de la información relevante. Esta técnica es ampliamente utilizada en el aprendizaje no supervisado para la extracción de características y la representación eficiente de datos en espacios de menor dimensión, lo que permite una mejor visualización, interpretación y comprensión de los datos. (Hinton, 2006)

Este conjunto de técnicas es usado comúnmente en la fase de preprocesamiento de datos.

Algunos de los principales beneficios que genera aplicar la técnica de reducción de dimensionalidad son los siguientes:

- Reducir las dimensiones de las características implica una reducción del espacio requerido para almacenar el conjunto de datos, porque este también se reduce.
- El tiempo de entrenamiento de modelos es menor para dimensiones reducidas.
- Se facilita la visualización de datos más rápidamente gracias a la reducción de características del conjunto de datos.
- Desaparecen las características redundantes en el ámbito de la multicolinealidad.

Existen a su vez varios métodos de reducción de la dimensionalidad, entre ellos podemos encontrar el Análisis de Componentes Principales (PCA por sus siglas en ingles), el Análisis Factorial, la Descomposición en Valores Singulares (SVD) o los Métodos Biplot.

Análisis de Componentes Principales (PCA)

El análisis de componentes principales es una técnica de reducción de la dimensionalidad que busca transformar un conjunto de variables correlacionadas en un conjunto más pequeño de variables no correlacionadas llamadas componentes principales. (Jollife, 2005)

Se utiliza para transformar un conjunto de variables altamente correlacionadas en un conjunto más pequeño y manejable de variables no correlacionadas, llamadas componentes principales. Esta técnica se utiliza comúnmente en el análisis exploratorio de datos para la visualización, clasificación y agrupación de datos.

En el ACP, se busca encontrar una combinación lineal de variables que maximice la varianza total de los datos. Cada componente principal resultante es una combinación ponderada de todas las variables originales, donde los pesos se eligen de manera que la varianza de la proyección de los datos en el componente sea máxima. De esta manera, los componentes principales explican la mayor parte de la variabilidad en los datos originales y permiten reducir la dimensionalidad sin perder información importante.

El primer componente principal es la dirección que maximiza la varianza del conjunto de datos. Si bien el segundo componente principal también encuentra la varianza máxima en los datos, no tiene ninguna correlación con el primer componente principal, lo que genera una dirección que es perpendicular u ortogonal al primer componente. Este proceso se repite en función del número de dimensiones, donde un siguiente componente principal es la dirección ortogonal a los componentes anteriores con mayor varianza. (IBM, s.f.)

Dado un conjunto de datos con n observaciones y p variables, el PCA se basa en los siguientes pasos:

1. Centrar los datos: Se resta la media de cada variable para que los datos estén centrados en cero. Esto implica restar a cada valor de la variable su media correspondiente.

2. Calcular la matriz de covarianza: Se calcula la matriz de covarianza, que es una matriz simétrica $p \times p$ que contiene las covarianzas entre todas las posibles combinaciones de variables.

3. Obtener los autovalores y autovectores de la matriz de covarianza: Se calculan los autovectores y autovalores de la matriz de covarianza. Los autovectores representan las direcciones principales en el espacio de variables, y los autovalores indican la cantidad de varianza explicada por cada componente principal.

4. Seleccionar las componentes principales: Se seleccionan las k componentes principales con los autovalores más grandes, que explican la mayor parte de la varianza total. Estas componentes principales forman un nuevo espacio de variables que captura la estructura de los datos originales de manera más compacta.

5. Proyectar los datos en el nuevo espacio: Se proyectan los datos originales en el nuevo espacio de componentes principales para obtener una representación reducida de los datos.

La proyección de los datos en el nuevo espacio se calcula mediante la multiplicación de la matriz de datos centrados por la matriz de autovectores seleccionados.

Es importante tener en cuenta que el PCA asume linealidad en los datos y puede no ser adecuado para datos no lineales. En tales casos, se pueden utilizar técnicas de reducción de dimensionalidad no lineales, como el Análisis de Componentes Independientes (ICA) o el t-SNE (t-Distributed Stochastic Neighbor Embedding).

Análisis Factorial

El análisis factorial es una técnica estadística utilizada para identificar factores subyacentes que expliquen la variabilidad observada en un conjunto de datos. En el análisis factorial, se busca reducir la dimensionalidad de los datos mediante la identificación de factores latentes que explican la mayor parte de la varianza observada. Cada variable original se asocia con uno o más factores, y se estima un conjunto de coeficientes que indican la fuerza de la relación entre las variables y los factores. (Gorsuch, 1997)

Estos factores son variables latentes que no pueden medirse directamente, pero que se pueden inferir a partir de las correlaciones observadas entre las variables originales.

Cada variable original se asocia con uno o más factores, y se estima un conjunto de coeficientes que indican la fuerza de la relación entre las variables y los factores. De esta manera, el análisis factorial permite reducir la dimensionalidad de los datos y proporciona una forma más compacta y manejable de representar la estructura subyacente de los datos.

Además, el análisis factorial también se puede utilizar en la confirmación de hipótesis sobre la estructura subyacente de los datos y para evaluar la fiabilidad y validez de las escalas de medición.

Podemos distinguir 2 tipos principales de Análisis Factorial, el Exploratorio y el Confirmatorio.

El análisis factorial exploratorio (AFE) es una técnica estadística que se utiliza para descubrir la estructura interna de un conjunto grande de variables. En este método, se asume que hay factores subyacentes que se relacionan con grupos de variables, y se utilizan las cargas de los factores para inferir su relación con las variables. Este tipo de análisis factorial es el más común y se utiliza para explorar y descubrir la estructura subyacente de los datos. (Fabrigar, 1999)

Por otro lado, *el análisis factorial confirmatorio (AFC) tiene como objetivo determinar si el número de factores y sus cargas se corresponden con una teoría previa sobre los datos. En este caso, se parte de la hipótesis a priori de que existen unos factores preestablecidos que se relacionan con subconjuntos específicos de las variables. El AFC proporciona un nivel de confianza para aceptar o rechazar la hipótesis previa. También considera las variables como dos medidas que pueden ser cuantificadas constantemente. (Fabrigar, 1999)*

Los métodos de obtención de los factores más ampliamente utilizados son el método de Componentes Principales, la Factorización de Ejes y el método de Máxima Verosimilitud.

La Factorización de Ejes Principales (FEP) es un método iterativo basado en la extracción sucesiva de aquellos factores que explican la mayor parte de la varianza común y, también, robusto a violaciones del supuesto de normalidad. Sin embargo, la principal limitación de este método es que no proporciona índices de tampoco permite la computación de test de significancia y de los intervalos de confianza (Fabrigar, 1999)

El método de Máxima Verosimilitud, aunque no presenta los inconvenientes anteriores, presupone distribución normal multivariada y ausencia de casos atípicos, lo cual con cierto tipo de datos es difícil de conseguir.

Existen diversos métodos de rotación de factores que se pueden clasificar en dos categorías más amplias: rotaciones ortogonales y rotaciones oblicuas.

Las rotaciones ortogonales, como varimax, quartimax y equimax, generan factores no correlacionados, lo que las hace apropiadas para casos en los que se asume que los factores son independientes entre sí.

Por otro lado, las rotaciones oblicuas, como la rotación promax, permiten que los factores estén correlacionados, lo que podría ser más apropiado cuando se asume que los factores están relacionados entre sí. (Kline, 2013)

Existen diversas formas de determinar el número de factores a retener en un análisis factorial, algunas de las más importantes son:

Criterio de Kaiser: Este criterio se basa en la idea de que se deben retener los factores con un valor propio (eigenvalue) mayor a 1.

Regla del codo: Esta técnica implica graficar los valores propios de cada factor en orden descendente y observar el punto en el que la gráfica forma un codo. Los factores hasta ese punto se retienen.

El porcentaje de la varianza total explicada es otro criterio que se puede utilizar para decidir el número de factores que serán retenidos. Como umbral para la extracción de los factores se suele establecer un mínimo de 60%.

Descomposición en Valores Singulares (SVD)

La descomposición en valores singulares (SVD) es una técnica matemática utilizada para descomponer una matriz en tres componentes, una matriz de vectores singulares izquierdos, una matriz diagonal de valores singulares y una matriz de vectores singulares derechos. (Golub, 2013)

La descomposición en valores singulares es otro enfoque distinto a la idea central de reducción de dimensionalidad, pero que en este caso tiene como objetivo el factorizar una matriz A principal en tres matrices de rango inferior.

Uno de los usos más habituales de esta técnica es la compresión de datos. La SVD se utiliza para comprimir datos al reducir la cantidad de información necesaria para representar una matriz.

Matemáticamente, un SVD descompone una matriz X tal que:

$$X = U\Sigma V^T$$

Donde sus elementos son:

X es la matriz de datos original.

U es una matriz de dimensiones $(m \times m)$ que contiene los vectores singulares izquierdos de X. Las columnas de U son ortogonales entre sí.

Σ es una matriz diagonal de dimensiones $(m \times n)$ que contiene los valores singulares de X en su diagonal principal. Los valores singulares son números no negativos y están ordenados de mayor a menor.

V^T es la matriz traspuesta de V que tiene dimensiones $(n \times n)$. Contiene los vectores singulares derechos de X. Las columnas de V también son ortogonales entre sí.

4. Aplicación de las técnicas a una base de datos

Descripción de la base de datos utilizada

Debido a su popularidad, utilidad y sencillez de uso, haremos uso del dataset “Iris Species” para implementar varios de los algoritmos expuestos a lo largo de este trabajo.

El conjunto de datos "Iris" es un conjunto de datos de clasificación ampliamente utilizado en el aprendizaje automático y la minería de datos. El conjunto de datos consiste en 150 instancias, donde cada instancia representa una flor de iris y tiene cuatro características: longitud del sépalo, ancho del sépalo, longitud del pétalo y ancho del pétalo. Además, cada instancia se etiqueta con la especie de iris correspondiente: setosa, versicolor o virginica. (Fisher, 1936)

El dataset "Iris" es un ejemplo de un problema de clasificación multiclase, donde el objetivo es clasificar cada instancia en una de las tres posibles clases de iris. Es un conjunto de datos balanceado, lo que significa que cada clase tiene el mismo número de instancias.

Además de ser ampliamente utilizado como un conjunto de datos de referencia en la literatura de aprendizaje automático y minería de datos, el conjunto de datos "Iris" también es conocido por su papel histórico en el desarrollo de la estadística y el análisis de datos.

El conjunto de datos fue recopilado por el estadístico británico Ronald A. Fisher en 1936 como parte de su trabajo en la clasificación de especies de plantas basado en medidas morfológicas, y fue introducido en su libro "The Use of Multiple Measurements in Taxonomic Problems" donde introdujo el concepto de análisis discriminante lineal para la clasificación de especies.

Desde entonces, el conjunto de datos "Iris" ha sido utilizado por muchos otros investigadores para evaluar y comparar técnicas de machine Learning y se ha convertido en un ejemplo de referencia en el campo.

Resultados

Dividiremos los resultados en 2 apartados, según la métrica que consideremos que estamos midiendo.

En el caso del aprendizaje supervisado, nuestro objetivo será medir la precisión de los modelos de clasificación utilizados. En cambio, en el caso del aprendizaje no supervisado mostraremos las diversas segmentaciones creadas con nuestros algoritmos de cluster, y analizaremos que porcentaje de varianza podemos explicar reduciendo las dimensiones de nuestra base de datos.

Comenzaremos nuestro estudio con un pequeño análisis descriptivo para contextualizar nuestra base de datos.

