Universidad Nacional de La Plata
Facultad de Informática

Especialización en Inteligencia de Datos Orientada a Big Data
Trabajo Final de Especialización

CLASIFICACIÓN DE DATOS DESBALANCEADOS

Su aplicación en la predicción de bajas de beneficiarios de un servicio de salud privado

Ing. Martinelli Jonatan Emanuel

Dirección

Dr. Hasperué Waldo

MAYO 2022 - ARGENTINA

Índice de contenido

Capítulo I
Introducción

El Aprendizaje automático es un tema de gran interés en la inteligencia artificial, puesto que se trata de la generalización de comportamientos a partir de muestras conocidas. Hoy en día tiene muchas aplicaciones prácticas como: diagnósticos médicos, detección de fraudes, análisis de mercados, detección de spam en correos, entre otros.

Con frecuencia, en el Aprendizaje automático, y específicamente con los problemas de clasificación binaria, nos encontramos con conjuntos de datos desbalanceados. Generalmente esto se refiere a un problema en el que se tienen dos clases y no se representan por igual. Los algoritmos de Machine Learning, están diseñados para operar en datos de clasificación con un número similar de muestras para cada clase. Si no fuera así, los algoritmos podrían aprender que muy pocas de ellas no son importantes y en base a ello, ignorarlas para alcanzar un buen desempeño. Este hecho puede causar graves problemas en su proceso de generalización de la información, conllevando a obtener un rendimiento predictivo inexacto, sesgado y con precisión engañosa, en particular cuando esas pocas muestras representan la clase minoritaria.

Pero… ¿Qué significa precisamente tener un conjunto de datos desbalanceado? ¿Son frecuentes? ¿Cómo impactan en el desempeño de los algoritmos de clasificación? ¿Por qué sucede? Son algunas de las preguntas que solemos hacernos cuando comenzamos a introducirnos en el universo del Aprendizaje automático y tratamos con algoritmos de clasificación.

Referirse a un conjunto de datos desbalanceado implica que las muestras no se encuentran distribuidas equitativamente entre todas las clases que la componen. Esto significa que alguna de las clases del conjunto de datos está altamente desproporcionada y tiene una mayor cantidad respecto al resto (clase mayoritaria). Por tanto, no están representadas por igual en un problema de clasificación y lo cual es bastante común en la práctica (*Véase la Figura 1.1*). Precisamente, una pequeña diferencia no sería considerable, pero podría observarse un sesgo severo en la distribución de clases con una proporción de 1:10, 1:100 o incluso 1:1000 muestras en la clase minoritaria. Indudablemente este tipo de distribución provocaría efectos adversos en el proceso de clasificación.

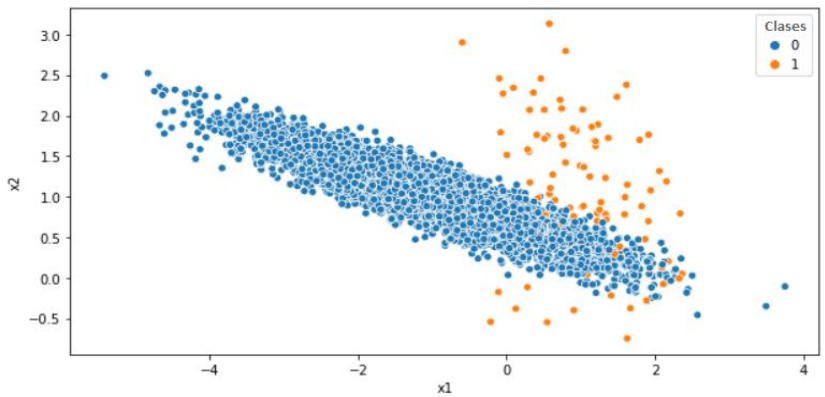


Figura 1.1. *Distribución de clases desbalanceadas.*

Con librerías como scikit-learn a disposición en Python, construir modelos de clasificación es solo cuestión de minutos. No obstante, construirlos sin examinar adecuadamente la estructura de los datos puede conducir a resultados catastróficos.

Véase en un médico. Lamentablemente, ha descubierto un tumor en uno de sus pacientes y se debe investigar si es maligno o no. Para dar un diagnóstico primario, anota algunas características del tumor y las alimenta a su modelo de clasificación de confianza para la predicción. ¿Qué sucede si su modelo predice que el tumor es benigno cuando en realidad es canceroso? El costo de esta predicción incorrecta es enorme: podría retrasar el tratamiento e incluso conducir a la muerte del paciente. Por otro lado, en la detección de fraudes con tarjetas de crédito, las transacciones fraudulentas suelen ser bastantes menores que las transacciones legítimas. Esto es un problema porque la clase minoritaria es exactamente la que más interesa en los problemas de clasificación desbalanceada y el algoritmo no obtiene la información necesaria para hacer una predicción precisa.

Entrenar un modelo predictivo con un conjunto de datos desbalanceado puede hacer que el resultado esté sesgado a favor de la clase mayoritaria, simplemente porque no se cuenta con suficientes datos para aprender sobre la minoría. Los algoritmos de clasificación estándar como Árboles de decisión y Redes neuronales tienen un sesgo hacia las clases que tienen varias muestras. Asumen que el conjunto de datos tiene distribuciones de clases equilibradas. Por lo tanto, tienden a predecir solo los datos de la clase mayoritaria. Las características de la clase minoritaria se tratan como ruido y, con frecuencia, se ignoran. Por lo tanto, existe una alta probabilidad de clasificación errónea de la clase minoritaria en comparación con la clase mayoritaria. Como se puede comenzar a ver, es crucial balancear el conjunto de datos antes de crear un modelo predictivo.

La problemática del desbalance de clases puede ocurrir en clasificación binaria como también en problemas de clasificación de clases múltiples. Cabe mencionar, que existe una gran variedad de métodos que permiten tratar el desbalance de clases y pueden utilizarse en cualquiera de los dos escenarios descritos.

1.1 Objetivo

El objetivo general de esta investigación es realizar un estudio, a partir de un conjunto de datos desbalanceado, de diferentes estrategias de balanceo de clases para medir el poder pronóstico de diversos modelos obtenidos a través de dos técnicas de clasificación: Redes neuronales y Árboles de decisión. Así mismo, en este contexto, se buscará demostrar lo importante que resulta disponer de un conjunto de datos equilibrado para la explotación de información.

Precisamente, se estudiarán diferentes métodos de balanceo de clases los cuales se aplicarán a un conjunto de datos desbalanceado. Dicho conjunto, fue proporcionado por una organización de cobertura de salud y está conformado por datos relevantes de sus afiliados. En base a ello, la clase mayoritaria estará representada por beneficiarios *activos* y la clase minoritaria por las *bajas*. De esta forma, los métodos de balanceo a estudiar y aplicar cumplen un rol muy importante para mejorar las condiciones del conjunto de datos original.

1.2 Definición del problema

El principal interés es la aplicación de métodos de balanceo de clases para optimizar, en la medida de lo posible, los resultados de clasificación y establecer la efectividad de estos algoritmos a través de diferentes enfoques. Basándose en ello, se

buscará que los modelos sean capaces de predecir la clase correspondiente de los beneficiarios. En consecuencia, en este trabajo se considerará el caso de un conjunto de datos que solamente tiene dos clases (“Activo” y “Baja”) y una de ellas cuenta con una mayor cantidad de muestras que la otra.

Por otra parte, las técnicas de balanceo de clases que estarán bajo estudio son: *Submuestreo (Undersampling)*, *Sobremuestreo (Oversampling)*, *SMOTE* y *Autoencoders*. Donde cada una de ellas dará lugar a diferentes versiones del conjunto de datos original para la misma representación; luego, para cada variante del set de datos, se aplicarán las mismas técnicas predictivas bajo el proceso de *Descubrimiento de Conocimiento en Base de Datos (KDD)*.

En referencia a los algoritmos de clasificación, se empleará una Red neuronal feedforward como *Backpropagation* y *Random Forest* para el caso de Árboles de decisión. Con esto se busca obtener modelos que puedan ser contrastados y construir un cuadro comparativo que permita visualizar los rendimientos e impactos de las diversas técnicas de balanceo, en base a su *Precision*, *Recall*, *Accuracy* y *F1-Measure*. Del mismo modo, se considerará el caso de la base de datos original y desbalanceada para la comparación de modelos.

1.3 Antecedentes

Los nuevos cambios tecnológicos en cuanto a hardware y software obligan a las organizaciones a adaptarse continuamente a nuevas condiciones del entorno, a estar en constante actualización y/o a cambiar sus procesos para incrementar sus capacidades tecnológicas, las cuales le permitan mejorar el rendimiento y tratamiento de los datos. Por otra parte, debido al aumento del volumen de los datos almacenados los directivos se enfrentan a un ambiente de incertidumbre y complejidad creciente. Generalmente no se cuenta con las herramientas necesarias para manipular estos datos y convertirlos en información útil. Tal es así, que ciertas organizaciones han establecido sistemas de inteligencia de negocio para proporcionar a sus trabajadores herramientas que asistan a la toma de decisiones y logren un mayor grado de competitividad.

Actualmente, los desarrollos más recientes de *Inteligencia Artificial* han resaltado la importancia de los sistemas de soporte de decisión en actividades empresariales, sugiriendo modelos desarrollados para asistir al tomador de resolución de problemas. Por lo tanto, el uso de herramientas de Aprendizaje automático se ha incrementado en los últimos años, especialmente en aplicaciones del mundo real. Sin embargo, es claro que los datos reales no son perfectos, ya que pueden contener inconsistencias y valores faltantes, por lo que, dichos algoritmos deben ser sólidos (generalmente producir buenos resultados) para enfrentar las imperfecciones de los datos, y ser capaces de extraer patrones realmente fuertes. No obstante, algunos de ellos que han sido considerados previamente como robustos no han mostrado tener un buen desempeño en ciertas aplicaciones del mundo real (Prodomidis y Stolfo, 1999; Bojarczuck, Lopes, Freitas y Michalkiewicz, 2004; Guo & Viktor, 2004). Una de las causas de este problema es que muchas de estas aplicaciones presentan un problema adicional: clases no balanceadas. Aplicaciones tales como detección de fraudes, detección de intrusos en la red y tareas de diagnóstico/pronóstico médico, exhiben el problema de clases no balanceadas donde existe una clase mayoritaria con muestras normales y una clase minoritaria con muestras anormales o importantes, los cuales generalmente tienen el mayor costo de clasificación errónea (Chan et al., 1999).

El principal problema que presentan actualmente los algoritmos de Aprendizaje automático, cuando son utilizados en tareas de clasificación con conjuntos de datos no

balanceados, es el bajo desempeño alcanzado para clasificar correctamente muestras de la clase minoritaria. La mayoría de los trabajos previos que han tomado en cuenta el problema de clases no balanceadas en aplicaciones del mundo real, se han enfocado en evaluar el desempeño de los clasificadores, exclusivamente en términos de su capacidad para minimizar los errores de clasificación (López Pineda 2018; Mera & Arrieta Ramos, 2015). Dichos clasificadores han demostrado un buen desempeño en conjuntos de datos balanceados, sin embargo, no resultó de la misma manera con datos desequilibrados. Por esta razón, los primeros enfoques dirigidos a resolver el problema de clases no balanceadas ha sido tratar de equilibrar la distribución.

Uno de los trabajos realizados (Nathalie Japkowicz, 2000), trata sobre el inconveniente de desbalance de clases en problemas de aprendizaje. Se abordan dos preocupaciones. En primera instancia, se busca establecer un modelo sistemático para relacionar tipos específicos de desbalance con el grado de insuficiencia de los clasificadores tradicionales. El segundo objetivo es establecer una comparación de los métodos propuestos hasta ese momento para remediar este problema. El propósito de dicha investigación es demostrar experimentalmente que el desbalance de clases obstaculiza el rendimiento de un clasificador estándar, y también se busca comparar el desempeño de diferentes enfoques. El artículo concluye con la idea de que los perceptrones multicapa tienen un mayor grado de afectación al problema de desbalanceo de clases conforme la complejidad se va incrementando, mientras que el tamaño del problema parece no ser un factor determinante; como también determina que tanto el *Submuestreo* como *Sobremuestreo*, dos técnicas de balanceo que veremos en el capítulo III, son métodos muy efectivos para tratar con el problema.

Otro trabajo realizado (Gustavo Batista, Ronaldo Prati y Maria Carolina Monard, 2004), analiza el comportamiento de diferentes métodos de balanceo de datos de entrenamiento, considerando diversos aspectos. Se lleva a cabo una experimentación amplia que involucra 10 métodos de balanceo, donde 3 de ellos son propuestos por los autores de este trabajo; que son implementados en 13 conjuntos de datos del repositorio de la Universidad de California en Irvine UCI. Los experimentos realizados demuestran que el desbalance de clases está relacionado con la superposición de clases, así como el grado de desbalance y complejidad del conjunto de datos. Los autores desarrollan un estudio sistemático del problema de desbalance, recurriendo a diversos métodos en la literatura con distintos grados de desbalance y tamaño del conjunto de datos. Estos son clasificados usando el Árbol de decisión C4.5 y comparando los resultados entre cada uno de los métodos de balanceo utilizados. Sus resultados muestran una mejoría en la clasificación general cuando existen pocos elementos de la clase minoritaria al realizar proceso de *Sobremuestreo*, utilizando los métodos de *SMOTE* con la función de elementos cercanos con *ENN* y *Tomek*. También se detalla que cuando existe una gran cantidad de elementos positivos los métodos aleatorios tienen cierta efectividad dado su bajo costo computacional.

Se puede observar al navegar la web que existen numerosos casos de éxitos en que las diversas técnicas de balanceo han mejorado los resultados de la clasificación binaria o múltiple. En la literatura existen varios trabajos que demuestran que es factible aplicar algoritmos de balanceo de clases en el preprocesamiento de datos y que permiten obtener mejores resultados sin alterar las características del proceso de aprendizaje.

Capítulo II
Aprendizaje automático

El *"Aprendizaje automático"* es definido como un conjunto de métodos que permiten, utilizando muestras de datos, detectar automáticamente patrones tan generales como sea posible. De esta forma, proporcionan información sin intervención o asistencia humana y ayudan a predecir datos futuros.

Entre las aplicaciones más utilizadas del Aprendizaje automático se destacan la clasificación de páginas web, el filtrado de correo electrónico no deseado, la clasificación de imágenes, el reconocimiento de escritura a mano y de rostros, la detección de fraudes, la traducción automática de documentos, la segmentación de clientes y análisis de carrito de mercado, el procesamiento del lenguaje natural, predicción de valores de acciones o temperatura, los sistemas de recomendación (como por ejemplo el de Netflix o Amazon), entre otros (Murphy, 2012).

2.1 Conceptos preliminares

El Aprendizaje automático se divide en tres categorías: *Aprendizaje no supervisado*, *Aprendizaje supervisado* y *Aprendizaje por refuerzo*. Las dos primeras son las más tradicionales y serán explicadas a continuación.

2.1.1 Aprendizaje no supervisado

Este proceso se denomina como no supervisado porque no existe un agente, o supervisor, que determine si los resultados son evaluados en la dirección correcta. El objetivo es encontrar una estructura en los datos de entrada, tal que, se identifiquen patrones que ocurren con mayor frecuencia que otros y buscar establecer una estimación de los datos de salida, es decir, no se recibe ninguna información por parte del entorno que le indique si la salida generada en respuesta a una determinada entrada es o no correcta.