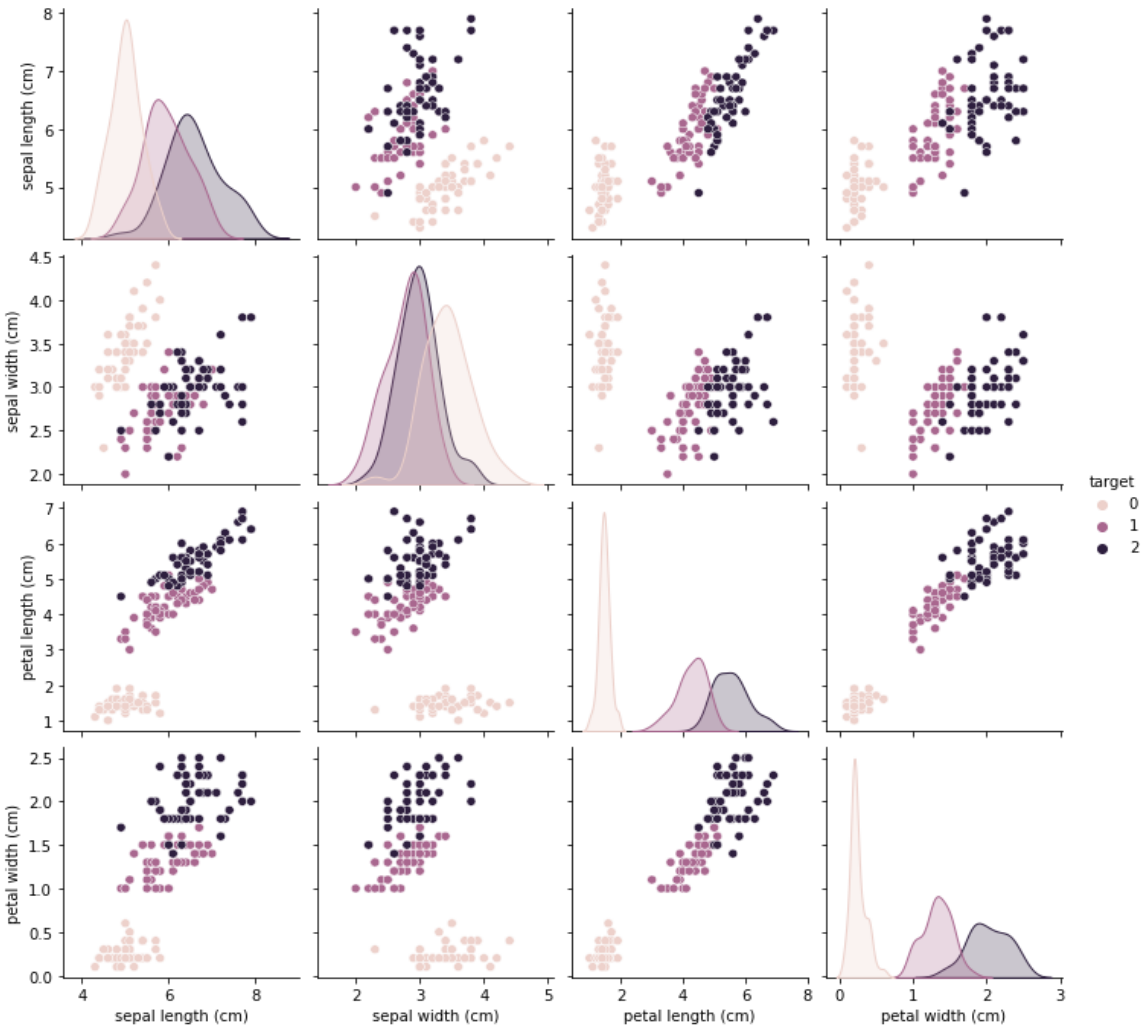

Figura 5: Gráficos de dispersión y distribuciones provenientes de las variables existentes en el dataset Iris, segmentadas por la variable de clasificación (Tipo de flor). Fuente: Elaboración propia.

Mostramos un gráfico de resumen con todas las distribuciones y los gráficos de dispersión, segmentados por target, siendo 0 = "Iris Setosa", 1 = Iris Versicolor", 2 = "Iris Virginica". Podemos ver que existen diferencias entre todas las variables, segmentando a partir la variable target a la que pertenezcan.

En lo referente al aprendizaje supervisado, analizaremos el rendimiento de distintos clasificadores. Comenzaremos con el SVM.

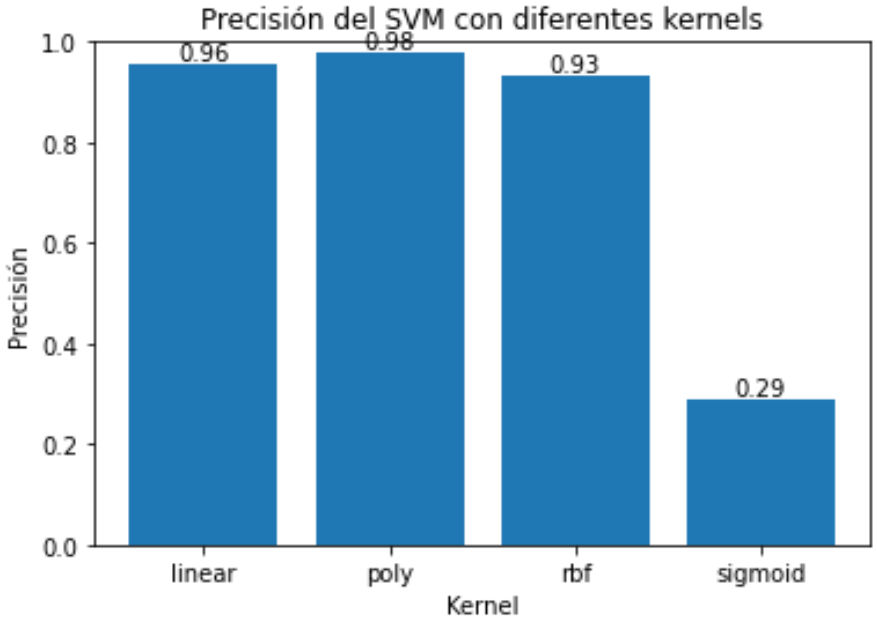

Figura 6: Precisiones obtenidas con el clasificador SVM, utilizando distintos kernels. Fuente: Elaboración propia.

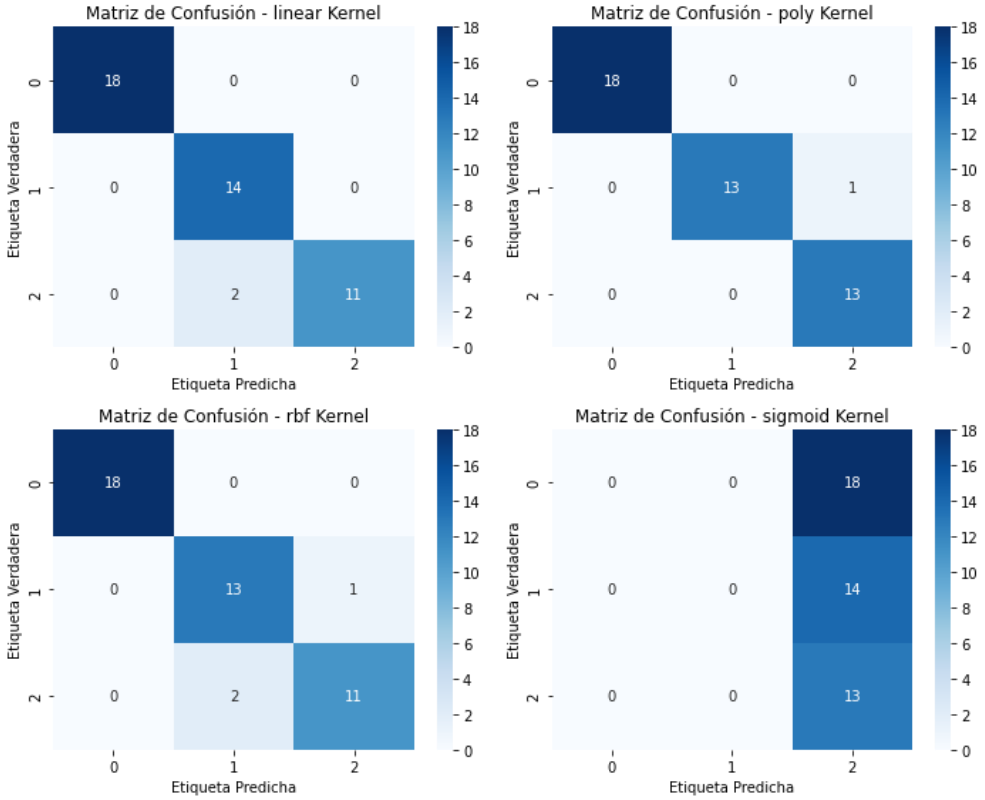

Figura 7: Matriz de confusión de las predicciones realizadas con los diferentes tipos de kernels. Fuente: Elaboración propia.

El modelo predice muy bien utilizando los 3 primeros algoritmos, siendo el mejor el linear (0.98). El sigmoide da resultados muy pobres.

El siguiente clasificador que estudiaremos será la clasificación logística.

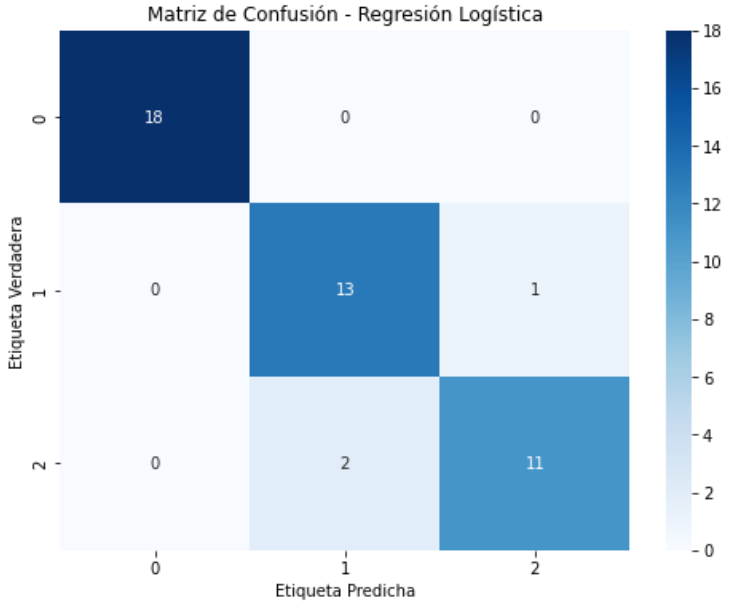

Figura 8: Matriz de confusión del clasificador logístico. Fuente: Elaboración propia.

```
Precisión: 0.9333333333333333
```

La precisión es muy buena, pero no llega al nivel de algunos kernels utilizados en SVM.

Continuaremos ahora estudiando el clasificador de árboles de decisión.

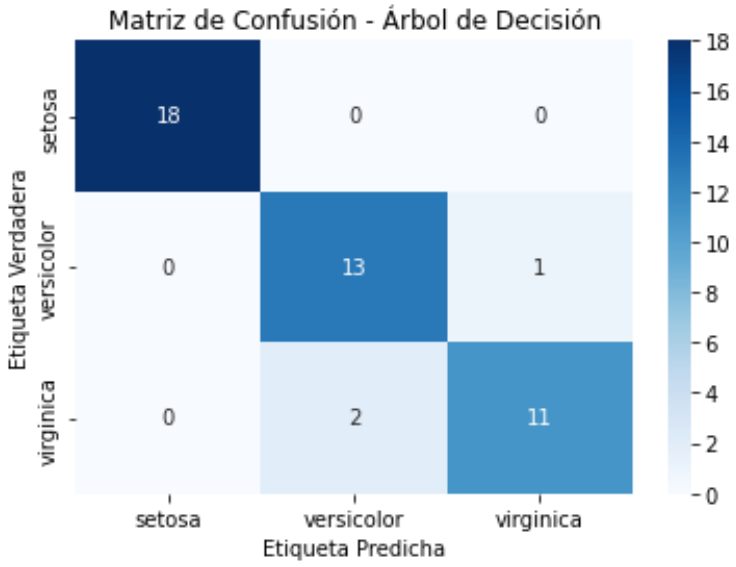

Figura 9: Matriz de confusión correspondiente al clasificador de árboles de decisión. Fuente: Elaboración propia.

```
Precisión: 0.9333333333333333
```

La precisión es idéntica a la obtenida con el clasificador logístico. Nuestro siguiente algoritmo utilizará métodos ensemble. En este caso, haremos uso de “Random Forest”.

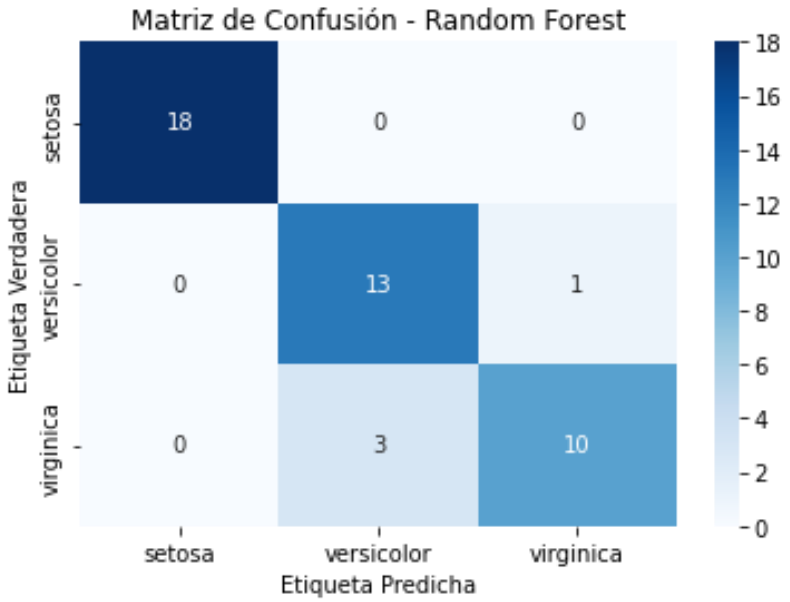

Figura 10: Matriz de confusión correspondiente al clasificador Random Forest. Fuente: Elaboración propia.

```
Precisión: 0.9111111111111111
```

La precisión es buena, aunque ligeramente inferior a la obtenida con otros clasificadores.

En el siguiente gráfico de barras podemos ver qué características son más importantes a la hora de construir el clasificador. Parece que petal width y petal length son las más relevantes.

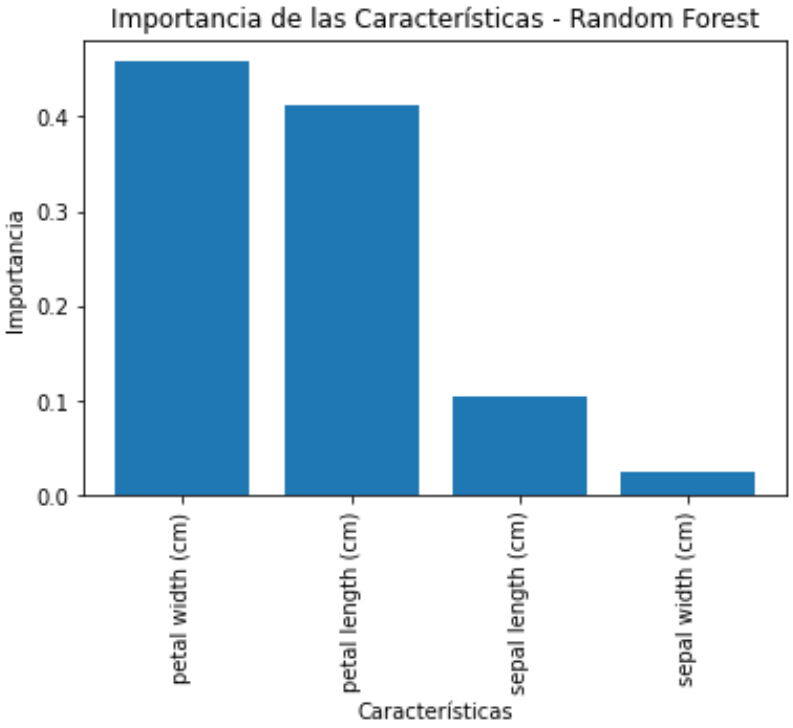

Figura 11: Grafico de barras con la importancia de cada variable en los métodos Random Forest. Fuente: Elaboración propia.

De seguido, estudiaremos el rendimiento del clasificador Naive-Bayes.

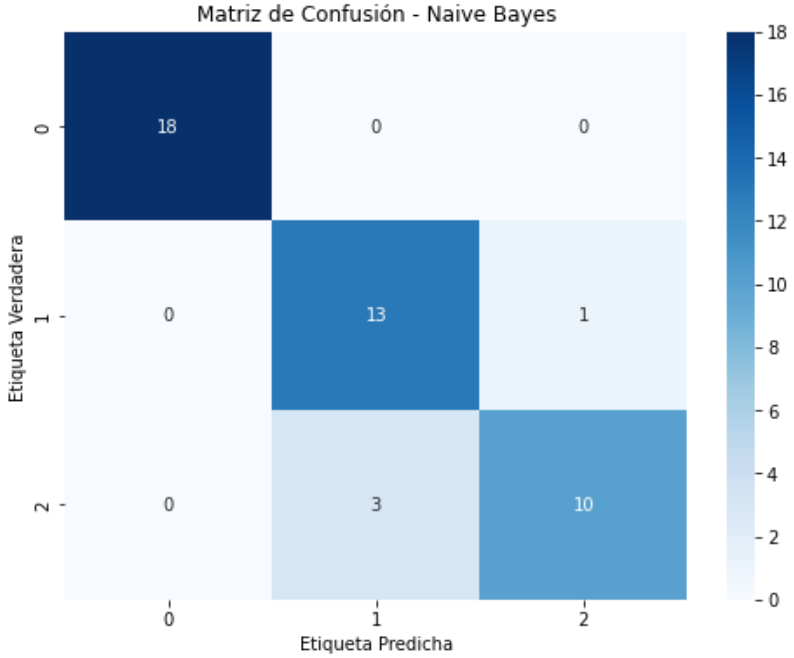

Figura 12: Matriz de confusión correspondiente al clasificador Naive Bayes. Fuente: Elaboración propia.

```
Precisión: 0.9111111111111111
```

Obtenemos una precisión idéntica a la obtenida con el clasificador Naive Bayes. Es buena pero no supera a otras obtenidas previamente.

Por último, estudiaremos el rendimiento del clasificador K vecinos cercanos.

Comenzamos realizando un gráfico de líneas con la precisión obtenida en dicho clasificador para distintos valores de k. Podemos apreciar que el máximo resultado se obtiene con k=10, por lo que usaremos dicho valor a la hora de calcular la precisión de este clasificador.

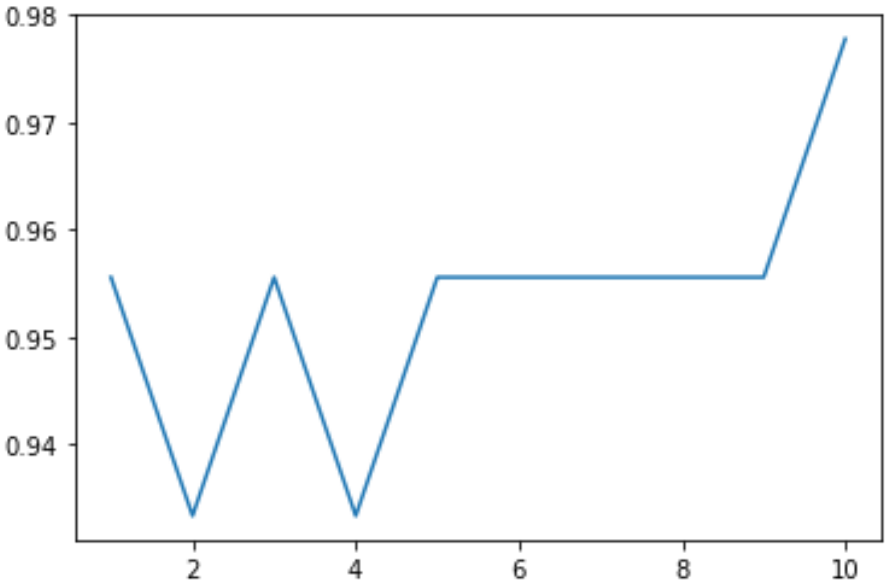

Figura 13: Gráfico con la precisión según el valor de k elegido en el algoritmo K-vecinos más cercanos. Fuente: Elaboración propia.

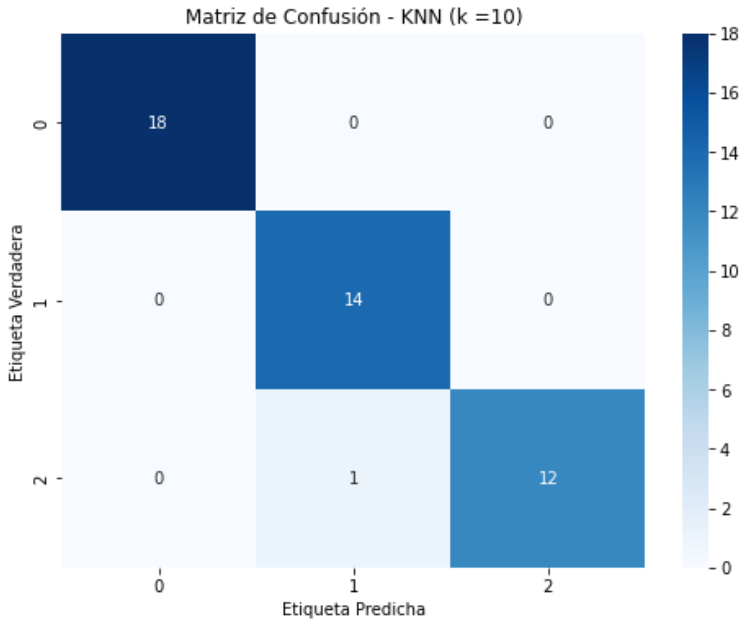

Figura 14: Matriz de confusión correspondiente al clasificador K-Vecinos más cercanos, con k =10. Fuente: Elaboración propia.

```
Precisión: 0.9777777777777777
```

La precisión obtenida con K-Vecinos cercanos es de las mejores obtenidas hasta este momento.

Una vez hemos obtenido los resultados de la precisión de cada uno de los algoritmos, crearemos una serie de tablas y gráficos para comparar el rendimiento de cada uno de ellos con relación al resto.

	Modelo	Precisión
0	SVM Poly	0.977778
1	KNN	0.977778
2	SVM Linear	0.955556
3	SVM RBF	0.933333
4	Regresión Logística	0.933333
5	Árboles de decisión	0.933333
6	Ensemble	0.911111
7	Naive Bayes	0.911111
8	SVM Sigmoid	0.288889

Figura 15: Resultados de la precisión obtenida con los distintos modelos de clasificación utilizados. Fuente: Elaboración propia.