Un método que utiliza este tipo de aprendizaje es el *"Agrupamiento"* o *"Clustering",* donde se intenta establecer conjuntos de datos de forma tal que se puedan definir relaciones entre diferentes elementos. Un modelo de agrupamiento trata de crear grupos homogéneos con máxima heterogeneidad entre ellos para encontrar las características, regularidades o categorías que se puedan establecer entre los datos. Cada muestra con características similares se ubica en el mismo grupo mientras que aquellos con atributos diferentes se ubican en grupos desemejantes. Este principio permite identificar anomalías o información implícita que puede resultar muy útil para la toma de decisiones. Así mismo, está asociado a la construcción de un *modelo descriptivo*.

2.1.2 Aprendizaje supervisado

En el *Aprendizaje supervisado* hay un *"profesor"* o supervisor que controla el proceso de aprendizaje. Para llevarlo adelante, el método aprende una función de datos de entrenamiento que generaliza a partir de haber considerado un número determinado de muestras históricas. Las mismas, están constituidas por dos partes: datos de entrada o atributos y datos de salida o clases. Este tipo de aprendizaje es

muy útil para la clasificación de patrones y para la aproximación de funciones. Así mismo, está asociado a la construcción de un *modelo predictivo.*

Cabe señalar que la salida de una función puede ser un valor continuo, *Regresión*, o predecir el valor de la clase de un objeto de entrada, *Clasificación*. A continuación, se explicará brevemente este último concepto puesto que está relacionado directamente con el presente trabajo.

2.1.3 Clasificación

En la *"Clasificación",* el objetivo principal es predecir la clase a la que pertenece una muestra desconocida**,** por lo cual, es muy importante el proceso de entrenamiento. Cada muestra corresponde a una etiqueta, la cual se indica mediante el valor de un atributo que llamamos *"clase de la instancia"*. Este atributo puede tomar diferentes valores discretos y el resto de ellos (los relevantes a la clase) se utilizan para predecir la misma.

Para entrenar y probar un modelo de clasificación es necesario separar las muestras en dos conjuntos: el conjunto de entrenamiento (del inglés *training set*) y el conjunto de prueba (*test set*). El de entrenamiento es un grupo de muestras que son utilizadas por el clasificador para generar el modelo de solución. El conjunto de validación se emplea para establecer el grado de certeza que tiene dicho modelo, haciendo que el clasificador determine la clase a la cual pertenece cada muestra y luego comparándola con la que realmente pertenece. Esta separación es necesaria para garantizar que la validación de la precisión del modelo es una medida independiente. Si no se usan conjuntos diferentes de entrenamiento y prueba, la precisión del modelo será sobreestimada, es decir, tendremos estimaciones muy optimistas y usualmente se le llama *"sobreajuste" (del inglés "overfitting").*

El método de evaluación para realizar los testeos en este trabajo será la "*validación simple"***.** Consiste en reservar un porcentaje del conjunto de datos como prueba, donde suele variar entre el 5% y 50%. El resto es utilizado para construir el modelo. Por consiguiente, dicha división debe ser aleatoria para que la estimación sea correcta.

Por último, los problemas de clasificación se pueden resolver utilizando diferentes técnicas, las más habituales son Árboles de decisión, en particular la técnica de *Random Forest*, y *Redes neuronales feedforward*. Por esta razón, las mismas serán estudiadas a lo largo de este capítulo y luego aplicadas en un trabajo experimental en el capítulo V.

2.1.4 Subajuste y Sobreajuste

El *"Subajuste"* y *"Sobreajuste"* ocurren cuando la información predictiva de las muestras se usa de manera poco apropiada para el objetivo final del Aprendizaje automático, que es precisamente la generalización del aprendizaje para predecir conjuntos de datos que no fueron vistos en la etapa de entrenamiento.

- El "*Sobreajuste" u "Overfitting"* ocurre cuando el modelo aprende las particularidades de los datos de entrenamiento, pero es incapaz de generalizar a muestras no vistas. Esto se refleja con frecuencia cuando un modelo funciona muy bien con los datos de entrenamiento, pero tiene un bajo rendimiento en el conjunto de datos de validación.

- El "*Subajuste"* ocurre cuando el modelo es incapaz de funcionar bien con los datos de entrenamiento o de generalización. Esto ocurre cuando las

características individuales de las muestras se agrupan en exceso y se les da poca importancia. Un modelo poco ajustado tiende a ignorar los patrones de la estructura predictiva.

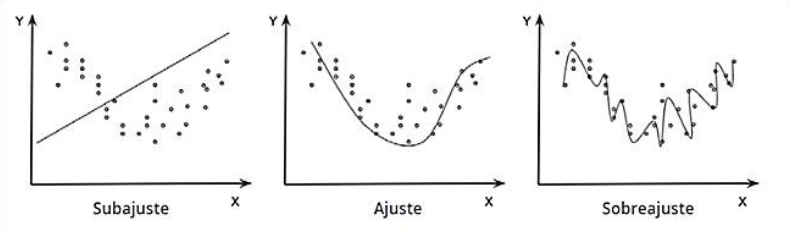


Figura 2.1. *Representación gráfica de Subajuste, Ajuste y Sobreajuste.*

2.1.5 Descubrimiento de conocimiento en bases de datos (KDD)

En la literatura actual se puede encontrar un gran número de definiciones acerca del *"Descubrimiento de conocimiento en bases de datos"*, término que se resume con las siglas KDD (*"Knowledge Discovery in Databases"*). Una de las definiciones más completas, es la siguiente: *"El Descubrimiento de conocimiento en bases de datos es un campo de la Inteligencia artificial de rápido crecimiento, que combina técnicas del aprendizaje de máquina, reconocimiento de patrones, estadística, bases de datos, y visualización para automáticamente extraer conocimiento o información, de un nivel bajo de datos (bases de datos)"* (Usama Fayyad y Evangelos Simoudis, 1997).

KDD es un proceso iterativo, interactivo y comprende numerosos pasos donde puede intervenir el usuario para tomar ciertas decisiones. Dicho proceso será utilizado en este trabajo, capítulo IV, como base para poder extraer de forma eficiente y fehaciente información del conjunto de datos. A continuación, se presentan las diferentes etapas que lo conforman:

1) Entender el dominio de aplicación, cuál es el problema por resolver, y cuáles son los objetivos.

2) ***Selección:*** seleccionar del conjunto de datos originales, un subconjunto apropiado, para el problema que deseamos resolver. Eliminando por ejemplo variables irrelevantes.

3) ***Preprocesamiento:*** en la etapa de limpieza y preprocesamiento se deberían tomar decisiones con respecto a valores faltantes, atípicos, erróneos, entre otros (ruido). También se podría necesitar normalizar los valores de las variables o llevar a cabo otras tareas similares. La etapa de preparación y limpieza es a veces una etapa descuidada, pero de suma importancia en este proceso, dado que grandes cantidades de datos son recolectados por medio de métodos automáticos (ej. vía web). Ingresar estos datos erróneos a los algoritmos de Data Mining solo lleva a entorpecer su proceso de aprendizaje, o a conseguir resultados alejados del comportamiento real.

4) ***Transformación:*** encontrar características útiles para representar a los datos dependiendo de los objetivos. Reducción de dimensiones (ej. Kohonen) para llevar adelante el trabajo con un número reducido de variables. Se utilizan métodos de reducción de dimensiones o de transformación para disminuir el número efectivo de variables bajo consideración o para encontrar representaciones invariantes de los datos (Fayyad et al., 1996).

5) ***Minería de datos (Data Mining)**:* elegir las herramientas de Data Mining adecuadas al problema a resolver, teniendo en cuenta el objetivo (predecir, explicar, clasificar, agrupar, entre otros). También se debe en esta etapa, establecer los parámetros de las redes utilizadas (arquitectura de la red, datos de entrenamiento, de validación y de testeo, entre otros). Una vez realizada la tarea, se procede con el descubrimiento de patrones y relaciones en los datos.

6) ***Interpretación y evaluación**:* interpretación de los datos, llevada a cabo por el analista. En esta etapa se debería consolidar el conocimiento ganado, probando los modelos creados contra los resultados obtenidos de la aplicación de estos modelos en el mundo real.

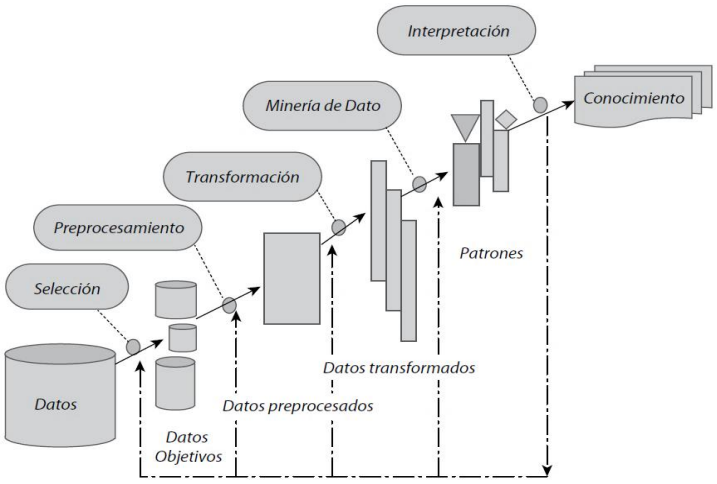


Figura 2.2. *Etapas del proceso KDD.*

2.2 Árboles de decisión

Los *"Árboles de decisión"* forman parte de los algoritmos de clasificación de Aprendizaje supervisado y su objetivo es trabajar sobre muestras históricas para predecir el comportamiento de una nueva entrada e identificar patrones. Su funcionamiento se basa en reglas de decisión donde la cantidad total de muestras están ubicadas en el *"nodo raíz"* y éste se divide en ramas, llamadas *"nodos hijos"*, representando un atributo del conjunto de datos. Los nodos finales, llamados *"nodos hoja"*, corresponden al resultado final de la clasificación generada por el árbol, lo que se conoce como *"etiquetas"* o *"clases"* (*Véase la Figura 2.3*).

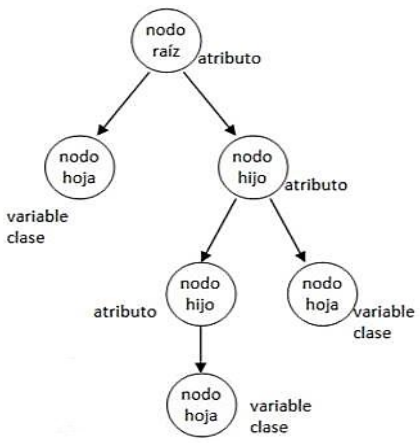


Figura 2.3. *Partes de un Árbol de decisión.*

La característica más importante en un problema de clasificación es que se asume que las clases son disjuntas, es decir, una muestra es de la clase "*a"* o de la clase "*b"*, pero no puede ser al mismo tiempo de las clases "*a"* y "*b"*. Debido al hecho de que la clasificación trata con clases o etiquetas disjuntas, un Árbol de decisión conducirá una muestra hasta una y sólo una hoja, asignando, por tanto, una única clase a la muestra.

Por otra parte, una de las grandes ventajas de los Árboles de decisión es que, son muy útiles para encontrar estructuras en espacios de alta dimensionalidad y en problemas que mezclan datos categóricos y numéricos. Por dicha forma, permiten visualizar la organización de los atributos y son usados en tareas de clasificación, agrupamiento y regresión. Son apropiados precisamente para expresar procedimientos médicos, legales, comerciales, estratégicos, matemáticos, lógicos, entre otros.

Existen dos tipos de árboles: *"Clasificación"* y *"Regresión".* El tipo de árbol que se utiliza dependerá de la naturaleza de la variable de respuesta; siendo los "*Árboles de regresión"* usados para variables de salida continuas (numéricas), y en el caso contrario, cuando se predicen variables categóricas o discretas se utilizan los *"Árboles de clasificación"*.

2.2.1 Estructura

Dado un conjunto inicial de datos, $X_1, X_2, \ldots, X_n$, el árbol se irá creando, realizando una serie de particiones. Estas particiones, serán lo que se van a denominar *"nodos"*, y van a ir dividiendo el espacio en función de una de las variables, con el fin de clasificar los datos. Al final del proceso, se obtendrá un árbol como el mostrado en la *Figura 2.4*.

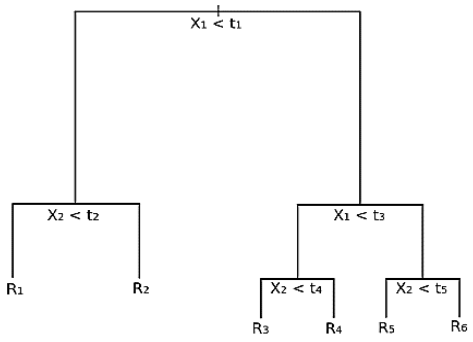


Figura 2.4. *Ejemplo de Árbol de decisión.*

Se puede establecer, por tanto, que la creación de un Árbol de decisión se basa en los siguientes principios:

1. La selección de las divisiones, es decir, cuándo y cómo se divide un nodo.
2. El momento en el que se decide detener las divisiones.
3. Una regla para asignar cada nodo terminal a una clase (Clasificación) o bien una predicción cuantitativa (Regresión).

Para realizar la partición, es necesario determinar un criterio que indique que tan "buena" es la partición a la hora de clasificar los datos y es denominada como la "bondad de la división". A su vez, debemos contar con una regla para determinar cuándo parar de realizar dichas particiones. En los siguientes apartados se explicarán ambos criterios.

2.2.2 Partición

Antes de comenzar a entender cómo los árboles efectúan las particiones, es necesario hablar sobre el tipo de dato de un atributo. De esta manera, se pueden abordar dos casos, uno en el que las variables predictoras son cuantitativas o cualitativas ordinales, y otro que engloba las variables cualitativas nominales.

En el primer caso, en principio son posibles infinitas divisiones, cada una de ellas asociada a un punto de corte c, que divide el rango de valores de la variable en dos regiones disjuntas:

$$\{x < c\} \ y \ \{x \geq c\}, c \ \in R,$$

pero dado el tamaño finito del conjunto de datos, solo hay que considerar las divisiones:

$$\{x < x_{(j)}\}, j = 2, \dots, n \qquad con\, x_{(1)} \ \leq \ x_{(2)} \leq \cdots \leq x_{(n)}$$

Equivalentemente, en el caso cuantitativo se pueden considerar los puntos medios de los intervalos definidos por la muestra ordenada.

En el caso de las variables cualitativas nominales, hay algoritmos que crean tantas particiones como posibles valores tenga el atributo en cuestión. Sin embargo, a modo de ejemplo, en un subconjunto S de categorías de variables, se plantea:

$$\{x \in S\} \ y \ \{x \notin S\}$$

En un principio se deben considerar $2^{k-1} - 1$ subconjuntos para un total de K categorías.

2.2.3 Bondad de la división

A la hora de realizar particiones en un Árbol de decisión, en cada nodo se buscará realizar la división, con el atributo que reduzca al máximo la "*impureza*" del nodo. La suma de todas las "purezas" en cada uno de ellos, será lo que se denomina *"función de impureza"*. Dicha función, permite elegir la mejor división posible de un nodo y conseguir nuevos; donde la mayoría de las muestras pertenezcan a la misma clase.

Existen muchas funciones de impureza, pero las más utilizadas son: el *"Criterio del error esperado"*, *"Criterio Gini"* (Breiman, 1984), los criterios *"Gain"*, *"Gain Ratio"* y la modificación del C4.5 (Quinlan, 1993). Cabe destacar que para poder resolver tareas de regresión se debe aplicar como criterio de impureza: *la varianza*.

Los criterios mencionados, buscan la partición "s" con el menor I(s), definido de la siguiente forma:

$$I(s) = \sum_{j=1..n} p_j \ \cdot \ f(p_j^1, p_j^2, \dots, p_j^c)$$

donde "*n"* es el número de nodos hijos de la partición (número de condiciones de la partición), p_j es la probabilidad de "caer" en el nodo "*j"*, p_j^1 es la proporción de elementos de la clase 1 en el nodo "*j"*, p_j^2 es la proporción de elementos de la clase 2 en el nodo "*j"*, y así para las "*c"* clases. Bajo esta fórmula general, cada criterio de partición implementa una función de impureza, como se expone en la Tabla 2.5:

Criterio	$f(p^1, p^2, \dots, p^c)$
Error Esperado	$min(p^1, p^2, \dots, p^c)$
Gini (CART)	$1 - \sum(P_i)^2$
Entropía (Gain)	$\sum_{i=1}^{n} - p_i \cdot \log_2(p_i)$

Tabla 2.5. *Representación de diversas funciones de impureza.*

Por tanto, la función *I*(*s*) calcula la media ponderada (dependiendo de la cardinalidad de cada hijo) de la impureza de los hijos en una partición. Para formalizar esta idea, en los siguientes apartados se estudiarán algunas funciones, como la "*Entropía"* e "*Índice de Gini"*.

2.2.4 Entropía

La "*Entropía*" es una medida que se aplica para cuantificar el desorden de información o aleatoriedad en las muestras, es decir, mide la incertidumbre de cada atributo. En un nodo, depende de la cantidad de muestras aleatorias que se encuentran en el mismo y se debe calcular para cada uno de ellos. Si un nodo es puro su *entropía* es 0 y solo tiene muestras de una clase, pero si es igual a 1, existe la misma frecuencia de muestras para cada una de las clases. Es posible calcular la *entropía* de un nodo utilizando la siguiente fórmula:

$$Entropía\ (S) = \sum_{i=1}^{n} - p_i\ .\ \log_2(p_i)$$

Donde p_i es la frecuencia de la clase "*i"* en un nodo y "n" es el número de clases únicas. Con esta fórmula podemos probar diferentes atributos para encontrar el que tiene la mejor "predictibilidad", menor entropía. Para ello se debería ir por todos los atributos y calcular la entropía después de la división y, a continuación, elegir el mejor atributo. Una vez elegido, se debe seleccionar el de mayor *"Ganancia de información"*. A continuación, se presenta su concepto.

2.2.5 Ganancia de información

La *"Ganancia de información"* es una propiedad estadística que mide cómo clasifica un atributo a las muestras y es la información que puede aumentar el nivel de certeza después de la división. Específicamente, es la diferencia entre la impureza del nodo padre y la suma ponderada de las dos impurezas del nodo hijo. Para calcular la ganancia de información dado un atributo A con V_a posibles valores y un conjunto de muestras E, se calcula de la siguiente manera:

$$Ganancia(E, A) = Entropía(E) - Entropía(E, A)$$

siendo E_v el subconjunto de muestras para los que el atributo A toma el valor v. Se debería repetir el proceso para cada rama y probar cada uno de los otros atributos para alcanzar la mayor cantidad de hojas puras.

Se puede pensar en la ganancia de información y en la entropía como opuestos. Como la entropía, o la cantidad de aleatoriedad, disminuye, la ganancia de la información, o la cantidad de certeza, aumenta, y viceversa. Por lo tanto, la

construcción de un Árbol de decisión se trata de encontrar atributos que devuelvan la más alta ganancia de información.

2.2.6 Índice de GINI

El "Índice *de Gini"* mide el grado de pureza de un nodo y es la función más utilizada en los Árboles de decisión. Proporciona la probabilidad de no sacar dos muestras de la misma clase del nodo.

Se trata de un índice cuyo rango va de 0 (un corte puro) a 0,5 (corte completamente impuro que divide los datos en partes iguales). A mayor índice de Gini menor pureza, por lo que se selecciona la variable con menor Gini. Si es 0.5 los resultados del corte no son buenos. Seguidamente, se puede observar cómo calcular el índice de Gini.

Cuando una clase *C* puede tomar *n* valores, la entropía del conjunto de muestras *E* respecto a la clase *C* se define como:

$$Gini\ (E) = 1 - \sum_{i=1}^{C} p_i^2$$

siendo p_i la proporción de muestras de E que pertenecen a la clase *i*.

2.2.7 Criterio de parada

En principio podría lograrse un árbol con un nodo final por cada muestra, pero en tal caso se obtiene un modelo muy complejo, que, si bien ofrece un error empírico nulo, a cambio cabe esperar una baja capacidad de generalización, un tamaño del árbol inmanejable y un escenario de sobreajuste. Para evitar que el árbol crezca de forma indiscriminada usamos precisamente un *"Criterio de parada"*, que es simplemente una condición que establecemos para evitar que un nodo continue subdividiéndose.

Determinar una regla de parada depende de la implementación, pero se mencionarán tres criterios fundamentales para establecer estas reglas:

1. Basada en un índice de mejora mínimo.
2. Número mínimo de casos de los grupos.
3. Profundidad máxima del árbol.

El primer criterio de parada consiste en no tener particiones posibles que no superen un índice de mejora mínimo. Es norma establecer un índice de mejora bajo (0,0001) para luego proceder a realizar una poda del árbol (procedimiento que se explicará en el siguiente apartado).

El segundo criterio de parada se refiere a la existencia de límites de mínimo de casos en los grupos para poder seguir particionado. Existen dos tipos de límite: límite para el nodo filial (nodo tras la partición) y límite para el nodo parental (nodo antes de la partición). En el primer caso, dicho límite impedirá que se produzca la partición si ésta conlleva la creación de un grupo con un número de casos inferior a dicho límite. En el segundo, impedirá que éste pueda particionarse si el número de casos de dicho nodo parental es inferior al límite establecido. Como regla general, se establece como referencia, 50 casos para un nodo filial y 100 para un nodo parental (Mercado, 2007).

El tercer criterio de parada consiste en limitar la profundidad del árbol, para evitar que éste se vuelva demasiado “poblado” y conlleve un aumento de la dificultad para la interpretación del árbol.

2.2.8 Poda y reestructuración

Cuando un árbol presenta sobreajuste es necesario realizar una “*poda”*. Consiste en cortar sucesivamente las ramas y nodos terminales hasta lograr un tamaño adecuado para dicho árbol. De esta forma, estamos logrando mayor generalización. Al proceso de *poda* se lo puede segmentar en dos:

- **Prepoda:** el proceso se realiza durante la construcción del árbol. Se trata de determinar el criterio de parada a la hora de seguir especializando una rama. En general, los criterios de prepoda pueden estar basados en el número de muestras por nodo, en el número de excepciones respecto a la clase mayoritaria (error esperado) o en técnicas más sofisticadas.

- **Pospoda:** el proceso se realiza después de la construcción del árbol. Consiste en eliminar nodos de abajo a arriba hasta un cierto límite. Suele brindar mejores resultados dado que se realiza con una visión completa del modelo.

Es posible combinar ambas técnicas. Una consecuencia de utilizar prepoda o pospoda (o los dos) es que los nodos hoja ya no van a ser puros, es decir, es posible que tengan muestras de varias clases. Normalmente se elegirá la clase con más muestras para etiquetar el nodo hoja y hacer, por tanto, la predicción.

2.2.9 Ventajas y desventajas de Árboles de decisión

Las Árboles de decisión presentan una serie de ventajas:

- Son fáciles de construir, interpretar y visualizar.
- Sirven tanto para variables dependientes cualitativas como cuantitativas, como para variables predictoras o independientes numéricas y categóricas.
- No necesita variables *dummys*, aunque a veces mejoran el modelo.
- Permiten que falten valores en alguna de las variables.
- Son capaces de tratar con grandes conjuntos de datos formados por un gran número de atributos.

Respecto a sus desventajas:

- Tienden al sobreajuste de los datos, por lo que el modelo al predecir nuevas muestras no estima con el mismo índice de acierto.
- Se ven influenciadas por los *outliers*, creando árboles con ramas muy profundas que no predicen bien para nuevas muestras. Se deben eliminar dichos *outliers*.
- No suelen ser muy eficientes con modelos de regresión.
- Crear árboles demasiado complejos puede conllevar que no se adapten bien a las nuevas muestras.
- La complejidad resta capacidad de interpretación.
- Se pueden crear árboles sesgados si una de las clases es más numerosa que otra.

2.2.10 Algoritmos de Árboles de decisión

Dentro de los Árboles de decisión encontramos diferentes algoritmos, donde cada uno define cómo dividen las muestras en cada nodo. Los más usados en el modelamiento son ID3, C4, C4.5, CART y Bosques Aleatorios (Random Forest). Los

primeros tres fueron desarrollados por Quinlan, en cambio, CART y Random Forest por Breiman. A continuación, se explicará rápidamente cada uno de ellos, y en el siguiente apartado, se interioriza en Bosques Aleatorios puesto que es el algoritmo elegido para realizar el experimento en este trabajo.

- CART [Breiman 1984] y derivados: son métodos "divide y vencerás" que construyen árboles binarios y se basan en el criterio de partición GINI y que sirve tanto para clasificación como para regresión. La poda se basa en una estimación de la complejidad del error ("error-complexity"). Generalmente se pueden encontrar en paquetes de minería de datos con el nombre C&RT.

- ID3 (Induction Decision Trees) [J. Ross Quinlan, 1986]: se trata de un algoritmo que recorre el Árbol de decisión desde la raíz y tanto en ella como en cada uno de los demás nodos se decide qué rama tomar basándose en el valor de algún atributo del objeto que se esté clasificando, hasta llegar a un nodo terminal (hoja), que corresponde a la clase final a la cual pertenece dicho objeto.

- C4.5 [Quinlan 1993]: surge como una modificación propuesta por el propio Quinlan, con el fin de mejorar el algoritmo ID3. Construye el árbol de clasificación a partir del conjunto de entrenamiento y después de hacerlo utiliza una técnica de poda basada en aplicar un test de hipótesis para deducir si merece la pena o no expandir dicha rama. El refinamiento que introduce respecto a ID3 es una medida alternativa llamada *ratio de ganancia*, es decir, en cada nodo selecciona el atributo con mayor ratio de ganancia de información evitando así favorecer la elección de atributos con un mayor número de valores. Una implementación de este algoritmo es la función J48 en la herramienta Weka de minería de datos.

2.2.11 Bosques Aleatorios (Random Forest)

El algoritmo "*Bosques Aleatorios"* (del inglés "*Random Forest"*), es una técnica de Aprendizaje supervisado. Precisamente, son bosques de decisión aleatorios formados por un conjunto de Árboles de decisión como los que se han explicado anteriormente. Estos bosques, se forman mediante un algoritmo que introduce una aleatoriedad para reducir la correlación entre los árboles. Una vez construido el bosque, se realiza la predicción. Fue propuesto inicialmente por Kam Ho (1995) y posteriormente desarrollado por Breiman (2001), quien presentó el modelo totalmente desarrollado.

Random Forest surge como combinación de las técnicas de *"Classification And Regression Tree (CART)"* y *"Bootstrap Aggregating"* (Bagging) para realizar la combinación de árboles predictores. Cada árbol, depende de los valores de un vector aleatorio probado independientemente y con la misma distribución para cada uno de estos, es decir, genera múltiples Árboles de decisión sobre un conjunto de datos de entrenamiento y los resultados obtenidos se combinan a fin de obtener un modelo único más robusto en comparación con los resultados de cada árbol por separado.

Por otro lado, *"Bootstrapping"* es una técnica estadística utilizada en varios algoritmos de Aprendizaje automático. Consiste en formar subconjuntos de muestras aleatorias con reemplazamiento, es decir, permite obtener subconjuntos de una población donde una muestra se puede considerar en más de un subconjunto (*Véase la Figura 2.6*).

Bagging también fue propuesto por Breiman (1994), la idea central es la de entrenar muchos clasificadores débiles independientes, para luego combinarlos todos

en un clasificador fuerte, usando muestreo con reemplazamiento en el conjunto de datos.

Cada árbol se obtiene mediante un proceso de dos etapas:

1. Se genera un número considerable de Árboles de decisión con el conjunto de datos. Cada árbol contiene un subconjunto aleatorio de variables m (predictores) de forma que m < M (donde M = total de predictores).

2. Cada árbol crecerá hasta su máxima extensión.

Cada árbol generado por el algoritmo Random Forest contiene un grupo de muestras aleatorias (como se ha dicho, elegidas mediante *Bootstrap)*. Las muestras no estimadas en los árboles (también conocidas como "*out of the bag*") se utilizan para validar el modelo. Las respuestas de todos los árboles se combinan en una salida final (conocida como *"ensamblado"*) que se obtiene mediante la siguiente regla:

- *Ensamblado categórico:* cuando la salida de los árboles es categórica, cada uno *"vota"* por una clase y se asigna la más "votada".

- *Ensamblado numérico:* el resultado del modelo es el promedio de las salidas de todos los árboles.

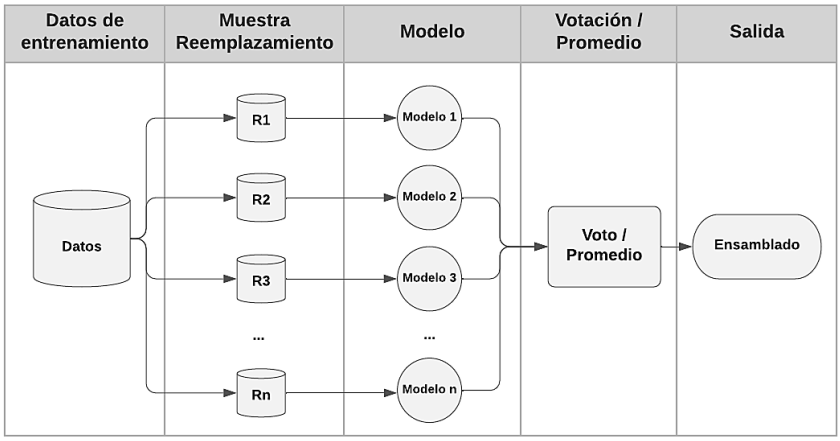


Figura 2.6. *Técnica Bootstrapping Aggregating y Random Forest*.

Las principales ventajas del algoritmo Random Forest (Cánovas *et al.,* 2017) son:

- Se considera un método muy preciso y robusto debido a la cantidad de árboles de decisión que participan en el proceso.
- No sufre el problema de sobreajuste. La razón principal es que toma el promedio de todas las predicciones, lo que cancela los sesgos.
- Se puede utilizar tanto en problemas de clasificación como de regresión.
- Pueden manejar valores faltantes. Hay dos formas de manejar esto: usar valores medianos para reemplazar variables continuas y calcular el promedio ponderado por proximidad de los valores faltantes.

Por otra parte, las desventajas son las siguientes:

- Es lento para generar predicciones porque tiene múltiples árboles de decisión. Cada vez que hace una predicción, todos los árboles en el bosque tienen que

hacer una predicción para la misma entrada dada y luego votarla. Todo este proceso requiere mucho tiempo.

- El modelo es difícil de interpretar en comparación con un Árbol de decisión, en el que se puede tomar una decisión fácilmente siguiendo la ruta del árbol.

2.3 Redes Neuronales Artificiales

Las "*Redes neuronales artificiales*" (RNA) son modelos que intentan reproducir el comportamiento del cerebro humano y están conformadas por un gran número de elementos simples de procesamiento llamados *"nodos"* o *"neuronas artificiales"*, organizados en capas. Cada una se caracteriza en un instante cualquiera por un estado de activación que transforma el estado actual en una señal de salida y están conectadas con otras mediante enlaces de comunicación con un peso asociado. Los pesos representan la información que será usada por la red neuronal para resolver un problema determinado. De este modo, las RNA son sistemas adaptativos que aprenden de la experiencia y lo logran mediante un entrenamiento con muestras históricas, que les permite crear su propia representación interna del problema. En virtud de ello, se dice que son autoorganizadas. Posteriormente, pueden responder apropiadamente cuando se les presentan situaciones a las que no habían sido expuestas anteriormente, es decir, son capaces de generalizar a partir de casos anteriores nuevas ocurrencias. Esta característica es fundamental ya que permite a la red responder correctamente, no solo ante información novedosa, si no también ante información distorsionada o incompleta.