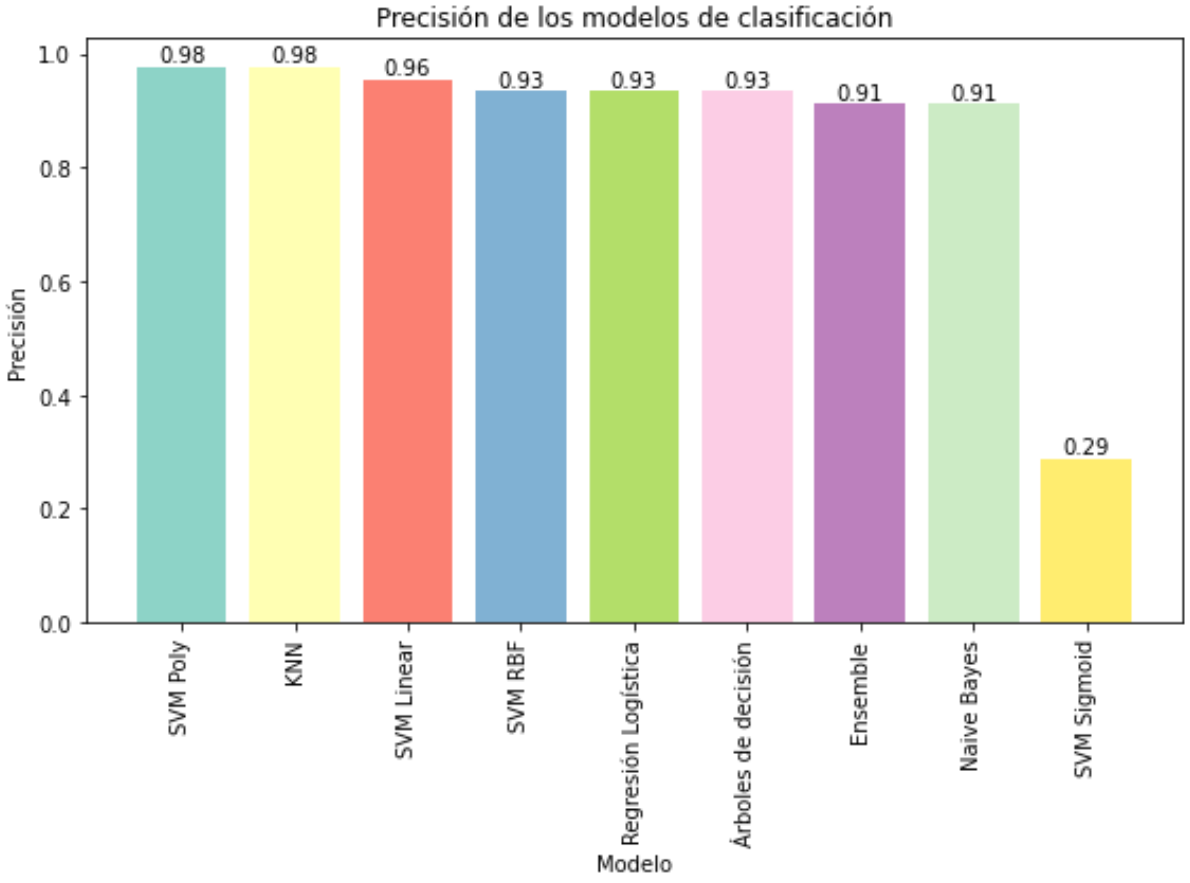

Figura 16: Gráfico de barras con la precisión obtenida en los clasificadores utilizados. Fuente: Elaboración propia.

Podemos apreciar como los mejores resultados los obtenemos con el clasificador SVM (con kernel poly) y con KNN (con k =10). En ambos casos, la precisión es 0.97778, muy buena. El clasificador SVM con kernel sigmoide es el único que no obtiene resultados satisfactorios (precisión de 0.29), pero el resto, aunque no llegan al nivel de los 2 primeros, obtienen resultados más que satisfactorios, con unas precisiones que se ubican entre el 0.955 del SVM con kernel lineal, y el 0.9111 de Naive Bayes o Random Forest.

Una vez hemos estudiado el rendimiento de diversos algoritmos de clasificación, comenzaremos con el estudio de los resultados obtenidos en nuestros modelos de aprendizaje no supervisado. Comenzaremos con los algoritmos de clustering, y tendremos como objetivo principal ver como se segmentan automáticamente los datos que disponemos. El algoritmo utilizado para realizar dicha segmentación será k-medias.

El primero de los pasos para aplicarlo será estudiar el número óptimo de clústeres en los cuales queremos dividir nuestros datos. Para tomar esta decisión podemos graficar la curva de la suma de los cuadrados intra-cluster y aplicar el método del codo, viendo donde se aplana la gráfica.

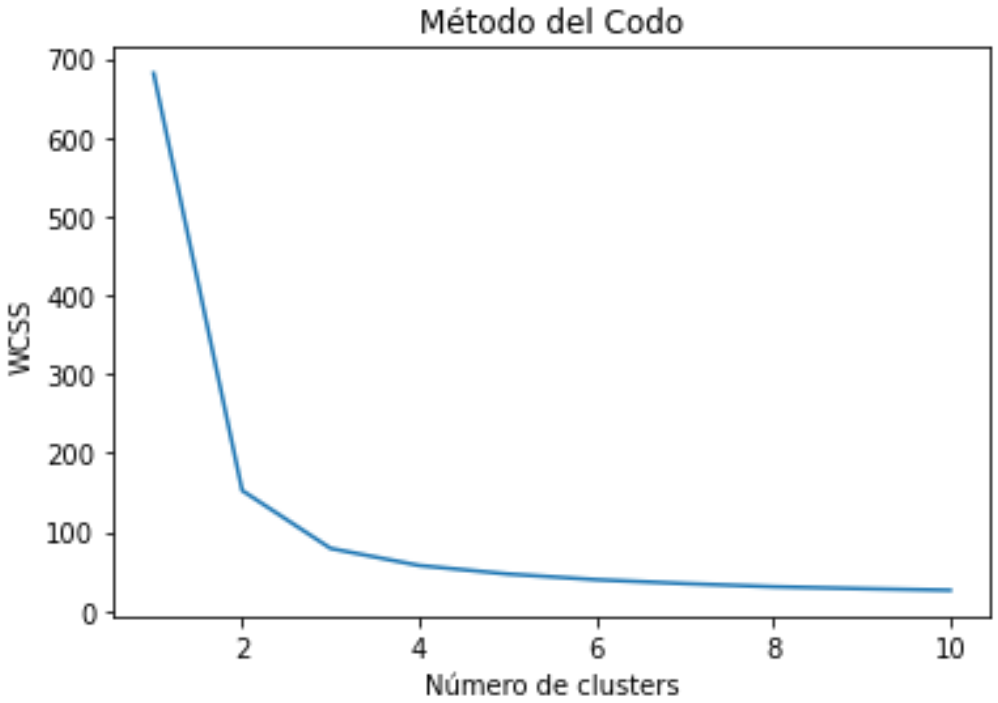

Figura 17: Gráfico con la suma de los cuadrados intra cluster. Fuente: Elaboración propia.

En este caso, parece que la solución óptima sería quedarse con 3 clústeres, lo cual coincide con el número de clases existentes en nuestro dataset. Una vez hemos decidido el número de clusters, graficaremos los resultados obtenidos tras aplicar el algoritmo. En este caso, al disponer de datos etiquetados previamente, es interesante comparar la clasificación realizada por el algoritmo con los valores reales. En este caso, podemos ver que clasifica casi todas las instancias correctamente.

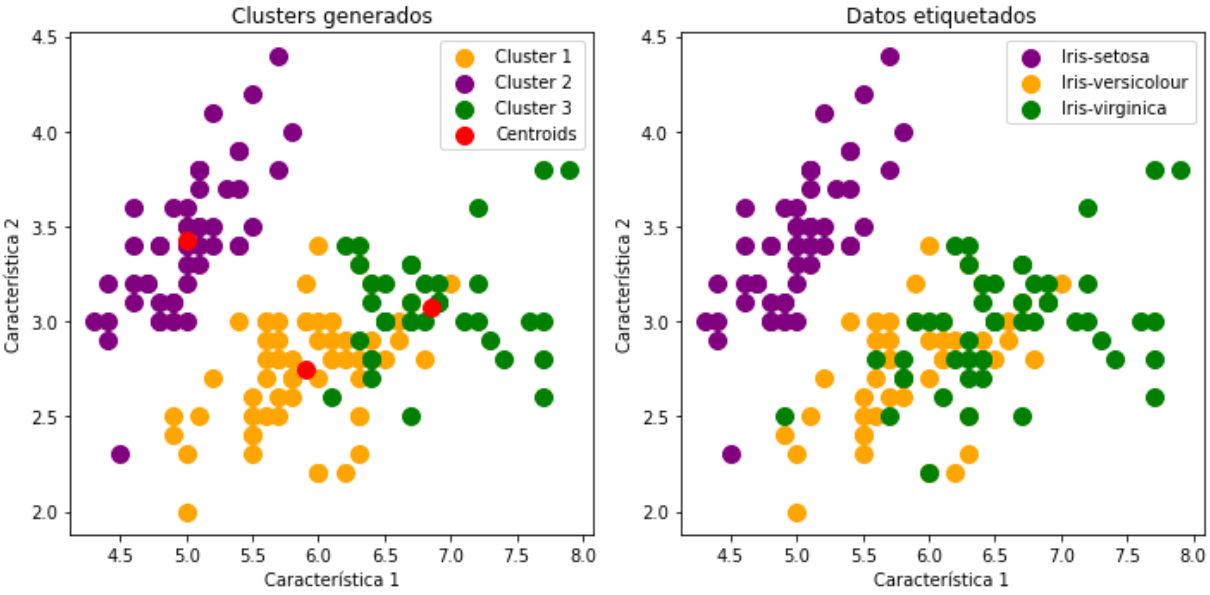

Figura 18: Comparación entre los k=3 clústers generados, y los datos etiquetados en la base de datos original, segmentado por la variable de clasificación. Fuente: Elaboración propia.

```
Coeficiente de silueta: 0.5528190123564095
```

El coeficiente de silueta nos proporciona una medida de cuán bien se separan las muestras dentro de los clusters y qué tan similares son las muestras de un cluster en comparación con los clusters vecinos. Un valor de coeficiente de silueta cercano a 1 indica una buena separación de los clusters, mientras que un valor cercano a -1 indica una mala separación. En este caso, al ser el valor más próximo a 1 (0.5528), lo consideramos un buen resultado.

El coeficiente de silueta se calcula utilizando la distancia media dentro del grupo (a) y la distancia media del grupo más cercano (b) para cada muestra xi del conjunto X.

Para cada xi se calcula ai como la distancia media al resto de puntos de su grupo. Y se calcula bi como la distancia media al siguiente grupo más cercano. Siendo el valor de la silueta en xi, si el grupo al que pertenece tiene más de 1 punto.

$$s_i = \frac{b_i - a_i}{max\{a_i, b_i\}}$$

Una vez hemos estudiado como se agrupan los datos, nuestro siguiente objetivo será estudiar varios algoritmos de reducción de la dimensión. En el caso nuestro objetivo principal será explicar el mayor porcentaje de varianza posible con cada uno de ellos, reduciendo la dimensión de la matriz original.

Comenzamos nuestro estudio realizando un PCA. En este caso, parece que la opción más lógica es quedase con 2 componentes principales, los cuales consiguen explicar más de un 95% de la

varianza, siendo el primero de los componentes el que aglutina la mayor cantidad de esta misma, siendo este valor mayor a un 75%.

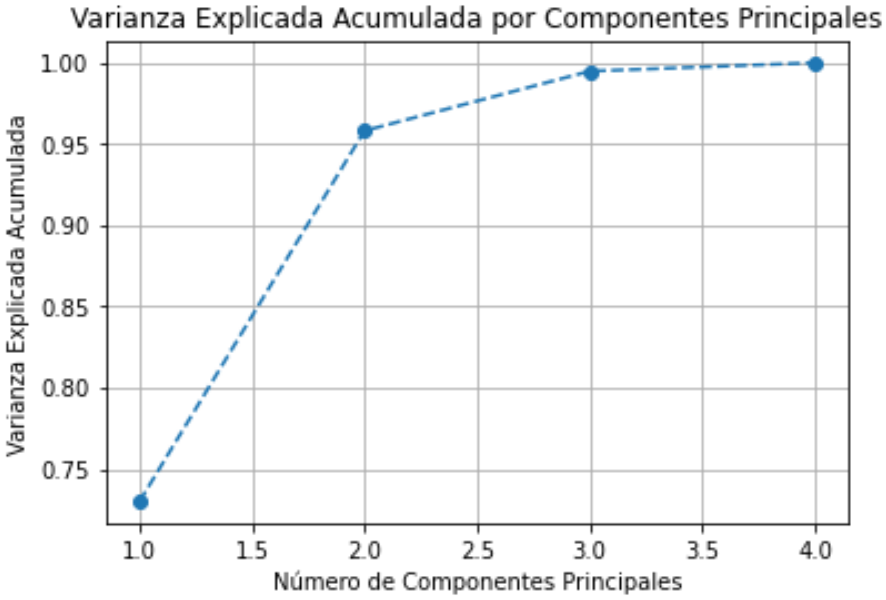

Figura 19: Varianza explicada acumulada según el número de componentes principales retenidos. Fuente: Elaboración propia.

```
Varianza explicada por cada componente principal:
Componente Principal 1: 0.7296
Componente Principal 2: 0.2285
Componente Principal 3: 0.0367
Componente Principal 4: 0.0052
```

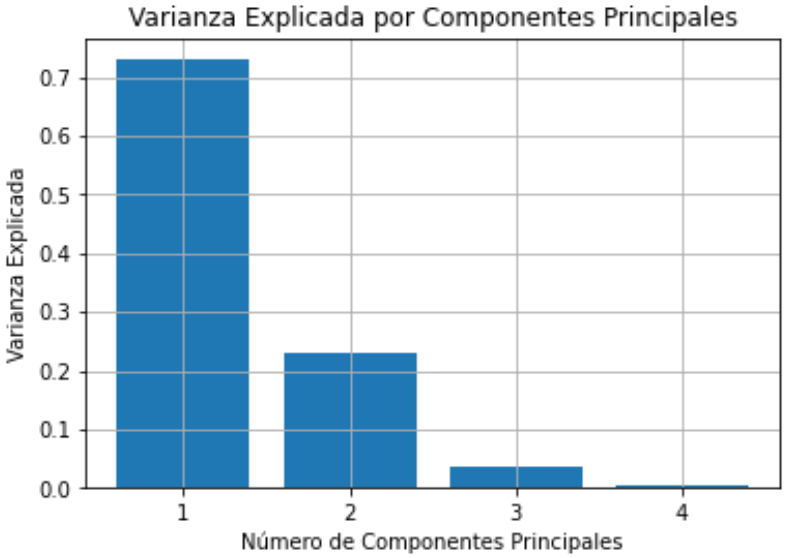

Figura 20: Varianza explicada por cada uno de los componentes principales. Fuente: Elaboración propia.