2.3.1 Funcionamiento

Como ya se mencionó en el apartado anterior, la neurona artificial pretende mimetizar las características más importantes de una neurona biológica. En general, recibe las señales de entrada de las neuronas vecinas ponderadas por los pesos de las conexiones. La suma de estas señales ponderadas proporciona la entrada total o neta de la neurona y, mediante la aplicación de una función matemática (denominada *"función de salida"*), sobre la entrada neta, se calcula un valor de salida, la cual es enviada a otras neuronas (*Véase la Figura 2.7*). Tanto los valores de entrada a la neurona como su salida pueden ser señales excitadoras (cuando el valor es positivo) o inhibitorias (cuando el valor es negativo).

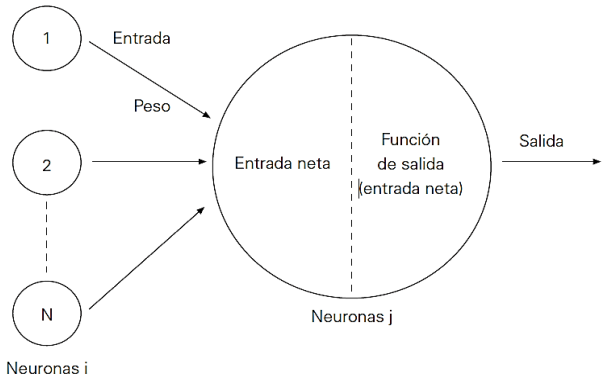


Figura 2.7. *Modelo de una neurona artificial.*

2.3.2 Arquitectura

Las neuronas que componen una RNA se organizan de forma jerárquica formando capas. Una capa o nivel es un conjunto de neuronas cuyas entradas de información provienen de la misma fuente (que puede ser otra capa de neuronas) y cuyas salidas de información se dirigen al mismo destino (también puede ser otra capa de neuronas). En este sentido, se distinguen tres tipos de capas:

- *Capa de entrada:* recibe la información del exterior.

- *Capa oculta:* son aquellas cuyas entradas y salidas se encuentran dentro del sistema y, por tanto, no tienen contacto con el exterior. Se encargan de procesar la información ingresada.

- *Capa de salida:* envía la respuesta de la red al exterior.

En función de la organización de las neuronas en la red, formando capas o agrupaciones, podemos encontrar dos tipos de arquitecturas básicas: *"redes monocapa"* y *"redes multicapa"*.

Las redes monocapa están organizadas, como el propio nombre indica, en una sola capa de neuronas (*Véase la Figura 2.8*). Cada neurona está conectada con todas las demás que conforman la arquitectura. Este tipo de redes se suelen utilizar en tareas denominadas autoasociativas. Para ello, mediante una etapa de entrenamiento en los pesos de la red se almacena cierta información. Posteriormente, cuando se presenta una información a la entrada de la red, ésta responde proporcionando la información más parecida de las almacenadas. Por tal motivo, las redes que llevan a cabo este tipo de tareas también reciben el nombre de *"redes autoasociativas"* ya que intentan asociar una información consigo misma. Este tipo de redes son útiles para regenerar información de entrada que se presenta incompleta o distorsionada como por ejemplo las imágenes.

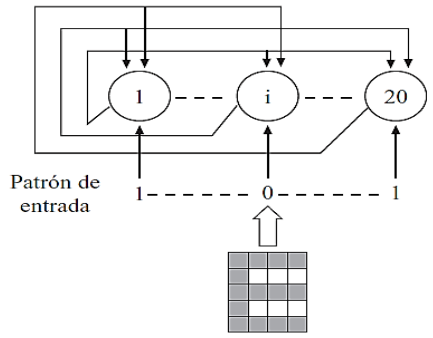


Figura 2.8. *Modelo de una red monocapa.*

Por su parte, las redes multicapa disponen de conjuntos de neuronas agrupadas en dos o más capas. Este tipo de redes están constituidas por una capa de entrada, una capa de salida y una o más capas intermedias u ocultas; donde la información se transmite desde la de entrada hasta la de salida y donde cada neurona está conectada con todas las de la capa siguiente (*Véase Figura 2.9*).

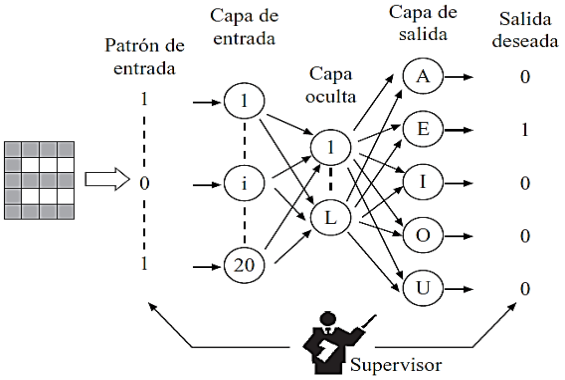


Figura 2.9. *Representación de un perceptrón multicapa supervisado.*

Así mismo, las redes multicapa se las puede clasificar en base a dos tipos básicos:

- *Feedforward:* donde todas las señales neuronales se propagan hacia adelante a través de las capas de la red y normalmente las conexiones autorecurrentes no son útiles en aplicaciones de reconocimiento o clasificación de patrones.

- *Feedforward/feedback:* en este tipo de redes circula información tanto hacia adelante como hacia atrás durante el funcionamiento de la red.

Las redes multicapa se suelen utilizar en tareas donde se procura que la red aprenda parejas de datos o muestras, de forma que cuando se presenta cierta información de entrada "A" deberá responder la correspondiente salida asociada "B". Por tal motivo, las redes que llevan a cabo este tipo de trabajo también reciben el nombre de *"redes heteroasociativas"*. Por ello, son útiles para la clasificación de patrones -ya que, en este caso, se asocia el ejemplo con la clase o categoría a la que pertenece, y la aproximación de funciones- donde se asocia una información de entrada con otra información de salida.

El tipo de arquitectura multicapa descrito se denomina perceptrón multicapa y ha sido el más utilizado en el campo de Aprendizaje automático. Su utilidad reside en la habilidad para operar como aproximador universal de funciones, es decir, puede aprender virtualmente cualquier relación entre un conjunto de variables de entrada y salida. Esta habilidad es el resultado de la adopción, por parte de las neuronas de la capa oculta, de una función de salida no lineal. Por su parte, el análisis discriminante lineal derivado de la estadística clásica no posee la capacidad de calcular funciones no lineales y, por tanto, presentará un rendimiento inferior frente al perceptrón multicapa en tareas de clasificación que impliquen relaciones no lineales complejas.

2.3.3 Aprendizaje

El conocimiento de una RNA se encuentra distribuido en los pesos de las conexiones entre las neuronas que forman la red. Todo proceso de aprendizaje implica cierto número de cambios en estas conexiones. Por ende, se explica brevemente cómo se comportan en base al tipo de aprendizaje:

- Aprendizaje no supervisado: la red debe autoorganizarse (es decir, auto enseñarse) dependiendo de algún tipo de estructura existente en el conjunto de datos de entrada. Típicamente esta estructura suele deberse a redundancias o agrupamientos en el conjunto de datos.
- Aprendizaje supervisado: el conjunto de datos de entrada es propagado hacia delante hasta que la activación alcanza las neuronas de la capa de salida. La respuesta calculada por la red es comparada con aquella que se desea obtener. Si la respuesta no es la deseada entonces se ajustan los pesos. Una vez obtenidos y guardados los pesos óptimos en la fase de entrenamiento se debe seguir con la fase de prueba.

2.3.4 Redes de propagación hacia atrás (Backpropagation)

Las *redes de propagación hacia atrás de errores o retropropagación* (del inglés *"Backpropagation"*), es un método de cálculo del gradiente utilizado en algoritmos de Aprendizaje supervisado para entrenar una RNA.

El nombre de Backpropagation resulta de la forma en que el error es propagado hacia atrás desde la capa de salida. Esto permite que los pesos sobre las

conexiones de las neuronas ubicadas en las capas ocultas cambien durante el entrenamiento.

Para entender su funcionamiento comencemos por el error. El mismo se calcula cuantificando la diferencia entre la salida prevista y la salida deseada. Esta diferencia se denomina *"pérdida"* y la función utilizada para calcular la diferencia se denomina *"función de pérdida"*. Las funciones de pérdida pueden ser de diferentes tipos, por ejemplo, *error cuadrático medio* o funciones de *entropía cruzada*. Una vez calculado el error, el siguiente paso es minimizarlo. Para ello, se calcula la derivada parcial de la función de error con respecto a todos los pesos y sesgos. Esto se llama *"descenso del gradiente"*. Las derivadas se pueden usar para encontrar la pendiente de la función de error. Si la pendiente es positiva, se puede reducir el valor de los pesos o si la pendiente es negativa, se puede aumentar el valor. Esto reduce el error general. La función que se utiliza para reducir el error se llama *"función de optimización"*.

Este ciclo único de propagación hacia adelante y hacia atrás se denomina una "*época*". Este proceso continúa hasta que se logra una precisión razonable. No existe un estándar para determinar qué tan razonable debe ser, lo ideal es que se esfuerce por lograr una precisión del 100%, pero esto es extremadamente difícil de conseguir para cualquier conjunto de datos. En muchos casos, se considera aceptable una precisión superior al 90%, pero realmente depende de su caso de uso.

2.3.5 Redes convolucionales

Las *"Redes convolucionales"* son otra variante de redes neuronales. Son utilizadas principalmente para el procesamiento de imágenes, pues son capaces de modelar variaciones y comportamientos complejos. Estas redes se caracterizan por el modelado consecutivo de pequeñas piezas de información de la entrada de datos a través de filtros que aumentan drásticamente la dimensionalidad del problema. Estas operaciones son conocidas como convoluciones, adicionalmente, utilizan otros tipos de capas que submuestrean los datos, o capas densamente conectadas, entre otras.

2.3.6 Autocodificadores (Autoencoders)

Un "*Autocodificador*" (del inglés "*Autoencoder*"), es un tipo específico de redes neuronales feedforward de aprendizaje no supervisada donde la salida esperada es la misma que la entrada, es decir, utiliza un algoritmo de propagación hacia atrás para hacer que el valor de salida sea igual al valor de entrada. Esta peculiaridad se debe a que el Autocodificador no debe identificar ninguna clase, sino que tiene como objetivo reconstruir la entrada que le ha sido introducida, por eso se trata de un modelo no supervisado. A modo de ejemplo, dada una imagen de un dígito escrito a mano, un Autocodificador primero codifica la imagen en una representación latente de menor dimensión y luego decodifica la representación latente de nuevo en una imagen (*Véase la Figura 2.10*).

Los Autocodificadores aprenden a comprimir los datos mientras minimiza el error de reconstrucción y están constituidos por:

- **Codificador (Encoder)**: comprime la entrada en una representación espacial latente (llamada comúnmente también como *código*) y puede expresarse mediante la función de codificación $h = f(x)$.

- **Decodificador (Decoder)**: Esta parte puede reconstruir la entrada a partir de la representación espacial latente, que puede expresarse mediante la función de decodificación $r = g(h)$.

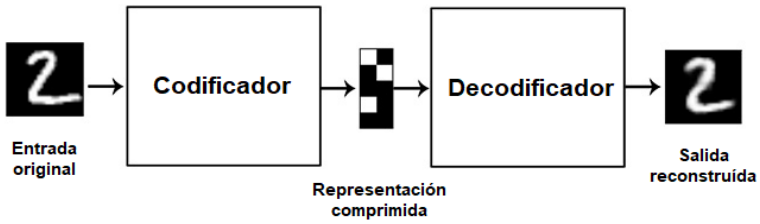


Figura 2.10. *Componentes de un Autocodificador.*

En el capítulo III, se interioriza sobre este tipo de Red neuronal, dado que, puede ser utilizado como una alternativa para abordar el tratamiento de desbalance de clases.

2.3.7 Ventajas y desventajas de una RNA

Las RNA presentan una serie de ventajas:

- *Aprendizaje adaptativo:* capacidad de aprender a realizar tareas basadas en un entrenamiento o una experiencia inicial. Como las Redes neuronales pueden aprender a diferenciar patrones mediante muestras y entrenamiento, no es necesario que elaboremos modelos a priori ni necesitamos especificar funciones de distribución de probabilidad.

- *Autoorganización:* una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje. Esta autoorganización provoca la generalización: facultad de las redes neuronales de responder apropiadamente cuando se les presenta datos o situaciones a las que no habían sido expuestas anteriormente.

- *Tolerancia a fallos:* la destrucción parcial de una red conduce a una degradación de su estructura, sin embargo, algunas capacidades de la red se pueden retener incluso sufriendo un gran daño.

- *Fácil inserción dentro de la tecnología existente:* se pueden obtener chips especializados para redes neuronales que mejoran su capacidad en ciertas tareas. Ello facilitará la integración modular en los sistemas existentes. Una red individual puede ser entrenada para desarrollar una única y bien definida tarea, en cambio para tareas complejas, que hagan múltiples selecciones de patrones, requerirán sistemas de redes interconectadas.

Respecto a las desventajas que presentan las RNA, una de las más importantes consiste en que es difícil comprender la naturaleza de las representaciones internas generadas por la red para responder ante un problema determinado. Es decir, se desconoce cómo el sistema interrelaciona las diferentes variables de entrada con los pesos de las conexiones entre neuronas para elaborar una solución. Esta limitación contrasta con los diferentes modelos estadísticos, los cuales permiten observar los parámetros o pesos relativos que el modelo otorga a cada una de las variables que intervienen en el modelo.

2.3.8 Aplicaciones

Las Redes neuronales son una tecnología computacional emergente que pueden utilizarse en un gran número y variedad de aplicaciones:

Disciplina	Aplicación
Biología	- Aprender más acerca del cerebro y otros sistemas. - Obtención de modelos de la retina.
Empresa	- Evaluación de probabilidad de formaciones geológicas y petrolíferas. - Explotación de bases de datos. - Reconocimiento de caracteres escritos.
Medio ambiente	- Analizar tendencias y patrones. - Previsión del tiempo.
Finanzas	- Previsión de la evolución de los precios. - Valoración del riesgo de los créditos. - Identificación de falsificaciones. - Interpretación de firmas.
Manufacturación	- Robots automatizados y sistemas de control (temperatura, gas, etc.) - Control de producción en líneas de proceso. - Inspección de la calidad.
Medicina	- Analizadores del habla para la ayuda de audición de sordos profundos. - Diagnóstico y tratamiento a partir de síntomas y/o de datos analíticos (Electrocardiograma, encefalograma, análisis sanguíneo, etc.). - Monitorización en cirugía. - Predicción de reacciones adversas a los medicamentos. - Lectores de rayos X. - Entendimiento de la causa de los ataques epilépticos.
Militares	- Clasificación de las señales de radar. - Creación de armas inteligentes. - Optimización del uso de recursos escasos. - Reconocimiento y seguimiento en el tiro al blanco.
Autocodificadores	- Eliminación de ruido de datos (por ejemplo, imágenes, audio). - Imagen en pintura. - Recuperación de información. - Reducción de dimensionalidad.