Podemos realizar una representación gráfica para ver cómo se distribuyen nuestros datos en un nuevo espacio 2-dimensional, conformado por dichas componentes principales. Segmentaremos además nuestros resultados clasificándolos además según el tipo de flore al que pertenezcan.

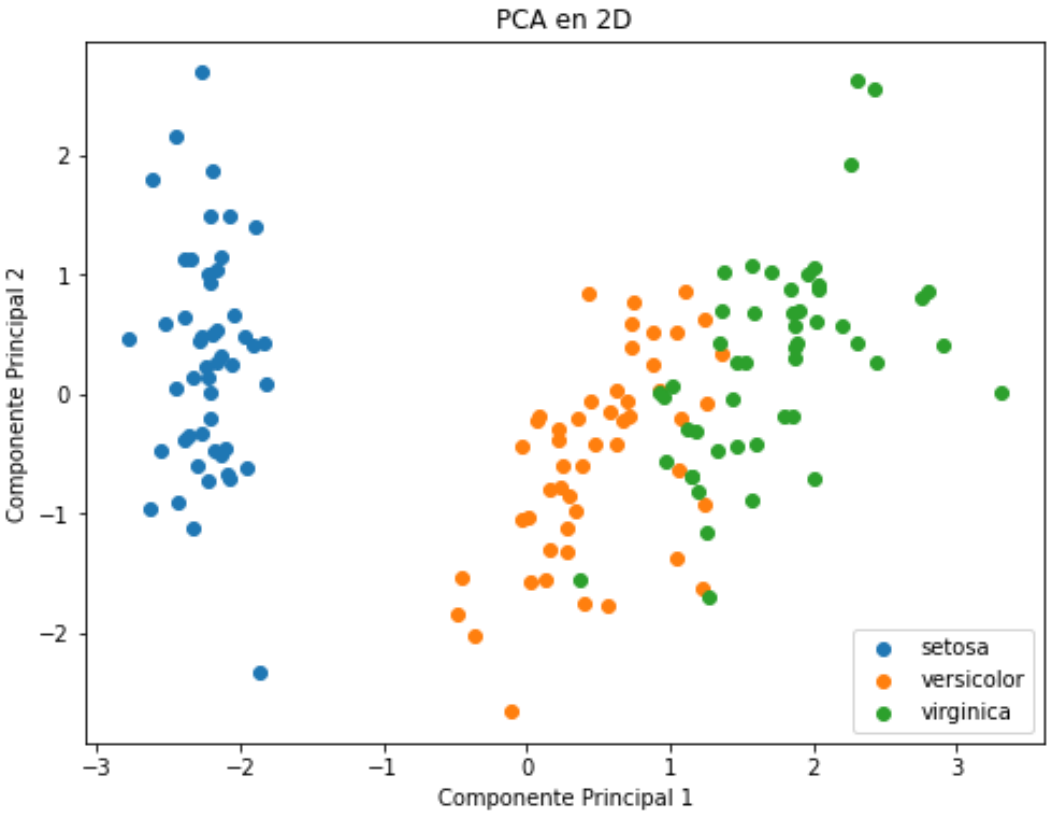

Figura 21: PCA en 2 dimensiones, segmentado por tipo de flor. Fuente: Elaboración propia.

Parece que la clase setosa obtiene puntuaciones negativas en el primer componente, al contrario que las versicolor y virginica que se caracterizan por puntuar positivamente en el mismo.

Procedemos a realizar ahora una representación en 3D. Es importante destacar que, aumentando una dimensión más, solo explicamos un 3,6% de varianza más, por lo que puede no ser la opción más recomendable.

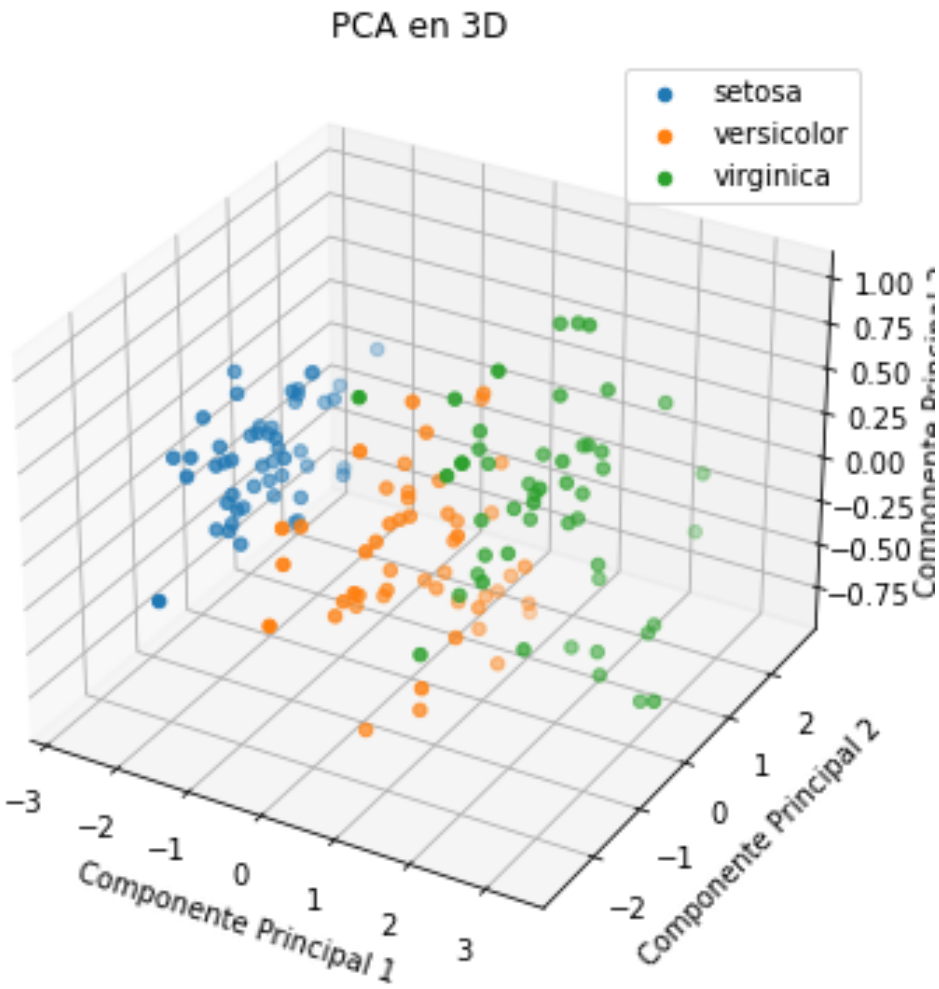

Figura 22: PCA en 3 dimensiones, segmentado por tipo de flor. Fuente: Elaboración propia.

Es interesante estudiar la relación entre el PCA y el SVD. Sabemos que los vectores y los valores propios de una matriz de covarianza representan el núcleo de un ACP. Por un lado, los vectores propios (componentes principales) determinan las direcciones del nuevo espacio de características, y los valores propios determinan su magnitud. En otras palabras, los valores propios explican la varianza de los datos a lo largo de los nuevos ejes de características.

Sabemos que la covarianza entre dos características se calcula de la siguiente manera:

$$Cov(X,Y) = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

Y que:

1. Los valores singulares en Σ obtenidos mediante SVD son en realidad las raíces cuadradas de los autovalores de la matriz de covarianza en PCA.
2. Si centramos nuestra matriz de datos X y luego realizamos un SVD, estamos haciendo lo mismo que un PCA

Por lo tanto:

Consideremos una Matrix $X \in R^{m \times n}$ donde $m > n$ y todo x_{ij} están centrados sobre la media de las columnas, con una matriz de covarianzas C. Con Análisis de Componentes Principales tenemos:

$$C = \frac{1}{m} X^{\intercal} X = \frac{1}{m} V \Gamma V^{\intercal}$$

Pero sabemos que $X = U \Sigma V^{\intercal}$

Por lo que $C = \frac{1}{m} V \Sigma U^{\intercal} U \Sigma V^{\intercal} = \frac{1}{m} V \Sigma^2 V^{\intercal}$

Por ello, aunque la descomposición en autovalores/autovectores de la matriz de covarianza o correlación puede ser más intuitiva, la mayoría de las implementaciones de PCA realizan una descomposición en valores singular (SVD) para mejorar la eficiencia computacional. Por lo tanto, vamos a realizar una SVD para confirmar que el resultado es el mismo.

Comenzaremos comparando los autovalores de la matriz de covarianza en PCA con los valores singulares obtenidos en Σ.

```
Autovalores del PCA:
[634.23625591  36.40061219  11.73142501   3.57526395]

Valores singulares del SVD al cuadrado:
[630.0080142   36.15794144  11.65321551   3.55142885]
```

Figura 23: Autovalores del PCA comparados con valores singulares del SVD al cuadrado. Fuente: Elaboración propia.

Podemos comprobar que en este caso son muy similares y pueden considerarse idénticos.

Ahora, comprobemos si podemos obtener la matriz de covarianza C derivada con PCA, pero utilizando $\frac{1}{m} V \Sigma^2 V^{\intercal}$.

```
Matriz de covarianzas derivada del PCA:
[[ 0.68569351 -0.042434    1.27431544  0.51627069]
 [-0.042434    0.18997942 -0.32965638 -0.12163937]
 [ 1.27431544 -0.32965638  3.11627785  1.2956094 ]
 [ 0.51627069 -0.12163937  1.2956094   0.58100626]]

Matriz de covarianzas usando S y V(t) desde el SVD:
[[ 0.68112222 -0.04215111  1.26582     0.51282889]
 [-0.04215111  0.18871289 -0.32745867 -0.12082844]
 [ 1.26582    -0.32745867  3.09550267  1.286972  ]
 [ 0.51282889 -0.12082844  1.286972    0.57713289]]

Son estas matrices equivalentes?
True
```

Figura 24: Matriz de covarianzas obtenida a través de PCA y SVD. Fuente: Elaboración propia.

Observamos que ambas matrices son iguales. Con estos resultados podemos verificar que se usa el SVD para mejorar la eficiencia computacional de los algoritmos de PCA.

5. Discusión

Interpretación y análisis de los resultados

Los resultados muestran que la mayoría de los algoritmos de clasificación obtienen resultados óptimos, al igual que sucede con las métricas relativas a la calidad de la clusterización o la varianza explicada en las técnicas de reducción de dimensiones. Podemos considerar que hemos conseguido satisfactoriamente nuestro objetivo de explorar las bondades de diversos algoritmos en un dataset real, además de comparar los resultados existentes entre ellos.

Limitaciones y posibles mejoras del estudio

Es importante destacar que los resultados tienen ciertas limitaciones y pueden ser no todo lo concluyentes que nos gustarían. Aunque el dataset Iris es perfecto a nivel didáctico y de usabilidad, tiene un gran inconveniente y es que su número de instancias no es excesivamente alto (150). Que esto sea así, nos ocasiona una serie de problemas, y es que muchos de los resultados estarán muy condicionados en torno a 2 variables de elección personal en el estudio,

las cuales son la semilla utilizada, y el porcentaje de datos que usaremos en nuestros tests de entrenamiento y validación.

Esto quiere decir que, usando otra semilla distinta, o usando otro porcentaje en los grupos de entrenamiento/validación, los algoritmos de clasificación que puntuaron peor en la métrica de clasificación podrían haber sido los mejores en otro contexto, y viceversa.

Por ello, aunque 150 instancias no es necesariamente un número excesivamente pequeño, y ya puede considerarse una muestra representativa, es importante tomar estos resultados con cierta cautela.

6. Conclusiones

Hemos comprobado que el rendimiento de todos los algoritmos de clasificación, en el problema de etiquetado de clases de flor (exceptuando el sigmoide), es significativamente bueno. Todos ellos obtienen una precisión superior al 0.91, siendo la más alta la obtenida con el clasificador SVM, con el kernel Poly, y con el K-vecinos cercanos, con una k=10, en ambos casos de 0,97, lo cual nos indica que prácticamente todas las instancias fueron clasificadas correctamente.

En lo referente a los algoritmos de clustering no jerárquico, aplicando k-medias hemos comprobado que la opción óptima a la hora de segmentar nuestros datos es crear 3 grupos, lo cual parece razonable ya que coincide con las clases de flor existente. Al disponer de los datos etiquetados originales los hemos comparado con los clusters generados por el algoritmo, obteniendo unos resultados extremadamente similares. Hemos obtenido un coeficiente de silueta de 0.5528, lo cual nos indica que las muestras están razonablemente bien agrupadas.

En relación con los algoritmos de clúster jerárquico, hemos podido observar que la elección de un tipo de enlace u otro cambia en gran manera la forma en la que nuestras instancias son organizadas a través de dendogramas.

Con respecto a las técnicas de reducción de la dimensión, hemos aplicado un Análisis de Componentes Principales, obteniendo unos resultados más que satisfactorios. Reteniendo un solo componente hemos podido conseguir explicar un 72.96% de la varianza, y añadiendo un segundo conseguíamos llegar a un más que óptimo 95,81%, por lo que nuestra decisión ha sido retener finamente 2 componentes.

7. Bibliografía

- Bengio, Y. C. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence., 35*(8), 1798-1828.
- Blogdatas. (s.f.). *Algoritmos Supervisados: Clasificación vs. Regresión*. Obtenido de https://blogdatlas.wordpress.com/2020/06/28/algoritmos-supervisados-clasificacion-vs-regresion-datlas-research/
- *Carnegie Mellon University*. (s.f.). Obtenido de https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15750-s20/www/notebooks/SVD-irises-clustering.html
- Fabrigar, L. R. (1999). Evaluating the use of exploratory factor analysis in psychological research. *Psychological Methods, 4*(3), 272.
- Fawcett, & Provost. (2013). Data Science and its Relationship to Big Data and Data-Driven Decision Making. *Big Data, 1*(1), 51-59.
- Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics, 7*(2), 179-188.
- Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media Inc.
- Golub, G. H. (2013). *Matrix Computations.* Baltimore: The Johns Hopkins University Press.
- Gorsuch, R. L. (1997). Exploratory factor analysis: Its role in item analysis. *Journal of Personality Assessment, 68*(3), 532-560.
- Hastie, T. T. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Vol. 2). New York: Springer.
- Hinton, G. E. (2006). Reducing the dimensionality of data with neural networks. *Science, 313*(5786), 504-507.
- IBM. (s.f.). *IBM*. Obtenido de https://www.ibm.com/es-es/topics/unsupervised-learning
- Jain, A. K. (1999). Data clustering: A review. *ACM computing surveys (CSUR), 31*(3), 264-323.
- Jollife, I. (2005). Principal component analysis. *Encyclopedia of statistics in behavioral science.*
- Kline, P. (2013). *Handbook of psychological testing.* London: Routledge.
- Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *International Joint Conference on Articial Intelligence, 14*(2), 1137-1145.

- Larose, D. T. (2014). *Discovering knowledge in data: An introduction to data mining* (Vol. 4). New York: John Wiley & Sons.
- LeCun, Y. B. (2015). Deep learning. *Nature*, *521*(7553), 436-444.
- Manning, C. D. (1999). *Foundations of statistical natural language processing.* Cambridge, MA: MIT Press.
- Mitchell. (1997). *Machine Learning.* New York: McGraw-Hill.
- Moreno, G. (2019). A la espera de un Big Bang de datos. *Digital Economy Compass*, 1-230.
- Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, *3*(3), 210-229.
- Telefónica. (2021). *Telefonica Tech*. Obtenido de https://empresas.blogthinkbig.com/que-algoritmo-elegir-en-ml-aprendizaje/

8. Summary

Data science is a set of fundamental principles that support and guide the extraction of information and knowledge from data. ***(Fawcett & Provost, 2013)***

Nowadays, Data Science has acquired great relevance due to the enormous amount of information that is generated every day in different fields. Data analytics has become a fundamental tool for decision making in companies, institutions, and organizations in general.

With the exponential increase of available data, the need to process and analyze them has led to the application of Machine Learning techniques, which has allowed researchers and Data Science professionals to face increasingly complex problems and obtain more accurate results.

The amount of Information available to analyze keeps growing, and this trend is expected to continue to grow over the next few years, so applying novel techniques has become more relevant than ever. (Moreno, 2019)

By making use of their properties and the available computer software, we can therefore simplify existing tasks, and find solutions much more efficiently and quickly.

The main objective of this work is therefore to explore the main Machine Learning techniques applied to Data Science. To do so, we will analyze in detail the theoretical foundations of each of them, and their range of applicability for each type of existing problem.

We will try to discern in which situation it is more or less advisable to use one or the other, the advantages and disadvantages of each of them, and of course discuss their limitations and prospects for the future.

Subsequently, we will apply these algorithms to a real database and analyze the results. Finally, we will draw some conclusions where we will synthesize and evaluate the results obtained.

Beginning with the description of the various sections of our work, the first section will be dedicated to the current state of the art. Within it, the first chapter will be devoted to the definition of several key concepts in this discipline.

We begin by defining what we mean by machine learning, quoting the definition of (Samuel, 1959). *Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed.* We continue with the concept of "learning", in the context of ML, and define what an algorithm is. *A computer program is said to learn from experience E with respect to some task T and some performance measure P if its performance on T, as measured by P, improves with experience E. (Mitchell, 1997)*

We follow this section by mathematically defining elements as transversal to all statistics as hypothesis, model or dataset. Finally, we explain more specific concepts of machine learning, such as the difference between training and validation data, testing, or deep learning.

In the next section of the chapter dedicated to the state of the art, we talk about the history of Machine Learning and its most relevant characters. We begin by discussing its origins in the 1950s with Samuel, and we progressively advance through the decades, mentioning all the relevant figures in this history, finally reaching its current state.

In this section we mention several of the most recognized software and applications that make use of these techniques, which are usually of great complexity due to the use of Deep Learning tools for their construction. In this chapter we also explore one of the most fashionable tools nowadays, the Large Language Models (LLM), and we mention some of its main exponents.

Once the section dedicated to the State of the Art is finished, we start with the theoretical part where we study in depth the algorithms that will be part of our study.

The first step will be to explain what kind of problems these algorithms try to solve. To do so, we introduce and define the concepts of regression and classification, which will be a fundamental part of our later results.

The task of classification is to predict a discrete variable, known as a response variable or class label, from a set of input or predictor variables On the other hand, in regression, the goal is to predict a continuous response variable, such as a real output variable. (Hastie, 2009)

We also define other common problems in the field of Machine Learning, such as Clustering, Dimension Reduction or Natural Language Processing (NLP), the latter technique being of great relevance nowadays due to its close relationship with the aforementioned Large Language Models. Finally, we mention several areas of application of these problems, and in which sectors of society one or the other are particularly relevant.

Once the definition of the problems to be solved has been completed, we move on to the section where we study the algorithms that attempt to attack these problems. We divide our study into two sections, the first focusing on algorithms that use supervised learning techniques, and the second on those that use unsupervised learning techniques.

In the case of supervised learning, we begin by defining through a quote from (Mitchell, 1997) what exactly we mean by this set of techniques.

Unsupervised learning is a type of machine learning that aims to find patterns in data without the help of labeled examples. Rather than trying to predict a specific output, the goal of unsupervised learning is to discover structures in the data and group them into similar categories or classes.

Subsequently, we show a series of examples where they are used, what kind of problems they usually try to solve, and several use cases for these techniques.

Once the basics of supervised learning have been explained, we begin to describe its most relevant algorithms.

We begin with least squares regression. In this case, in addition to explaining what type of problems it tries to solve, we will mathematically define the components of this, i.e., the elements that make up the formula:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n +$$

In addition, we explain the elements that compose the cost function, which we try to minimize.

$$\text{Cost}(\beta_0, \beta_1, \beta_2, \dots, \beta_n) = \sum_{i=1}^{m} \left(y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_n x_{in})\right)^2$$

Next, we explain the theoretical foundations of our next classifier, Logistic Regression, in addition to studying its range of applicability, and we analyze the components of the sigmoid function that is transversal to it.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

We continue with our next classifier, the decision trees. In this case, we define the iterative process necessary to build it. We then focus on explaining the logical process that must be followed to interpret it, and to be able to extrapolate its use to new, unfamiliar instances.

Next, we analyze the algorithm known as Support Vector Machines (SVM). In this case, in addition to explaining its most common use cases, we introduce the concept of the optimal hyperplane of separation between classes, which, if linear, is given by the following equation:

$$w \cdot x + b = 0$$

Subsequently, we explain concepts associated with it, such as its kernels.

The next algorithm we talk about will be the Naive Bayes classifier. In this case, in addition to explaining its particularities and assumptions (it assumes that the features are independent of each other), we focus especially on explaining its underlying idea, Bayes' theorem, which is given by the formula.

$$P(C_k | x_1, x_2, \dots, x_n) = \frac{P(C_k) \cdot P(x_1, x_2, \dots, x_n | C_k)}{P(x_1, x_2, \dots, x_n)}$$

Finally, we analyze the Ensemble methods. We explain the different methodologies used in them, and we discuss the differences between their 2 most important approaches, model averaging and voting ensemble. Subsequently, we analyze the most common resampling techniques used in these approaches, such as Random Forest or Gradient Boosting.

Once the explanation of the different types of algorithms is concluded, we include a section dedicated to discussing the advantages and disadvantages that the use of this type of unsupervised learning techniques may have. Subsequently, we dedicate a section to mention the challenges they face for the future.

Having concluded the section dedicated to supervised learning, we move on to unsupervised learning. We start by defining it.

Unsupervised learning is a type of machine learning that aims to find patterns in data without the help of labeled examples. Instead of trying to predict a specific output, the goal of unsupervised learning is to discover structures in data and group them into similar categories or classes. *(Bengio, 2013)*

We can distinguish 3 main categories around the objective that the algorithms we study in this section of the paper try to achieve: Clustering, Association Rules, and Dimensionality Reduction.

In the case of the chapter dedicated to clustering, we define what is the objective that these algorithms try to achieve, and we explain the differences between the different existing clustering approaches: Exclusive, Hierarchical and Probabilistic.

Starting with the exclusive ones, we highlight the K-means algorithm. In addition to describing its technical fundamentals, we describe the iterative process required to arrive at a solution.

Regarding hierarchical algorithms, we highlight the use of dendograms. We explain their most relevant concepts, and then we define mathematically the concept of similarity, as well as the different types of links, and the existing ways to measure the similarity between the different instances.