Figura 2.11. *Aplicaciones de Redes Neuronales en distintas disciplinas.*

Capítulo III
Métodos de balanceo de clases

En un problema de clasificación un "*conjunto de datos desbalanceados"* se produce cuando la distribución de las muestras entre las clases conocidas no está distribuida por igual. Por lo tanto, estos tipos de conjuntos tendrán una o más clases con mayor cantidad de muestras, denominadas *"clases mayoritarias"*, y una o más clases con una pequeña cantidad, denominadas *"clases minoritarias"*. Este tipo de inconveniente se presenta con frecuencia en el mundo real.

Uno de los grandes problemas que enfrentan los algoritmos de clasificación es dicha dificultad, de modo que, consiguen buenos resultados cuando trabajan con muestras de la clase mayoritaria, pero cuando lo hacen con la clase minoritaria son frecuentemente inexactos. Vale decir, obtienen buenos resultados con conjuntos de datos balanceados, sin embargo, no alcanzan un buen rendimiento con los desbalanceados. Las diversas razones para ello son:

- Muchas medidas de rendimiento usadas para guiar el proceso de entrenamiento penalizan a las clases minoritarias.

- Las reglas que predicen las clases minoritarias están muy especializadas y su cobertura es muy baja, por ello, a menudo son descartadas en favor de reglas más generales, es decir, aquellas que predicen a las clases mayoritarias.

- El tratamiento del ruido puede afectar a la clasificación de las clases minoritarias, ya que estas pueden ser identificadas como ruido y ser descartadas erróneamente o el ruido existente puede afectar en gran medida la clasificación de estas.

No obstante, el principal obstáculo es que los algoritmos de clasificación se entrenan con un mayor número de muestras de la clase mayoritaria y cometen más errores cuando intentan clasificar la clase minoritaria. Para afrontar dicho problema han surgido numerosas propuestas, sin embargo, en este trabajo solo se abordarán las siguientes:

1. ***Muestreo de datos:*** el objetivo es, a partir de las muestras originales, obtener un nuevo conjunto de datos cuyas clases tengan una distribución más balanceada. En este caso, el algoritmo de clasificación no sufre ninguna modificación y solo implica modificar el conjunto de datos original.

2. ***Aprendizaje sensible al costo:*** este tipo de solución puede incorporar modificaciones a nivel de datos y de algoritmo. Basadas en selección de umbral o en incluir penalizaciones más fuertes a los errores cometidos con la clase minoritaria que a los que se producen al clasificar la clase mayoritaria.

3. ***Métodos de ensamble:*** consisten en la combinación de dos o más clasificadores.

Cabe destacar, al abordar problemas con datos desbalanceados, que se deben considerar otros factores que pueden tener gran influencia en los resultados obtenidos, por ejemplo, el solapamiento entre clases o el ruido existente en los datos.

3.1 Muestreo de datos

El método de *"Muestreo de datos"* o "*Remuestreo*", ofrece una flexibilidad para generar nuevas muestras o reconsiderar diferentes partes del espacio de atributos. En general, este método tiene como objetivo transformar datos desbalanceados en una distribución equilibrada utilizando algún mecanismo. La modificación se produce alterando el tamaño del conjunto de datos original y abasteciendo la misma proporción de muestras en cada clase, para lograr un equilibrio. Las "muestras raras" o excepcionales se ubican en áreas particulares del espacio, por lo que en problemas de clasificación la presencia de estas muestras les corresponde una importante consideración, en particular para aislar dichos casos y modificar su proporción respecto de la otra clase, para así beneficiar al proceso de clasificación. Los "casos raros" son los de mayor interés para indagar porque son más propensos a generar errores en la clasificación. Por consiguiente, el método de Remuestreo está conformado por las siguientes técnicas:

- ***Submuestreo:*** consiste en modificar la distribución del conjunto de datos reduciendo el número de muestras de la clase mayoritaria.

- ***Sobremuestreo:*** consiste en modificar la distribución del conjunto de datos incrementando el número de muestras de la clase minoritaria.

- ***Algoritmos Híbridos:*** consiste en combinar ambas técnicas.

Los métodos avanzados de Remuestreo utilizan cierta inteligencia al reducir o añadir muestras para crear distribuciones equilibradas de datos. Por lo que, se puede dividir también por el tipo que se utiliza para llevar adelante la modificación del conjunto de datos:

- ***Aleatorios*****:** involucra la selección aleatoria de cualquier muestra de alguna de las dos clases.

- ***Duplicados*****:** se tomarán muestras duplicadas tanto para eliminar como para aumentar.

- ***Muestras Cercanas*****:** se toma una norma de cercanía respecto de los atributos de las muestras para llevar a cabo el cambio de proporción.

Existen muchas técnicas de Muestreo de datos para elegir, con el objetivo de ajustar la distribución de clases del conjunto de datos. No existe la mejor técnica, al igual que no existe el mejor algoritmo de Aprendizaje automático. Las técnicas se comportan de manera diferente dependiendo de la elección del algoritmo de aprendizaje, de la densidad y composición del conjunto de datos. En muchos casos, el muestreo puede mitigar los problemas causados por un desbalance de clases, pero no hay un claro "ganador" entre los diversos enfoques. Además, muchas técnicas de modelado reaccionan de manera diferente al muestreo, lo que complica aún más la idea de una guía simple sobre qué procedimiento usar. Como tal, es importante diseñar cuidadosamente experimentos para probar y evaluar un conjunto de métodos y configuraciones diferentes para descubrir qué funciona mejor en un proyecto específico.

3.1.1 Submuestreo

La técnica de *"Submuestreo"* (del inglés *"Undersampling"*), consiste en balancear la distribución de los datos reduciendo el número de muestras de la clase mayoritaria y dejando intacta la cantidad de la clase minoritaria. A pesar de su sencillez y de la reducción del tiempo de procesado de los datos, existe el riesgo de eliminar muestras potencialmente importantes en el proceso de clasificación, por esta razón, se han desarrollado algoritmos capaces de realizar una selección inteligente sobre las muestras del conjunto de datos de la clase mayoritaria.

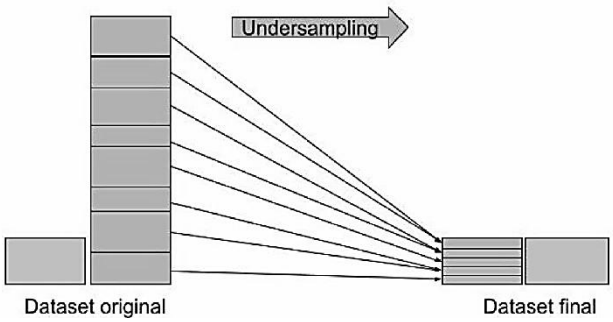


Figura 3.1. *Undersampling elimina muestras de la clase mayoritaria.*

En la figura anterior, se observa cómo se produce una disminución de la clase mayoritaria entre el lado izquierdo al lado derecho a raíz de Undersampling.

Ventajas:

- Es recomendable utilizar Undersampling cuando el conjunto de datos es de gran tamaño, ya que, al reducir la cantidad de muestras ayuda a mejorar el tiempo de ejecución del modelo y resolver los problemas de almacenamiento.

Desventajas:

- Existe el riesgo de eliminar muestras potencialmente importantes en el proceso de clasificación, por tanto, puede hacer que el clasificador tenga un comportamiento inadecuado con muestras desconocidas.

3.1.1.1 Algoritmos de Submuestreo

La reducción de la clase mayoritaria puede realizarse de numerosos modos, por esta razón, a continuación, se presentan algunos algoritmos de Undersampling:

- **Submuestreo aleatorio (RUS).**
- **Tomek Link.**
- **Vecinos cercanos (NearMiss).**

3.1.1.2 Submuestreo aleatorio (RUS)

El *"Submuestreo aleatorio"* (del inglés *"Random UnderSampling", RUS*), descrito por Van-Hulse, consiste en eliminar muestras de la clase mayoritaria para equilibrar el conjunto de datos.

La desventaja de este método es que ignora información potencialmente útil que puede ser importante para inducir a los clasificadores. Esta pérdida de información conduce a un menor rendimiento de los modelos. Además, RUS reduce el tamaño de la muestra, lo que contribuye aún más al empeoramiento del modelo.

Además, esta técnica es adecuada para conjuntos de datos con bajo desequilibrio y no es apropiada para distribuciones altamente desequilibradas y con bajo número de instancias.

3.1.1.3 Tomek links

La técnica *"Tomek links",* descrito por Ivan Tomek (1976), se basa en el vecino más cercano. Consiste en eliminar sólo muestras de la clase mayoritaria que sean redundantes o que estén muy próximas a muestras de la clase minoritaria. Un "*Tomek link"* es un par de muestras de clases diferentes pero que son muy parecidas. El algoritmo Tomek busca este tipo de pares y elimina el dato perteneciente a la clase mayoritaria. Así se pueden establecer grupos bien definidos en el conjunto y conducir a un mejor rendimiento de la clasificación (*Véase la Figura 3.2*).

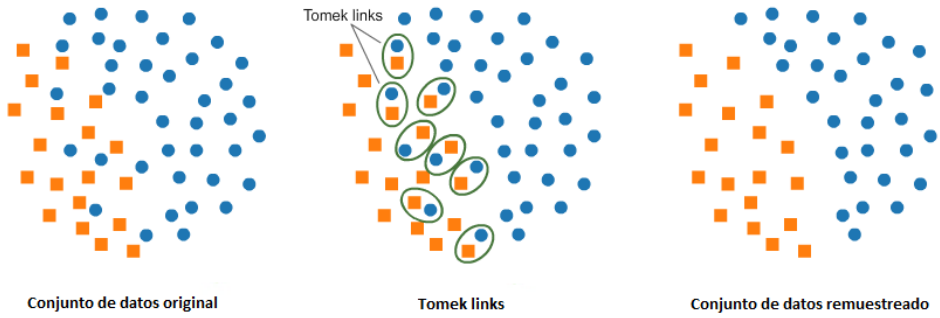


Figura 3.2. *Funcionamiento Tomek Links.*

Aunque el objetivo de la técnica no es devolver datos equilibrados, puede considerarse como un método de *Submuestreo* cuando se eliminan las muestras de la clase mayoritaria.

3.1.1.4 Vecinos cercanos (NearMiss)

La técnica *"NearMiss"*, descripto por Zhang y Mani (2003), se basa en el algoritmo del vecino más cercano. Cuando las muestras de dos clases diferentes están muy cerca una de la otra, se eliminan las muestras de la clase mayoritaria para aumentar los espacios entre las dos clases. Los conceptos básicos sobre su funcionamiento son:

- ***Paso 1:*** se encuentran las distancias entre todas las muestras de la clase mayoritaria y la minoritaria.
- ***Paso 2:*** Luego, se seleccionan ***"n"*** muestras de la clase mayoritaria que tienen las distancias más pequeñas a las de la clase minoritaria.
- ***Paso 3:*** si hay ***"k"*** muestras en la clase minoritaria, la técnica del más cercano dará como resultado $k \, . n$ muestras de la clase mayoritaria.

Para encontrar **"n"** muestras más cercanas en la clase mayoritaria, existen variaciones de la aplicación del algoritmo NearMiss:

- ***Versión 1***: selecciona para limpiar aquellas muestras de la clase mayoritaria cuya distancia media a las *k* muestras más cercanas de la clase minoritaria es menor.

- ***Versión 2*:** selecciona aquellas muestras de la clase mayoritaria cuya distancia media a las ***k*** muestras más lejanas de la clase minoritaria es menor.

- ***Versión 3*:** selecciona un número dado de muestras de la clase mayoritaria más próximas a cada muestra de la clase minoritaria.

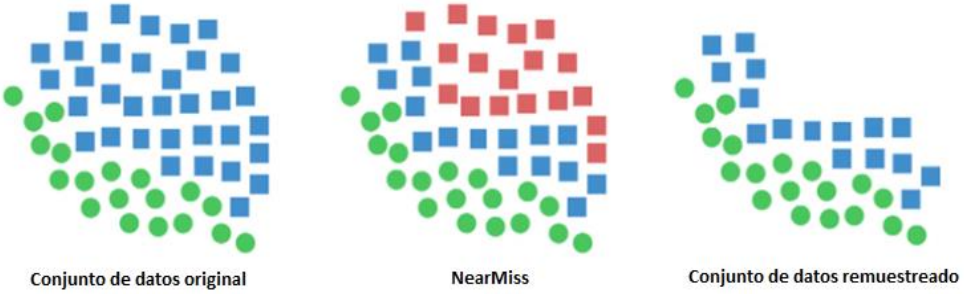


Figura 3.3. *Funcionamiento NearMiss Versión 2.*

3.1.2 Sobremuestreo

La técnica de *"Sobremuestreo"* (del inglés *"Oversampling"*), consiste en balancear la distribución de los datos incrementando el número de muestras de la clase minoritaria y dejando intacta la cantidad de la clase mayoritaria. Sin embargo, Oversampling posee una gran dificultad y es que añadir muestras a la clase minoritaria para balancear el conjunto de datos, crea muestras que no se pueden asegurar que provengan de la distribución original. De este modo, se genera ruido para los clasificadores y lo cual podría resultar en pérdida de rendimiento en términos de clasificación.

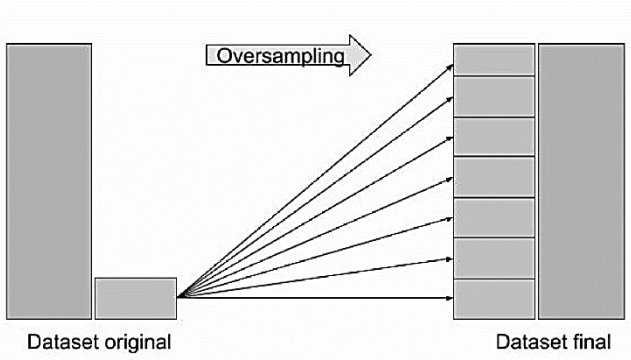


Figura 3.4. *Oversampling agrega muestras a la clase minoritaria.*

En la figura anterior, se observa cómo se produce un aumento de la clase minoritaria entre el lado izquierdo al lado derecho a raíz de Oversampling.

Ventajas:

- No conduce a ninguna pérdida de información.
- Es recomendable utilizar Oversampling cuando el conjunto de datos es pequeño.

Desventajas:

- Aumenta el tiempo de procesamiento de datos.
- Generación de muestras ruidosas.
- Podría ocasionar un sobreajuste, ya que, algunas muestras de la clase minoritaria podrían aparecer en el conjunto de entrenamiento con mayor frecuencia respecto a la clase mayoritaria.

3.1.2.1 Algoritmos de Sobremuestreo

El incremento de la clase minoritaria puede realizarse de numerosas maneras, por esta razón, a continuación, se presentan algunos algoritmos de Oversampling:

- **Sobremuestreo aleatorio (ROS)**
- **Sobremuestreo sintético (SMOTE)**
- **Muestreo sintético adaptativo (ADASYN)**

3.1.2.2 Sobremuestreo aleatorio (ROS)

El *"Sobremuestreo Aleatorio"* (del inglés "*Random OverSampling", ROS*), descrito por Van-Hulse, consiste en equilibrar el conjunto de datos replicando las muestras de la clase minoritaria.

La ventaja de este método es que no produce pérdida de información como en RUS. La desventaja es que conduce a un sobreajuste e introduce un coste computacional adicional si el índice de desequilibrio del conjunto de datos es alto. Además, la técnica ROS aumenta la proporción de solapamiento entre instancias de diferentes clases, debido a la duplicación de datos, ya que la replicación de las muestras hace que la similitud entre los atributos sea mayor. ROS es muy eficiente para los conjuntos de datos con un alto índice de desequilibrio.