The next section of our work will be the association rules. In them, we focus on explaining their most important technique, the a priori algorithm, commonly used in purchase analysis.

We continue with one of the fundamental sections of our study, the dimension reduction techniques. We explained 3 of them, which we used later in our results, but there are several more. Principal Component Analysis, Factor Analysis, and SVD.

In all of them, in addition to mathematically defining their most relevant concepts, (such as the prior assumptions made by each one) we make special emphasis on analyzing the iterative processes required to obtain the final solutions.

In the case of Factor Analysis, we also explain the difference between Exploratory and Confirmatory FA, and in the case of SVD we study its relationship with PCA, with which it is closely interconnected.

Once the theoretical part of our work is concluded, we move on to the results section.

In them, we distinguished 2 clearly differentiated sections. In the first, we applied the supervised learning algorithms to a real dataset (Iris Species), with the goal of testing the

performance of these algorithms in predicting the flower class (i.e., a multiclass classification problem).

In the second section of our results, we apply several types of unsupervised learning algorithms to the previously mentioned database. Regarding the exclusive clustering section, we use K-Means to analyze how our instances are segmented, and in how many groups it is recommended to segment them.

Regarding hierarchical clusters, we perform a series of dendograms to check how our instances are grouped with various linking methods.

Finally, with respect to the dimension reduction techniques we performed a PCA, with the aim of checking what percentage of variance we can explain with a lower number of dimensions, and we studied their relationship with the VDS with concrete examples.

The conclusions obtained were the following:

- We have verified that the performance of all the classification algorithms, in the flower class labeling problem (excepting the sigmoid), is significantly good. All of them obtain an accuracy higher than 0.91, being the highest obtained with the SVM classifier, with the Poly kernel, and with the K-nearest neighbors, with a k=10, in both cases of 0.97, which indicates that practically all the instances were classified correctly.

- Regarding the non-hierarchical clustering algorithms, by applying k-means we have found that the optimal option when segmenting our data is to create 3 groups, which seems reasonable since it coincides with the existing flower classes. Having the original labeled data, we have compared them with the clusters generated by the algorithm, obtaining extremely similar results. We obtained a silhouette coefficient of 0.5528, which indicates that the samples are reasonably well clustered.

- Regarding the hierarchical clustering algorithms, we have been able to observe that the choice of one type of linkage or another greatly changes the way in which our instances are organized through dendograms.

- With respect to dimension reduction techniques, we have applied a Principal Component Analysis, obtaining more than satisfactory results. By retaining only one component we were able to explain 72.96% of the variance, and by adding a second one we were able to reach a more than optimal 95.81%, so our decision was to retain only 2 components.

Finally, and to conclude our work, as far as the discussion is concerned, we can consider that we have satisfactorily achieved our objective of exploring the benefits of different algorithms on a real dataset, as well as comparing the existing results between them.

It is important to note that the results have certain limitations and may not be as conclusive as we would like. Although the Iris dataset is perfect at a didactic and usability level, it has a major drawback and that is that its number of instances is not excessively high (150). That this is so, causes us a series of problems, and that is that many of the results will be very conditioned around 2 variables of personal choice in the study, which are the seed used, and the percentage of data that we will use in our training and validation tests. This means that, using another different seed, or using another percentage in the training/validation groups, the classification algorithms that scored worse on the classification metric might have been the best in another context, and vice-versa. Thus, although 150 instances are not necessarily an excessively small number, and can already be considered a representative sample, it is important to take these results with some caution.

9. Anexo

En esta sección incluiré el código utilizado para obtener los resultados. Aquellos gráficos que hayan sido usados previamente serán omitidos de esta sección, aunque mostraremos igualmente el código utilizado para realizarlos.

Carga de librerías y de la base de datos

Comenzamos cargando todas las librerías necesarias.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
import warnings

from sklearn import svm, metrics
from sklearn.metrics import accuracy_score, confusion_matrix, silhouette_score
from sklearn.cluster import KMeans
from sklearn.decomposition import TruncatedSVD
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D
from factor_analyzer import FactorAnalyzer
```

Cargamos la base de datos, la obtendremos de la librería sklearn.

```
iris = load_iris()
```

Convertimos los datos en un dataframe compatible con la librería pandas.

```
iris_data = pd.DataFrame(data=iris['data'], columns=iris['feature_names'])
iris_data['target'] = iris['target']
```

Análisis Exploratorio

Visualizamos las primeras filas del dataset

```
print(iris_data.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
```

Mostramos los descriptivos

```
print(iris_data.describe())
```

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)      target
count        150.000000  150.000000
mean           1.199333    1.000000
std            0.762238    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

Mostramos un gráfico de resumen con todas las distribuciones y los gráficos de dispersión, segmentados por target, siendo 0 = "Iris Setosa", 1 = Iris Versicolor", 2 = "Iris Virginica".

```
sns.pairplot(iris_data, hue='target')
plt.show()
```

Aprendizaje Supervisado

Comenzaremos aplicando los algoritmos de aprendizaje supervisado que hemos explicado a lo largo del trabajo. Primero, dividiremos nuestro dataset en 2 grupos, X e Y, que contendrán a las variables de características (X) y de target (Y).

```
X = iris.data
y = iris.target
```

Creamos nuestros datasets de entrenamiento y validación.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=60)
```

Usaremos una proporción de 70/30 para nuestros conjuntos de entrenamiento y validación respectivamente. Establecemos una semilla = 987 para poder replicar los resultados.

SVM

El primero de los algoritmos que estudiaremos será el SVM (Support-Vector-Machines). Comenzaremos creando el clasificador y entrenándolo con los datos que hemos seleccionado. Dentro del clasificador, estudiaremos el rendimiento de diversos kernels: Linear, Poly, RBF, y "Sigmoid". En este caso, usaremos las 4 características

```
# Crear el clasificador SVM con diferentes kernels
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
accuracies = []
confusion_matrices = []

for kernel in kernels:
    # Entrenar el clasificador SVM
    svm = SVC(kernel=kernel)
    svm.fit(X_train, y_train)

    # Realizar predicciones en el conjunto de prueba
    y_pred1 = svm.predict(X_test)

    # Calcular la precisión del clasificador
    accuracy = accuracy_score(y_test, y_pred1)
    accuracies.append(accuracy)

    # Calcular la matriz de confusión
    cm = confusion_matrix(y_test, y_pred1)
    confusion_matrices.append(cm)



# Creacion de variables
accuracy1a = accuracies[0]
accuracy1b = accuracies[1]
accuracy1c = accuracies[2]
accuracy1d = accuracies[3]
```

Creamos un gráfico de barras para mostrar las precisiones de cada kernel

```
# Crear un gráfico de barras para mostrar las precisiones de cada kernel
plt.bar(kernels, accuracies)
plt.xlabel('Kernel')
plt.ylabel('Precisión')
plt.title('Precisión del SVM con diferentes kernels')

# Mostrar el valor de cada barra
for i, acc in enumerate(accuracies):
    plt.text(i, acc, f'{acc:.2f}', ha='center', va='bottom')

plt.ylim([0, 1])
plt.show()
```

Graficamos la matriz de confusión con los 4 kernels.

```
#Graficamos la matriz de confusión

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 8))
for i, (cm, kernel) in enumerate(zip(confusion_matrices, kernels)):
    ax = axes.flatten()[i]
    sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', ax=ax)

    ax.set_title(f'Matriz de Confusión - {kernel} Kernel')
    ax.set_xlabel('Etiqueta Predicha')
    ax.set_ylabel('Etiqueta Verdadera')

plt.tight_layout()
plt.show()
```

Ahora, repetiremos el proceso, pero en este caso tomaremos solamente 2 características dentro de la base de base de datos a la hora de construir nuestro SVM (Sepal Length y Sepal Width), con el objetivo de poder representar los resultados gráficamente.

```
# Cambiar los nombres de las variables
X_mod = iris.data[:, :2]  # Tomar solo las dos primeras características para facilitar la visualización

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train_mod, X_test_mod, y_train_mod, y_test_mod = train_test_split(X_mod, y, test_size=0.2, random_state=987)

# Crear el clasificador SVM con diferentes kernels
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
accuracies = []

# Configurar la malla para visualizar las regiones de decisión
h = 0.02  # Tamaño de paso de la malla
x_min, x_max = X_mod[:, 0].min() - 1, X_mod[:, 0].max() + 1
y_min, y_max = X_mod[:, 1].min() - 1, X_mod[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Crear una figura y subgráficos 2x2
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 8))

# Entrenar y visualizar el clasificador SVM para cada kernel
for ax, kernel in zip(axes.flatten(), kernels):
    # Entrenar el clasificador SVM
    svm = SVC(kernel=kernel)
    svm.fit(X_train_mod, y_train_mod)

    # Realizar predicciones en la malla
    Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Visualizar las regiones de decisión
    ax.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu, alpha=0.8)

    # Visualizar los puntos de entrenamiento
    ax.scatter(X_train_mod[:, 0], X_train_mod[:, 1], c=y_train_mod, cmap=plt.cm.RdYlBu, edgecolors='k')
    ax.set_xlabel('Sepal Length')
    ax.set_ylabel('Sepal Width')
    ax.set_title(f'SVM con kernel {kernel}')

    # Calcular y mostrar la precisión del clasificador
    y_pred_mod = svm.predict(X_test_mod)
    accuracy_mod = accuracy_score(y_test_mod, y_pred_mod)
    ax.text(x_min + 0.1, y_max - 0.2, f'Precisión: {accuracy_mod:.2f}', color='white')

# Ajustar los márgenes y mostrar los gráficos
plt.tight_layout()
plt.show()
```

Regresión logística

El siguiente clasificador que estudiaremos será la regresión logística.

```
# Crear y ajustar el modelo de regresión logística
modelo2 = LogisticRegression()
modelo2.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred2 = modelo2.predict(X_test)

# Calcular la precisión del modelo
accuracy2 = accuracy_score(y_test, y_pred2)
print("Precisión:", accuracy2)

# Calcular la matriz de confusión
confusion2 = confusion_matrix(y_test, y_pred2)
```

Graficamos la matriz de confusión

```
fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(confusion2, annot=True, cmap='Blues', fmt='d', ax=ax)
ax.set_title('Matriz de Confusión - Regresión Logística')
ax.set_xlabel('Etiqueta Predicha')
ax.set_ylabel('Etiqueta Verdadera')

plt.show()
```

Árboles de decisión

```
# Crear y ajustar el modelo de árbol de decisión
modelo3 = DecisionTreeClassifier()
modelo3.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred3 = modelo3.predict(X_test)

# Calcular la precisión del modelo
accuracy3= accuracy_score(y_test, y_pred3)
print("Precisión:", accuracy3)

# Calcular la matriz de confusión
confusion3 = confusion_matrix(y_test, y_pred3)
```

Graficamos la matriz de confusión

```
labels = iris.target_names
sns.heatmap(confusion3, annot=True, cmap='Blues', fmt='d', xticklabels=labels, yticklabels=labels)
plt.xlabel('Etiqueta Predicha')
plt.ylabel('Etiqueta Verdadera')
plt.title('Matriz de Confusión - Árbol de Decisión')
plt.show()
```

Graficamos el árbol de decisión

```
plt.figure(figsize=(12, 8))
plot_tree(modelo3, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()
```

Métodos Ensemble

El método RandomForest combina varios clasificadores y obtiene una puntuación ponderada entre ellos.

```
# Crear y ajustar el clasificador de ensamble Random Forest
modelo4 = RandomForestClassifier()
modelo4.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred4 = modelo4.predict(X_test)

# Calcular la precisión del clasificador
accuracy4 = accuracy_score(y_test, y_pred4)
print("Precisión:", accuracy4)

# Calcular la matriz de confusión
confusion4 = confusion_matrix(y_test, y_pred4)

# Visualizar la matriz de confusión
labels = iris.target_names
sns.heatmap(confusion4, annot=True, cmap='Blues', fmt='d', xticklabels=labels, yticklabels=labels)
plt.xlabel('Etiqueta Predicha')
plt.ylabel('Etiqueta Verdadera')
plt.title('Matriz de Confusión - Random Forest')
plt.show()

# Obtener la importancia de las características
importances = modelo4.feature_importances_
feature_names = iris.feature_names

# Ordenar las importancias de manera descendente
indices = sorted(range(len(importances)), key=lambda i: importances[i], reverse=True)
```