3.1.2.3 Sobremuestreo Sintético (SMOTE)

El "*Sobremuestreo Sintético"* o *SMOTE* ("*Synthetic Minority Oversampling Technique")*, descrito por Chawla (2002), consiste en generar muestras "sintéticas" o artificiales para equilibrar el conjunto de datos basándose en la regla del vecino más cercano. La generación se realiza extrapolando nuevas muestras en lugar de duplicarlas como hace ROS. SMOTE selecciona una muestra de la clase minoritaria y de entre sus vecinos más cercanos selecciona uno y crea una nueva muestra entre ambos. De tal forma, esta técnica evita el sobreajuste que ocurre cuando se agregan réplicas exactas de muestras minoritarias al conjunto de datos principal.

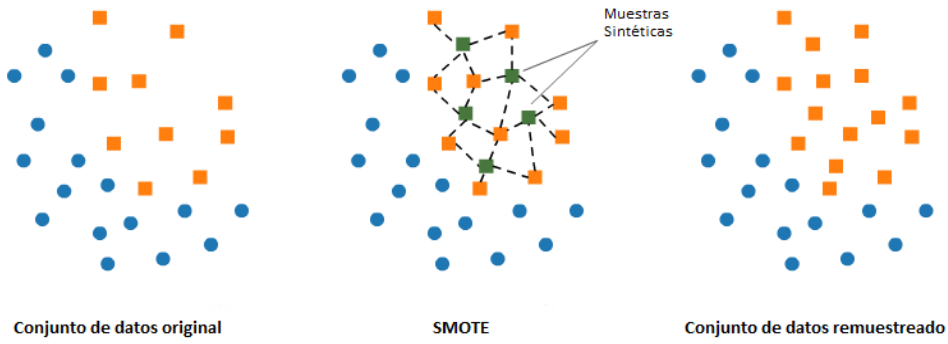


Figura 3.5. *SMOTE (Synthetic Minority Oversampling Technique).*

En la Figura 3.5, se puede ver en el lado izquierdo el conjunto de datos original. En el centro, se crean las muestras sintéticas pertenecientes a la clase minoritaria. En el lado derecho, las nuevas muestras son agregadas al conjunto de datos original y es el que se va a utilizar para entrenar el modelo de clasificación.

Ventajas:

- Mitiga el problema del sobreajuste causado por el ROS, ya que se generan muestras sintéticas en lugar de la replicación.
- Sin pérdida de información útil.

Desventajas:

- No es muy efectivo para datos de alta dimensión.
- Presenta un problema conocido como "*generalización excesiva"*, y se atribuye en gran medida a la forma en cómo crea muestras sintéticas. Al generar estas, no tiene en cuenta las muestras vecinas de otras clases. Esto puede dar como resultado un aumento en la superposición de clases y puede introducir ruido adicional.

No obstante, SMOTE es un algoritmo especialmente eficiente para conjuntos de datos altamente desequilibrados y su rendimiento puede mejorarse combinándolo con técnicas de *Undersampling*. Por consiguiente, también ha servido de base para el desarrollo de otras técnicas de muestreo como *"Bordeline SMOTE"*, *"Adaptative Synthetic Sampling (ADASYN)"*, *"SMOTE Editing Nearest Neighbor"*, entre otros. A continuación, se explicará solo ADASYN.

3.1.2.4 Muestreo Sintético Adaptativo (ADASYN)

El *"Muestreo Sintético Adaptativo"* (del inglés *"Adaptive Synthetic Sampling Approach for Imbalanced Learning"*, *ADASYN*), descripto por He y Garcia (2009), es una extensión de SMOTE.

ADASYN consiste en generar más muestras sintéticas en las zonas fronterizas para "aliviar" el problema del solape entre clases, pero propone un método sistemático para crear un número distinto de muestras sintéticas según la posición de cada muestra de la clase minoritaria.

Básicamente, tiene dos objetivos: el primero es crear muestras sintéticas a través de la interpolación lineal, entre las muestras de la clase minoritaria para reducir su desequilibrio con la clase mayoritaria del conjunto de datos. El segundo objetivo que hace diferente a ADASYN respecto a SMOTE, es que las muestras generadas cambian adaptativamente el límite de decisión agregando muestras en la zona de la clase minoritaria que es difícil aprender y se realiza a través de una distribución de densidad. Sin embargo, las muestras de la clase minoritaria que son fáciles de aprender no son alcanzadas por dicha técnica, es decir, ADASYN busca darle mayor peso a los datos de la clase minoritaria que son difíciles de captar.

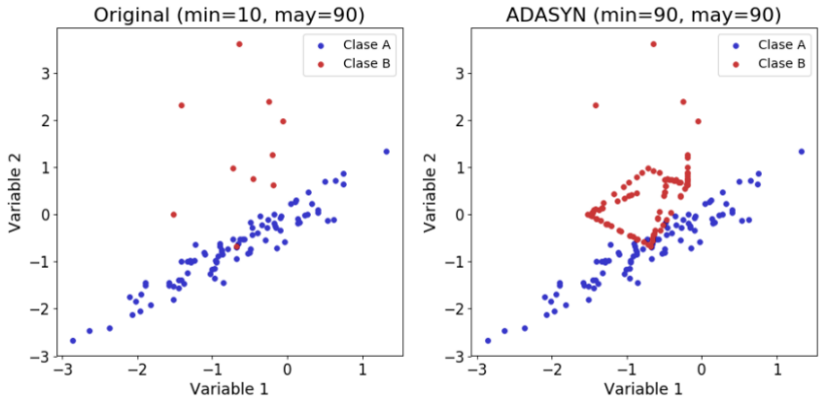


Figura 3.6. *ADASYN, "Adaptive Synthetic Sampling Approach for Imbalanced Learning".*

En la Figura 3.6, se puede observar el resultado de aplicar el algoritmo ADASYN a un conjunto de datos original. En ella, se puede comprobar que el funcionamiento de este algoritmo es distinto al de todos los anteriores. Basta con ver el gráfico de la izquierda, donde la muestra situada aproximadamente en el punto (-0.7, -0.7) podría ser tratada como ruido. No obstante, como se mencionó, la idea

principal de este algoritmo es la de usar la densidad para decidir cuántas muestras sintéticas deben generarse, con el fin de afianzar la presencia de las muestras minoritarias en lugar de desecharlas como ruido. Por esta razón, se puede visualizar que en el gráfico de la derecha la muestra del punto (-0.7, -0.7) tiene mayor densidad de muestras sintéticas a su alrededor.

3.1.3 Algoritmos Híbridos

A pesar de que tanto Oversampling como Undersampling logran buenos resultados por separado, es posible combinar ambas técnicas descriptas, donde existen cuantiosas literaturas que han demostrado obtener buenos resultados. Entre las cuales se pueden destacar:

- ***SMOTE + Tomek Links*:** inicialmente se realiza el Oversampling con la clase minoritaria y luego se aplica Tomek Link a ambas clases (Batista, 2002).

- ***SMOTE + ENN*:** es similar a SMOTE + Tomek Links. ENN tiende a eliminar más muestras que los que hace Tomek Links, por lo que se espera que proporcionará una mayor depuración de los datos en profundidad (Batista, 2002).

- ***CNN + Tomek Links*:** es similar a la selección de un solo lado (One-Sided Selection -OSS-), pero el método CNN se aplica para encontrar el subconjunto consistente antes de aplicar el método Tomek Links.

- ***SMOTE + RSB:*** es un nuevo método híbrido para procesamiento previo de conjuntos de datos desbalanceados que construye muestras nuevas, utilizando *SMOTE* más la teoría de los conjuntos aproximados (Ramentol, 2009).

3.2 Aprendizaje sensible al costo

No todos los métodos requieren de modificar el conjunto de datos. Otra manera de abordar el problema del desbalance de clases es mediante el método de *"Aprendizaje sensible al costo"*. Algunas técnicas para llevar adelante este método se basan en modificar los umbrales de decisión o asignando pesos a las clases en función de los costos.

La técnica de movimiento de umbral, aunque no es tan conocida como Undersampling y Oversampling, es simple y ha mostrado cierto éxito para los datos desequilibrados de dos clases. El umbral nos permite decidir a qué clase pertenece una muestra. Generalmente, se utiliza un valor de 0.5, pero puede ser algún otro valor. Si la probabilidad es mayor o igual a ese valor, entonces la muestra pertenece a la clase positiva. Caso contrario, corresponde a la clase negativa. Distintos valores de umbral generan diferentes cantidades de una y otra clase. Podemos usar esos valores para generar una gráfica. Esta gráfica se denomina *"Curva ROC" y* se explicará más adelante. Por otra parte, el movimiento de umbral funciona bien en conjuntos de datos que están extremadamente desbalanceados. El problema de desbalance de clases en tareas multiclase es mucho más complicado, donde Oversampling y el desplazamiento del umbral son menos efectivos (Morgan, 2012).

Otro tipo de enfoque consiste en tener en consideración los costos de los distintos errores que se pueden cometer en una clasificación durante la construcción del modelo, es decir, en la manipulación de la función de costos. Por ejemplo, es posible agregar costos para obligar al modelo a prestar atención a la clase minoritaria. Hay versiones penalizadas de algoritmos, como *"Máquinas de vector soporte"* (del

inglés *"Support-Vector Machines"*, SVM). Estos clasificadores intentan aprender más características de la clase minoritaria estableciendo un alto costo a los errores de clasificación de dichas muestras.

Como se puede observar, las técnicas Costo-Sensitivas proporcionan una alternativa viable a los métodos de Remuestreo para dominios de clasificación de datos desbalanceados.

3.3 Métodos de ensamble

Los clasificadores basados en ensambles son también denominados *"múltiples sistemas clasificadores"* y es conocido como la técnica de usar múltiples algoritmos de aprendizaje para obtener un mejor rendimiento de predicción que el que se pudiese obtener con cada algoritmo de forma independiente. Estas técnicas están divididas en dos categorías:

- ***Ensambles secuenciales*:** *"Boosting"* es el más común y más efectivo método de ensamble. El primer algoritmo de Boosting aplicado al desbalance de clases fue *Adaboost*. En este tipo de ensambles las muestras que no se asignan a la clase correcta reciben pesos más altos, lo que obliga a un futuro clasificador a concentrarse más en el aprendizaje de estas muestras clasificadas fallidas.

- ***Ensambles paralelos*:** se refiere a modelos de ensamble en los cuales cada clasificador base puede ser entrenado en paralelo. Uno de los modelos más implementados es el denominado *"Bagging"*, el cual ya se ha explicado en el capítulo II, sección 2.2.11.

Si bien estas técnicas suelen mostrar buenos resultados, tienen la desventaja de tener alto costo computacional debido a que entrenan diversas máquinas de clasificación. A pesar de ello, funcionan relativamente bien para problemas de dos clases, no ocurre así en problemas multiclase. Por otra parte, se demostró empíricamente que las técnicas de Desplazamiento de umbral y de Ensamble superan a Oversampling y Undersampling (Morgan, 2012).

3.4 Autoencoders

Como se ha mencionado en capitulo III, sección 2.3.6, un Autoencoder, es una red neuronal del tipo feedforward, con la particularidad de que posee una capa de entrada, otra de salida y una o más capas intermedias que se denomina *"código"*, es decir, entre la entrada y el código puede haber más capas, al igual que entre código y la salida. Básicamente, consiste en una función de *codificación* y una de *decodificación*. Esta última función produce una reconstrucción de la entrada a partir del *código* generado por la función de *codificación*. Durante el entrenamiento un Autoencoder será entrenado con el fin de poder reproducir la misma información de entrada en la salida.

Situando a los Autoencoders en el contexto del problema de clases desbalanceadas, precisamente, pueden ser usado para generar características que proporcionen una mejor capacidad de clasificación hacia la clase minoritaria. La red tiene capas ocultas que pueden ser usadas como nuevas representaciones de los datos de entrada, puesto que es posible reconstruir estos datos a partir de estas capas. Así mismo, dada su habilidad de aprender nuevas características, un Autoencoder puede ser utilizado para reducir la dimensionalidad de un conjunto de datos. Esto se lograría usando la salida de la capa de codificación como un nuevo conjunto (*Véase Figura 3.7*).

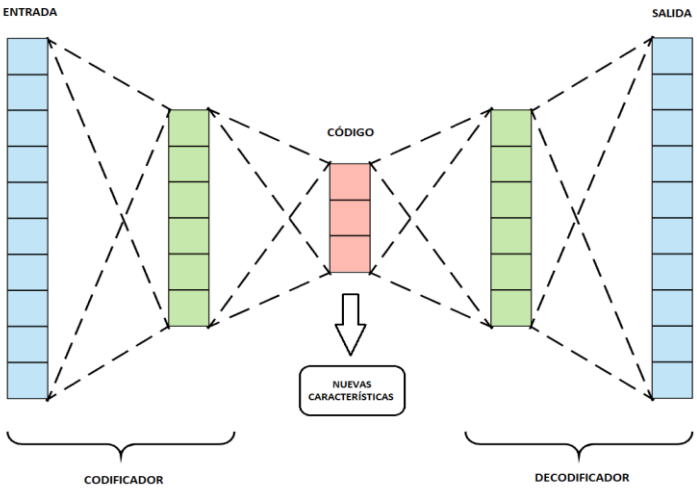


Figura 3.7. *Autoencoder para generar nuevas características.*

En la literatura, existen numerosos trabajos que demuestran que el uso de Autoencoders permite generar un conjunto con nuevas características y así obtener mayor precisión en la clasificación. A continuación, se mencionan precisamente dos trabajos, donde se han utilizado Autoencoders para resolver el problema de clases desbalanceadas. En ambos casos se utilizan "*Dual Autoencoders*", con la diferencia de que en el segundo trabajo se adiciona "*Denoising Autoencoders*".

El primer trabajo (Ala W.Y. Ng, Guangjun Zeng, Jiangjun Zhang, Daniel S. Yeung, Witold Pedrycz, 2016), propone un método de aprendizaje de características, basado en Autoencoders para aprender un conjunto con mejores capacidades de clasificación de las clases mayoritarias y minoritarias. Dos conjuntos de características son aprendidos por dos Autoencoders apilados, denominados *"Stacked Autoencoders"*, con diferentes funciones de activación (sigmoid y tanh), para capturar diferentes características de los datos y se combinan para formar las funciones de codificación automática dual. Luego, las muestras se clasifican en el nuevo espacio de características aprendido. Los resultados experimentales muestran que la propuesta supera al método de Remuestreo con significancia estadística para problemas de clasificación de patrones desequilibrados.

Otra de las investigaciones presentadas (Ting Wang, Guangjun Zeng, Wing W. Y. Ng, Jinde Li, 2017), propone un acercamiento muy similar al descrito anteriormente, con la diferencia de que se recibe como entrada una versión de los datos corrompidos y el Autoencoder es entrenado para generar los datos originales. Este tipo de Autoencoder es denominado *"Denoising Autoencoders"* y permite reducir el ruido dual. De igual manera al primer trabajo, se combinaron las características generadas por dos Stacked Autoencoders, uno utilizando la función de activación sigmoid y el otro tanh. Finalmente, al comparar los diversos resultados experimentales, se pudo determinar que utilizar las características generadas por el Denoising Autoencoder permitió obtener mejores resultados frente a otras técnicas de balanceo de clases.

Tal como se ve en ambas investigaciones, los Autoencoders pueden ser utilizados para generar nuevas características que cuenten con una mejor capacidad de clasificación aun cuando se tienen clases desbalanceadas. Sin embargo, en la práctica, se recomienda probar y comparar el comportamiento de distintas técnicas descritas a lo largo de todo el capítulo y escoger la más apropiada para cada caso, ya que no todos los conjuntos de datos se comportan de igual manera ni tienen las mismas propiedades.

En este trabajo, se desarrollará una experimentación con los algoritmos Random Oversampling, NearMiss, SMOTE para obtener distintos conjuntos de datos que serán utilizados por dos clasificadores (Random Forest y Backpropagation). De esta manera se espera poder visualizar los rendimientos de los diversos modelos y evaluar el impacto de los algoritmos de balanceo mencionados. A su vez, se diseñará un modelo basado en Autoencoders como alternativa al tratamiento de clases desbalanceadas. A continuación, en el capítulo IV, se describen en profundidad los pasos llevados a cabo para lograr lo mencionado.

Capítulo IV
Experimentación

El presente trabajo está enfocado en la investigación y aplicación de técnicas de balanceo de clases, por ello, en este capítulo se emplean algunas de las expuestas en el capítulo III en un caso real. Bajo el proceso de *Descubrimiento de Conocimiento en Base de Datos (KDD)*, cada técnica de balanceo de clases dará lugar a diferentes versiones del conjunto de datos original para la misma representación, luego, para cada variante se aplicarán las mismas técnicas predictivas. Con esto se busca obtener modelos que puedan ser contrastados y por consiguiente elaborar un cuadro comparativo que permita visualizar los impactos de las técnicas de balanceo en los distintos enfoques, midiendo *Accuracy, Precision, Recall y F1-Measure*. Cabe mencionar que, el alcance comparativo comprende tanto las técnicas de balanceo como el caso del conjunto de datos original desbalanceado. Sin embargo, antes de comenzar a examinarlas y poder visualizar su impacto, se presentará, en base al diseño experimental, un análisis exploratorio del conjunto de datos y la descripción de las prácticas realizadas sobre este.

Por otra parte, todos los algoritmos involucrados en dicha experimentación fueron implementados en el lenguaje de programación Python y pueden ser consultados en el Apéndice de este trabajo.

4.1 Diseño experimental

Desde el punto de vista teórico se considerará para la elaboración de los modelos predictivos el proceso KDD para obtener un alto grado de eficacia, por tal motivo, el diseño experimental es directo de lo expuesto en el capítulo II, sección 2.1.5, que, siguiendo los pasos descritos, se detallan las siguientes etapas realizadas: selección, preprocesamiento, minería de datos e interpretación y evaluación.

4.2 Selección

El primer paso consiste en identificar las fuentes relevantes de datos y consolidarlos en un repositorio único para la construcción de los modelos de clasificación. Este paso fue directo, puesto que todas las bases de datos ya estaban consolidadas.

4.3 Preprocesamiento y limpieza

En esta etapa se lleva a cabo un análisis del conjunto de datos, selección de atributos, toma de decisiones respecto a valores faltantes, atípicos y erróneos, transformación de datos, estandarización de atributos y balanceo de datos. Dichas tareas tienen la finalidad de transformar los datos recolectados en una *“vista minable”, es decir,* consolidar en una única tabla todas las muestras y atributos sobre los que se aplicarán los algoritmos de minería de datos. A continuación, se presentará con mayor detalle cada una de estas tareas.

4.3.1 Análisis del conjunto de datos

Antes de entrenar un modelo predictivo, o incluso antes de realizar cualquier cálculo con un nuevo conjunto de datos, es muy importante realizar una exploración descriptiva de los mismos. Este proceso permite entender mejor qué información contiene cada atributo, así como detectar posibles errores.

En el proceso de experimentación se utilizará la base de datos *"Afiliados",* proporcionada por una compañía de cobertura médica. Los datos contenidos se refieren a datos personales, demográficos, reclamos, autorizaciones, reintegros, consultas, facturación y obra social de los afiliados a dicha compañía.

Para llevar a cabo el experimento se procederá a trabajar con datos anonimizados, el cual está conformado por 113.089 registros, cuya distribución es la siguiente:

- **Clase Activo:** 87.749 beneficiarios activos correspondientes al mes de marzo de 2020. Representa el 77,60% de los casos.
- **Clase Baja:** 25.341 beneficiarios que dieron de baja el servicio en el año comprendido entre abril 2019 – marzo 2020. Representa el 22,40% de los casos.

El conjunto de datos cuenta con 67 atributos que se pueden agrupar de la siguiente manera:

- **Clase:** el atributo "*capita_baja"* será utilizado como la etiqueta de clase para la clasificación debido a que indica si el registro se trata de un Activo o Baja.
- **Datos personales:** edad, sexo, fecha de nacimiento, código postal, provincia y región.
- **Obra social:** identificador único de grupo familiar y beneficiario, número de socio, fecha de ingreso y egreso de la obra social, si posee o no cobertura, atributos del plan, parentesco, condición titular, cantidad de integrantes del grupo familiar, si está asociado a la prepaga de forma directa o por empresa, antigüedad del grupo familiar, si adquirió un servicio especial, si el grupo familiar posee servicios por discapacidad, quien realizó la venta del grupo familiar, si interesa retenerlo o no (por rentabilidad, juventud, etc.) y atributos referidos a la baja del beneficiario (pedida por la obra social, por fallecimiento, etc.).
- **Facturación:** importe neto facturado de cuota, si existió un aumento de cuota (aumento por ley y respecto al neto del mes anterior); en caso de existir aumento, indica si fue menor o mayor a 5%, 10% y 15%, precio como socio individual y precio preferencial de empresa, que tan rentable fue el beneficiario para la obra social (datos solo para bajas), estado del pago de la cuota, si la factura se remite a una empresa o al individuo y rentabilidad.
- **Autorizaciones, reintegros y consultas:** si existieron o no autorizaciones, reintegros y excepciones rechazadas en los últimos 3 meses, si tuvo o no autorizaciones, reintegros y excepciones emitidos en los últimos 3 meses. Si existieron o no consultas en los últimos 3 meses.
- **Casos abiertos:** estados abiertos o pendientes de resolución de casos y si el socio se comunicó para preguntar por un reclamo previo.

- **Reclamos:** cantidad de reclamos hechos por el titular en los últimos 2 y 6 meses en las áreas de Gestión de Clientes, Gestión de Recursos Financieros, Acceso al Servicio y Gestión de Proveedores.

Si bien originalmente el conjunto de datos cuenta con 67 atributos se limitará el uso a solo 22 de ellos, debido a que la entidad de cobertura médica prefiere no compartir estos datos. Por lo tanto, en la siguiente imagen se puede observar qué estructura tiene esta nueva base de datos de la que se dispone para la clasificación:

Orden	Atributo	Tipo	Descripción
1	id_beneficiario	Numérico	Identificador único.
2	adhesion	Categórico	A quien se emite la factura: I = Individuo; E = Empresa.
3	sexo	Categórico	Sexo del Titular.
4	edad	Numérico	Edad del Titular.
5	region	Categórico	Región
6	cartilla	Categórico	Familia de Línea del Titular. Classic/Genesis/Global/Premium es un agrupamiento de planes que comparten cartilla médica.
7	antig_mes	Numérico	La antigüedad del Grupo familiar en meses
8	discapacidad	Categórico	Indica si alguien del grupo familiar tiene marca de discapacidad.
9	deuda	Categórico	estado del pago de la cuota
10	aut_rech	Categórico	Indica si tuvo o no Autorizaciones Rechazadas en los 3 meses anteriores
11	reint_rech	Categórico	Indica si tuvo o no Reintegros Rechazados en los 3 meses anteriores
12	excep_rech	Categórico	Indica si tuvo o no Excepciones Rechazadas los 3 meses anteriores
13	aut_emi	Categórico	Indica si tuvo o no Autorizaciones Emitidas en los 3 meses anteriores
14	reint_emi	Categórico	Indica si tuvo o no Reintegros Emitidos en los 3 meses anteriores
15	excep_emi	Categórico	Indica si tuvo o no Excepciones Emitidas los 3 meses anteriores
16	reclamo_tot	Numérico	Cantidad de reclamos en los últimos 6 meses.
17	aumento_cuota	Categórico	Indica si hubo un aumento de cuota, con respecto al neto del mes anterior.
18	aumento_menor_5	Categórico	En caso de existir aumento, indica si el mismo fue menor a un 5%
19	aumento_entre_5_10	Categórico	En caso de existir aumento, indica si el mismo fue entre un 5% y 10%
20	aumento_mayor_10	Categórico	En caso de existir aumento, indica si el mismo fue mayor a 10%
21	cant_capita	Numérico	Cantidad de integrantes del grupo familiar.
22	capita_baja	Categórico	Clase

Figura 4.1. *Información de la base de datos.*

El conjunto de datos contiene un total de 113.089 registros. La Figura 4.1 presenta los 22 atributos. Además, se observa que existen datos numéricos y categóricos. El primer atributo representa un ID para identificar un registro del otro. Por otro parte, el último atributo identifica la clase a la que pertenece el registro, indicando un “SI” para *Baja* y un “NO” para *Activo*; y el resto de los campos intermedios representan las características descritas más arriba. En la Figura 4.2 se puede observar qué nivel de desbalance existe entre ambas clases.

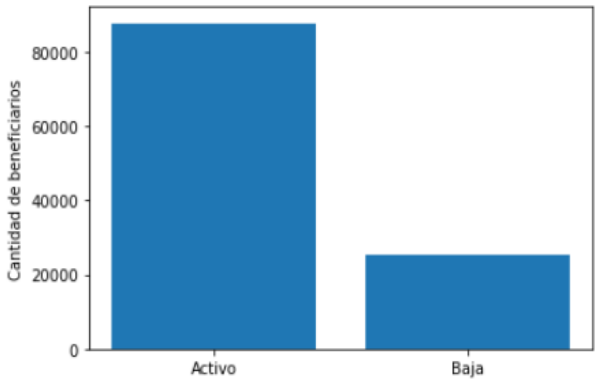


Figura 4.2. *Gráfico del balanceo de clases de la base de datos.*

Como se puede ver existe un claro desbalance entre los beneficiarios. Por lo tanto, el conjunto de datos es válido para llevar adelante el experimento que se desea realizar. A continuación, se podrá observar en detalle algunas de las variables y se realizarán modificaciones en ellas para mejorar el proceso de clasificación.

4.3.2 Datos faltantes

Muchos trabajos de Aprendizaje automático están destinados a fallar por la poca calidad de los datos con los que se cuenta. Cuando se recolectan datos del mundo real es muy común que algunas muestras tengan datos faltantes o incompletos, lo cual es fundamental tratarlo para no tener un efecto significativo en la clasificación.

Para evaluar faltantes en el conjunto de datos “*afiliados*” se utilizó la función “*isnull ()”* de la librería Pandas y su resultado fue "*False*", lo que indica que no existen valores nulos en el data set. Si el resultado hubiera sido “*True*”, es decir, existen valores nulos, se debería determinar qué hacer con ellos. Las alternativas serían: dejarlos como están, eliminarlos o inferir su valor. Cada una con sus ventajas y desventajas, no obstante, no se profundizará en este tema.

4.3.3 Duplicados

Tal como se mencionó en la sección 4.3.1, el primer atributo representa un ID para identificar un registro del otro. Por medio de este, se pudo detectar que 4.388 bajas y 22.093 activos estaban duplicados, por lo que se procedió a removerlas para evitar contar con datos redundantes. Por tanto, luego de esta operación, el conjunto de datos quedó conformado por un total de 86.610 registros.

4.3.4 Boxplot

Luego de haber tratado datos faltantes y duplicados, se realiza la detección de outliers o valores atípicos. Un outlier es un valor que numéricamente es muy diferente al resto de los datos, lo que puede afectar los resultados, por lo que es aconsejable retirarlos antes de seguir adelante con el mismo.

Mediante el diagrama de cajas se pueden identificar estos casos y en el conjunto de datos “Afiliados” se analizan los siguientes atributos: *“edad”, “antig_mes”, “cant_capita” y “reclamo_tot”*.

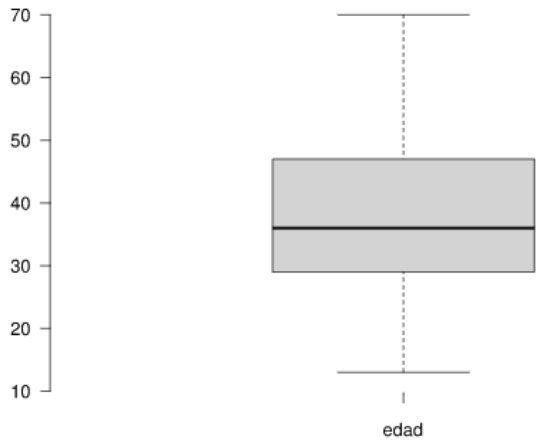


Figura 4.3. *Diagrama de caja atributo “edad”.*

La Figura 4.3 muestra que la caja está representada por el rango intercuartil de 29 a 47 años, siendo la edad mediana 36 años. También se puede ver que existen muestras con edades inferiores a 18 años y lo cual incumple con la condición de un afiliado titular. Por esta razón, se decide remover 9 muestras con la finalidad de prescindir de dichos datos para no distorsionar los resultados.

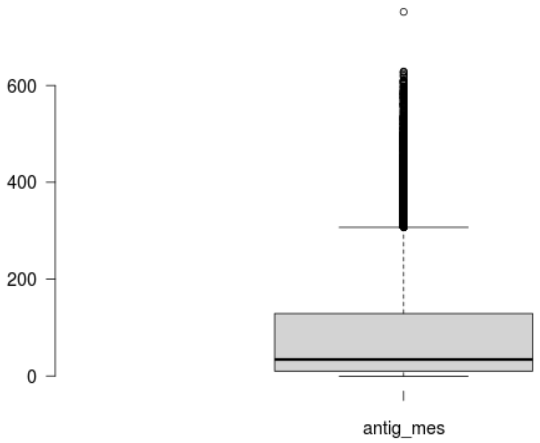


Figura 4.4. *Diagrama de caja atributo “antig_mes”.*

El Figura 4.4 muestra que la caja está representada por el rango intercuartil de 10 a 129 meses, siendo la mediana 34 meses. También se puede ver que existen muestras con valor “*-1*” y lo cual no es un dato válido. Por otra parte, los afiliados que superan los 307 meses son casos realmente poco frecuentes por lo que constituyen valores atípicos extremos y sería prudente prescindir de ellos para no alterar los resultados. De esta forma, el respectivo análisis conlleva a remover 118 muestras.

En cuanto a “*reclamo_tot*” y “*cant_capita*”, se detectaron outliers, pero se decidieron no modificarlos por su baja proporción y tratarse de datos muy particulares de un afiliado. De todas maneras, a continuación, se presentan sus respectivos diagramas:

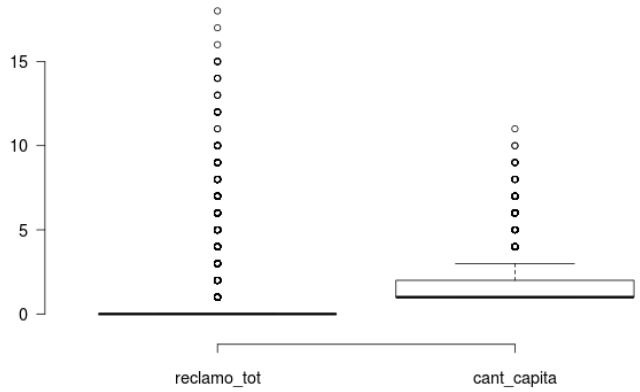


Figura 4.5. *Diagrama de caja atributos "reclamo_tot" y "cant_capita".*

De este modo se puede considerar que se han tratado los valores atípicos extremos y se conservan los valores atípicos leves. Ahora el conjunto de datos "*afiliados*" está conformado por 86.483 registros.