Graficamos la importancia de las características

```
plt.figure()
plt.title("Importancia de las Características - Random Forest")
plt.bar(range(len(importances)), [importances[i] for i in indices], align="center")
plt.xticks(range(len(importances)), [feature_names[i] for i in indices], rotation=90)
plt.xlabel("Características")
plt.ylabel("Importancia")
plt.show()
```

Naive Bayes

```
# Crear y ajustar el clasificador Naive Bayes
modelo5 = GaussianNB()
modelo5.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred5 = modelo5.predict(X_test)

# Calcular la precisión del modelo
accuracy5 = accuracy_score(y_test, y_pred5)
print("Precisión:", accuracy5)

# Calcular la matriz de confusión
confusion5 = confusion_matrix(y_test, y_pred5)
```

Graficamos la matriz de confusión

```
plt.figure(figsize=(8, 6))
sns.heatmap(confusion5, annot=True, cmap='Blues', fmt='d')
plt.title('Matriz de Confusión - Naive Bayes')
plt.xlabel('Etiqueta Predicha')
plt.ylabel('Etiqueta Verdadera')
plt.show()
```

KNN

```
# Crear y ajustar el clasificador k-vecinos cercanos, empezaremos probando con k = 3.
modeloi = KNeighborsClassifier(n_neighbors=3)
modeloi.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred6 = modeloi.predict(X_test)

# Calcular la precisión del modelo
accuracyi = accuracy_score(y_test, y_pred6)
print("Precisión:", accuracyi)

# Calcular la matriz de confusión
confusioni = confusion_matrix(y_test, y_pred6)
```

Graficamos la matriz de confusión

```
plt.figure(figsize=(8, 6))
sns.heatmap(confusioni, annot=True, cmap='Blues', fmt='d')
plt.title('Matriz de Confusión')
plt.xlabel('Etiqueta Predicha')
plt.ylabel('Etiqueta Verdadera')
plt.show()
```

Comprobaremos si con otro valor de k se podría mejorar este resultado.

```
a_index=list(range(1,11))
a=pd.Series()
for i in list(range(1,11)):
    modelo7=KNeighborsClassifier(n_neighbors=i)
    modelo7.fit(X_train,y_train)
    prediction7=modelo7.predict(X_test)
    a=a.append(pd.Series(metrics.accuracy_score(prediction7,y_test)))
plt.plot(a_index, a)
```

El máximo valor obtenido es con k=10, usaremos ese valor en nuestros resultados.

```
# Crear y ajustar el clasificador k-vecinos cercanos, en este caso con 10
modelo6 = KNeighborsClassifier(n_neighbors=10)
modelo6.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred7 = modelo6.predict(X_test)

# Calcular la precisión del modelo
accuracy6 = accuracy_score(y_test, y_pred7)
print("Precisión:", accuracy6)
```

Graficamos la matriz de confusión

```
# Calcular la matriz de confusión
confusion6 = confusion_matrix(y_test, y_pred7)

# Mostrar la matriz de confusión
plt.figure(figsize=(8, 6))
sns.heatmap(confusion6, annot=True, cmap='Blues', fmt='d')
plt.title('Matriz de Confusión - KNN (k =10)')
plt.xlabel('Etiqueta Predicha')
plt.ylabel('Etiqueta Verdadera')
plt.show()
```

Comparación de resultados

Crearemos un dataframe donde incluiremos todos los resultados obtenidos en los clasificadores usados hasta ahora, y veremos cual ha obtenido una mayor precisión.

```
resultados = pd.DataFrame({
    'Modelo': ['SVM Linear',
               'SVM Poly',
               'SVM RBF',
               'SVM Sigmoid',
               'Regresión Logística',
               'Árboles de decisión',
               'Ensemble',
               'Naive Bayes',
               'KNN'],
    'Precisión': [ accuracy1a,
                   accuracy1b,
                   accuracy1c,
                  accuracy1d,
                  accuracy2,
                  accuracy3,
                  accuracy4,
                  accuracy5,
                  accuracy6]})
result_df = resultados.sort_values(by='Precisión', ascending=False)
result_df = result_df.reset_index(drop=True)
result_df.head(9)
```

Realizamos un gráfico de barras para mostrar de forma más visual los resultados.

```
result_df = resultados.sort_values(by='Precisión', ascending=False)
result_df = result_df.reset_index(drop=True)

# Asignar colores a las barras
colors = plt.cm.Set3(np.linspace(0, 1, len(result_df)))

# Ajustar el tamaño de la figura
plt.figure(figsize=(8, 6))  # Ajusta el tamaño a tus preferencias

# Crear el gráfico de barras
plt.bar(result_df['Modelo'], result_df['Precisión'], color=colors)

# Añadir el valor de cada barra encima de ellas
for i, v in enumerate(result_df['Precisión']):
    plt.text(i, v, str(round(v, 2)), ha='center', va='bottom')

plt.xlabel('Modelo')
plt.ylabel('Precisión')
plt.title('Precisión de los modelos de clasificación')
plt.xticks(rotation=90)
plt.tight_layout()



plt.show()
```

Aprendizaje No Supervisado

Clustering

Nuestro objetivo con los algoritmos de clustering será ver como se segmentan automáticamente los datos que disponemos.

K-Means

El algoritmo utilizado para realizar dicha segmentación será k-medias. Primero, con la ayuda del método del codo, elegiremos el número óptimo de clústers.

```
# Función para determinar el número óptimo de clusters utilizando el método del codo
def determine_optimal_clusters(X):
    wcss = []  # within-cluster sums of squares
    for i in range(1, 11):
        kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=785)
        kmeans.fit(X)
        wcss.append(kmeans.inertia_)

    # Graficar la curva de la suma de los cuadrados intra-cluster (WCSS)
    plt.plot(range(1, 11), wcss)
    plt.title('Método del Codo')
    plt.xlabel('Número de clusters')
    plt.ylabel('WCSS')  # within cluster sum of squares
    plt.show()

# Determinar el número óptimo de clusters
determine_optimal_clusters(X)
```

```
# Aplicar el algoritmo de clustering k-means con el número óptimo de clusters
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10, random_state=785)
y_kmeans = kmeans.fit_predict(X)


# Crear el gráfico de 1x2
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# Gráfico de los clusters generados
axes[0].scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s=100, c='orange', label='Cluster 1')
axes[0].scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s=100, c='purple', label='Cluster 2')
axes[0].scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s=100, c='green', label='Cluster 3')
axes[0].scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=100, c='red', label='Centroids'
axes[0].set_title('Clusters generados')
axes[0].set_xlabel('Característica 1')
axes[0].set_ylabel('Característica 2')
axes[0].legend()

# Gráfico de dispersión con datos etiquetados
axes[1].scatter(X[y == 0, 0], X[y == 0, 1], s=100, c='purple', label='Iris-setosa')
axes[1].scatter(X[y == 1, 0], X[y == 1, 1], s=100, c='orange', label='Iris-versicolour')
axes[1].scatter(X[y == 2, 0], X[y == 2, 1], s=100, c='green', label='Iris-virginica')
axes[1].set_title('Datos etiquetados')
axes[1].set_xlabel('Característica 1')
axes[1].set_ylabel('Característica 2')
axes[1].legend()

# Ajustar el diseño del gráfico
plt.tight_layout()

# Mostrar el gráfico
plt.show()
```

En este caso, al disponer de datos etiquetados previamente, es interesante comparar la clasificación realizada por el algoritmo con los valores reales.

```
# Calcular el coeficiente de silueta
silhouette_avg = silhouette_score(X, y_kmeans)
print("Coeficiente de silueta:", silhouette_avg)
```

Dendograma

Realizamos un clúster jerárquico. Para ello, haremos uso de un dendograma con diversos métodos de enlace (ward, average, complete y single), para mostrar sus diferencias.

```
# Calcular las matrices de enlace con diferentes métodos y la métrica euclidiana
methods = ['ward', 'average', 'complete', 'single']
metric = 'euclidean'

fig, axes = plt.subplots(len(methods), 1, figsize=(8, 12))

for i, method in enumerate(methods):
    # Calcular la matriz de enlace
    Z = linkage(X, method=method, metric=metric)

    # Generar el dendrograma correspondiente
    ax = axes[i] if len(methods) > 1 else axes
    dendrogram(Z, ax=ax, color_threshold=Z[-2, 2], labels=iris.target)
    ax.set_title(f'Dendograma. Método: {method}, Métrica: {metric}')
    ax.set_xlabel('Muestras')
    ax.set_ylabel('Distancia')

plt.tight_layout()
plt.show()
```

Reducción de la dimensión

En el caso de los algoritmos de reducción de la dimensión, nuestro objetivo principal será explicar el mayor porcentaje de varianza posible con cada uno de ellos, reduciendo la dimensión de la matriz original.

PCA

```
# Normalizar los datos, especialmente util en PCA
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

# Aplicar PCA
pca = PCA()
X_pca = pca.fit_transform(X_normalized)

# Obtener la varianza explicada por cada componente principal
explained_variance = pca.explained_variance_ratio_
```

```
# Graficar la varianza explicada acumulada
cumulative_variance = np.cumsum(explained_variance)
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance, marker='o', linestyle='--')
plt.xlabel('Número de Componentes Principales')
plt.ylabel('Varianza Explicada Acumulada')
plt.title('Varianza Explicada Acumulada por Componentes Principales')
plt.grid(True)
plt.show()
```

```
# Graficar la varianza explicada por cada componente principal
plt.bar(range(1, len(explained_variance) + 1), explained_variance, align='center')
plt.xlabel('Número de Componentes Principales')
plt.ylabel('Varianza Explicada')
plt.title('Varianza Explicada por Componentes Principales')
plt.xticks(range(1, len(explained_variance) + 1))
plt.grid(True)
plt.show()
```

```
# Graficar los resultados en 2D
plt.figure(figsize=(8, 6))
for target, target_name in enumerate(labels):
    plt.scatter(X_pca[y == target, 0], X_pca[y == target, 1], label=target_name)
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('PCA en 2D')
plt.legend()
plt.grid()
plt.show()
```

```
# Graficar los resultados en 3D
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
for target, target_name in enumerate(labels):
    ax.scatter(X_pca[y == target, 0], X_pca[y == target, 1], X_pca[y == target, 2], label=target_name)
ax.set_xlabel('Componente Principal 1')
ax.set_ylabel('Componente Principal 2')
ax.set_zlabel('Componente Principal 3')
ax.set_title('PCA en 3D')
ax.legend()
plt.show()

# Mostrar la varianza explicada por cada componente principal
print("Varianza explicada por cada componente principal:")
for i, var in enumerate(explained_variance):
    print(f"Componente Principal {i+1}: {var:.4f}")
```

SVD y su relación con PCA

Realizaremos un SVD y calcularemos las matrices que lo componen: X, U, S, V. Posteriormente compararemos los resultados obtenidos en el PCA.

```
#Realizamos el PCA
pca.fit(X)

# Centramos nuestros datos
X1 = X - np.mean(X, axis=0)
U, S, Vt = np.linalg.svd(X1)
```

Comparamos los autovalores derivados del PCA con los valores singulares obtenidos en Σ.

```
Cov_pca = pca.get_covariance()

print('Autovalores del PCA:\n{}\n'.format(np.linalg.eigvals(Cov_pca * X1.shape[0])))

print('Valores singulares del SVD al cuadrado:\n{}'.format(np.square(S)))
```

```
print('Matriz de covarianzas derivada del PCA:\n{}\n'.format(Cov_pca))

Cov_svd = (1. / X1.shape[0]) * Vt.T.dot(np.diag(np.square(S))).dot(Vt)

print('Matriz de covarianzas usando S y V(t) desde el SVD:\n{}\n'.format(Cov_svd))

allclose = np.allclose(Cov_pca, Cov_svd, atol=1e-1)
print('Son estas matrices equivalentes?\n{}'.format(allclose))
```