4.3.5 Matriz de correlación

La "*Matriz de correlación"* muestra los valores de correlación de Pearson, que miden el grado de relación lineal entre cada par de atributos numéricos. Para que exista una relación fuerte el valor debe ser igual o mayor a 0,80. Si observamos la Matriz de correlación de la siguiente *Figura 4.6*, se puede destacar que "*edad*" y "*antig_mes*" tienen una correlación lineal débil (0.55), sin embargo, es insuficiente como para prescindir de alguno de los dos atributos. Por lo tanto, se puede concluir que ninguna de las variables está relacionada estrechamente con ninguna otra.

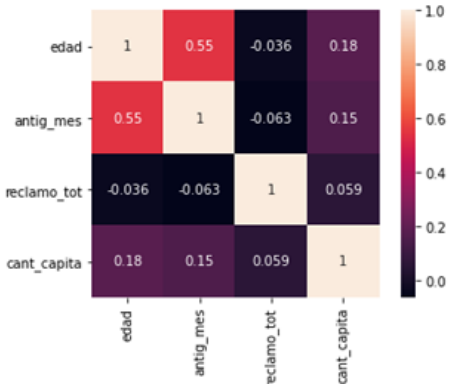


Figura 4.6. *Matriz de correlación.*

4.3.6 Transformaciones

Si se observa nuevamente el conjunto de datos del problema: *afiliados* (*Véase Figura 4.1*), se ve que existen valores numéricos y categóricos. Al trabajar en la experimentación con Redes neuronales es necesario realizar una "*Numerización"* de aquellos atributos que no son numéricos. Para ello, los campos que posean valores binarios y son nominales, serán codificados como "0" y "1" representando los siguientes valores:

Atributo	Numerización
Adhesion	"I":0; "E":1
Sexo	"F":0; "M":1
Discapacidad	"NO":0; "SI":1
deuda	"NO":0; "SI":1
aut_rech	"NO":0; "SI":1
reint_rech	"NO":0; "SI":1
excep_rech	"NO":0; "SI":1
aut_emi	"NO":0; "SI":1
reint_emi	"NO":0; "SI":1
excep_emi	"NO":0; "SI":1
aumento_cuota	"NO":0; "SI":1
aumento_menor_5	"NO":0; "SI":1
aumento_entre_5_10	"NO":0; "SI":1
aumento_mayor_10	"NO":0; "SI":1
capita_baja	"NO":0; "SI":1

Figura 4.7. *Atributos Numerizados.*

Para el caso de los atributos "*region*" y "*cartilla*", el caso es distinto dado que, si codificamos como "0", "1", "2", "3", "4" y "5" por ejemplo, la región con el valor "2" tendría más peso que la región con el valor "0" por lo que la estrategia para este tipo de atributos es diferente. Dentro de las transformaciones lógicas, los atributos de intervalo se pueden convertir en ordinales o en nominales, y crear atributos ficticios o "*dummy*".

Por consiguiente, los atributos "*region*" y "*cartilla*" serán modificados por seis y cuatro columnas nuevas respectivamente, es decir, misma cantidad de columnas que valores posibles. De esta forma quedarían de la siguiente manera:

Atributo	Valor	Dummies
Region	Cuyo	region_Cuyo
	Metro AMBA	region_Metro_AMBA
	Noreste	region_Noreste
	Noroeste	region_Noroeste
	Pampeana	region_Pampeana
	Patagonia	region_Patagonia
Cartilla	C (Classic)	cartilla_C
	GE (Genesis)	cartilla_GE
	GL (Global)	cartilla_GL
	P (Premium)	cartilla_P

Figura 4.8. *Nuevos atributos dummies.*

Dado que los datos de "*edad*" y "*antig_mes*" mantienen una escala diferente y en las ecuaciones de algún método de clasificación, dada la distancia euclidiana entre dos puntos, el valor de "*antig_mes*" podría hacer que "*edad*" dejara de ser representativa o importante para el análisis. Por esta razón, lo más conveniente es hacer una transformación de escalas, ya sea una "*Normalización*" o una "*Estandarización*" de los datos.

4.3.7 Estandarización

Es muy importante considerar el *escalado* que poseen los atributos numéricos. En pocas palabras, al hablar de escalado se refiere a poner todos los atributos en la misma escala para que ninguno esté dominado por otro. Como se mencionó en la sección anterior, las dos técnicas más habituales para escalar datos numéricos antes del modelado son la *"Normalización"* y la *"Estandarización",* y el propósito de ambas es mejorar la velocidad de convergencia y precisión del modelo.

Precisamente, "*Normalizar"* es escalar los datos desde sus valores originales a un rango entre 0 y 1, por lo tanto, se requiere conocer cuál es el valor mínimo y máximo del universo para realizar la operación. Por otra parte, "*Estandarizar"* se refiere a escalar la distribución de los datos de forma tal que la media de los valores observados sea igual a 0 y su desviación estándar igual a 1.

De esta manera, existen algunas diferencias notables entre ambos. En primer lugar, los valores obtenidos por la estandarización no están acotados en ningún rango (por ejemplo [0-1]). En segundo lugar, este método consigue solucionar el problema de valores atípicos u outliers que presenta la normalización.

En este experimento se realiza una transformación de los datos por medio de la *estandarización* a través de *"StandardScaler"*, paquete de sklearn.preprocessing. De esta forma, los outliers mostrados en la sección 4.3.4 para los atributos "*cant_capita*" y "*reclamo_tot*", , se ven "suavizados" por esta técnica y su impacto hacia los modelos es realmente despreciable.

4.3.8 Vista minable

El conjunto de datos luego de este preprocesamiento sufrió ciertos cambios con el objetivo de simplificar el estudio y obtener un modelo confiable, dejando solo los atributos de mayor peso y transformando los datos categóricos a numéricos. A continuación, se puede observar el estado natural de los datos luego del preprocesamiento:

Orden	Atributo	Valor
1	adhesion	Individuo = 0; Empresa = 1
2	sexo	Femenino = 0; Masculino = 1
3	edad	[18, 70]
4	region_Cuyo	No = 0; Si = 1
5	region_Metro_AMBA	No = 0; Si = 1
6	region_Noreste	No = 0; Si = 1
7	region_Noroeste	No = 0; Si = 1
8	region_Pampeana	No = 0; Si = 1
9	region_Patagonia	No = 0; Si = 1
10	cartilla_C	No = 0; Si = 1
11	cartilla_GE	No = 0; Si = 1

12	cartilla_GL	No = 0; Si = 1
13	cartilla_P	No = 0; Si = 1
14	antig_mes	[0, n]
15	discapacidad	No = 0; Si = 1
16	deuda	No = 0; Si = 1
17	aut_rech	No = 0; Si = 1
18	reint_rech	No = 0; Si = 1
19	excep_rech	No = 0; Si = 1
20	aut_emi	No = 0; Si = 1
21	reint_emi	No = 0; Si = 1
22	excep_emi	No = 0; Si = 1
23	reclamo_tot	[0, n]
24	aumento_cuota	No = 0; Si = 1
25	aumento_menor_5	No = 0; Si = 1
26	aumento_entre_5_10	No = 0; Si = 1
27	aumento_mayor_10	No = 0; Si = 1
28	cant_capita	[1, n]
29	capita_baja	No = 0; Si = 1

Figura 4.9. *Conformación de la vista minable.*

Sin embargo, antes de proceder a las técnicas de balanceo, se detectó que, del total de 86.483 muestras, 5.848 beneficiarios estaban presentes en ambas clases lo cual obligó a descartarlos para evitar alterar los resultados de cada modelo. Por lo tanto, se procederá a trabajar con **80.635** muestras únicas, distribuidas por 18.016 Bajas (22,35%) y 62.619 Activos (77.65%) *(Véase Figura 4.10).*

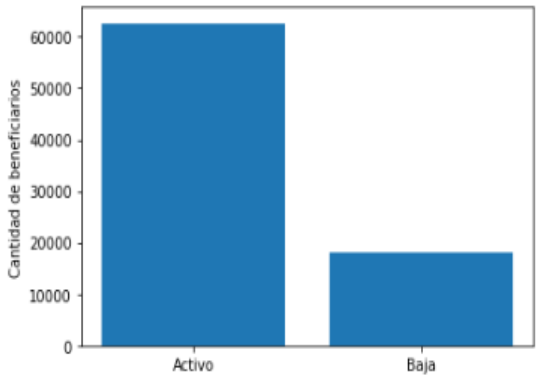


Figura 4.10. *Representación de clases de la vista minable.*

4.4 Aplicación de técnicas de balanceo

Los métodos de Remuestreo explicados en el capítulo III, involucran dos enfoques de muestreo que aumentan o disminuyen muestras de alguna de las clases, según sea el caso, para modificar las proporciones del conjunto de datos original. Por tanto, en este trabajo se seleccionan los algoritmos más representativos de cada una de las técnicas que se expusieron: muestras aleatorias, sintéticas y cercanas. Los mismos serán desarrollados utilizando la librería "*imbalanced-learn"* con las configuraciones recomendadas. Esto debe, a que se han probado con diferentes hiperparámetros para cada algoritmo y se han obtenido resultados similares respecto a los valores sugeridos por la librería.

Para muestras aleatorias se selecciona respectivamente el algoritmo de Sobremuestreo con las siglas ROS *(*Random Oversampling*).* Dicho algoritmo se emplea mediante la generación aleatoria de 44.603 muestras, adicionales a las 18.016 ya existentes, esto significa que se elimina la diferencia entre ambas clases sin ningún tipo de información más que la de pertenecer a la clase minoritaria. Para llevar adelante lo descrito, se utiliza la función **RandomOverSampler()**.

Para muestras sintéticas se utiliza el algoritmo de *Sobremuestreo* con las siglas SMOTE (Synthetic Minority Oversampling Technique*).* En el algoritmo SMOTE se emplea mediante la generación de 44.603 muestras sintéticas, adicionales a las 18.016 ya existentes, esto significa que incrementa las muestras de la clase minoritaria al mismo valor que la clase mayoritaria. Para llevar adelante lo mencionado, se utiliza la función **SMOTE ()**. Emplear los hiperparámetros por defecto conlleva a utilizar $k = 5$, es decir, la generación de muestras sintéticas se da a partir de las 5 muestras más cercanas a cada una de ellas.

Para muestras cercanas se utiliza el algoritmo de Submuestreo NearMiss. Este algoritmo reduce las muestras de la clase mayoritaria al mismo valor que la clase minoritaria. Para llevar adelante lo mencionado, se utiliza la función **NearMiss ()**. Los hiperparámetros por defecto conlleva a utilizar versión = 1 y k_vecinos = 3, es decir, selecciona para remover aquellas muestras de la clase mayoritaria cuya distancia media a las 3 muestras más cercanas de la clase minoritaria es menor.

Por último, también se utilizará Autoencoders con la finalidad de que el modelo sea capaz de aprender una de las clases, por lo general la minoritaria, para luego, mediante la elección adecuada de un umbral sobre los errores de reconstrucción de las muestras de ambas clases, permita decidir a qué clase pertenece una muestra. En la siguiente sección se puede ver con mayor detalle su arquitectura y configuración paramétrica.

4.5 Minería de datos

Una vez que el conjunto de datos fue procesado con las técnicas para corregir el desbalance de clases, la siguiente etapa comprende la elección de los algoritmos de Minería de datos. En el presente trabajo, se entrenaron y probaron dos modelos predictivos para detectar la clase perteneciente de cada beneficiario. El primero está basado en Árboles de decisión -Random Forest- y el segundo basado en Redes neuronales -Backpropagation-. Adicionalmente se utilizan los Autoencoders, que, si bien no son modelos predictivos, se los utiliza con ese fin dada su capacidad de entrenarse con las muestras de una sola clase junto con la capacidad de obtener un error de reconstrucción de muestras. Para lo cual se establece un umbral que determina que un error por debajo del umbral implica una muestra de la clase minoritaria, mientras que un error por encima del umbral implica que la muestra

pertenece a la otra clase. Con este criterio es posible usar a los Autoencoders como modelos predictivos de clasificación.

Cada algoritmo de Minería de datos va a ser entrenado con los tres muestreos obtenidos por los métodos de Remuestreo, con lo cual se consideran 6 modelos diferentes. Además de ellos, se evalúa el rendimiento de los Autoencoders y también el caso de la base de datos original y desbalanceada, es decir, sin aplicar ninguna técnica de balanceo. De esta manera, el número de evaluaciones se incrementa a 9. Por otra parte, cabe destacar que cada modelo va a ser ejecutado 30 veces para obtener un promedio general de sus rendimientos y utilizar estos valores para analizar los resultados finales. En la Tabla 4.11, se observan los modelos a diseñar de los cuales aquellos etiquetados con "*****" utilizarán la vista minable sin balancear.

ALGORITMO		TÉCNICA
Random Forest		S/ Balance *
		Random Oversampling
		NearMiss
		SMOTE
RNA	**Backpropagation**	S/ Balance *
		Random Oversampling
		NearMiss
		SMOTE
		Autoencoders *

Tabla 4.11. *Representación de los modelos a implementar y evaluar.*

Para cada conjunto de datos, los conjuntos de entrenamiento y testeo fueron generados usando "*validación simple*". Donde el conjunto de entrenamiento está conformado por el 80% y el de prueba por el 20%. Por otra parte, del mismo modo que en la sección anterior, se han probado diferentes configuraciones de parámetros para cada modelo, sin embargo, se obtuvieron siempre resultados similares respecto a los valores sugeridos por la librería "*scikit-learn"*, por esta razón se procederá a aplicar las configuraciones recomendadas. A continuación, se describen los diversos modelos con sus respectivos hiperparámetros.

4.5.1 Red neuronal

Se construye para cualquiera de los conjuntos de datos una Red neuronal, utilizando el modelo de perceptrón multicapa que construye una red de retropropagación: Backpropagation. La red está completamente conectada, con 28 nodos en la primera capa, 100 nodos en la capa oculta y 1 nodo en la capa de salida. La elección del número de neuronas de la capa oculta sigue la configuración sugerida por defecto de la librería de scikit-learn. Por otra parte, los 28 datos de entrada corresponden a todos los atributos del conjunto de datos y 1 dato de salida que indica la clase a la que pertenece. A continuación, se presentan los hiperparámetros que se utilizan:

- Librería: sklearn.neural_network.MLPClassifier
- Función de optimización: SGD (descenso de gradiente estocástico)

- Tasa de aprendizaje: 0.0001
- Función de activación: Relu
- Tolerancia de optimización: 0.0001
- Momentum: 0.9
- Máximo de iteraciones: 200

4.5.2 Random Forest

Como se ha mencionado en el capítulo II, en esta técnica se construyen Árboles de decisión buscando en cada nodo del árbol el mejor "split" que separe los datos. Por consiguiente, el modelo depende de una serie de parámetros los cuales estarán configurados por los valores sugeridos por scikit-learn. Cabe mencionar, que la elección del número de árboles es fundamental en esta técnica, debido a que marca, en gran medida, la precisión y el tiempo de cálculo del algoritmo. La configuración adoptada de sus parámetros es:

- Librería: *sklearn.essemble.RandomForestClassifier*
- Número de árboles en el bosque: 100
- Criterios de división: GINI
- Número mínimo de muestras requeridas para dividir un nodo interno: 2
- Profundidad máxima del árbol: Ninguno (los nodos se expanden hasta que todas las hojas sean puras o hasta que todas las hojas contienen menos del número mínimo de muestras requeridas para dividir un nodo interno)

4.5.3 Autoencoders

Se utiliza un Autoencoder con la estructura mostrada en la Figura 4.12 para contribuir una alternativa al problema del desbalance de clases. El mismo tiene como objetivo poder aprender una de las clases. Por esta razón, se construyó un Autoencoder con 5 capas densas que están completamente conectadas y donde todas utilizan la función de activación tanh. Las capas contienen 28, 15, 10, 15 y 28 neuronas respectivamente. Las capas 2 y 3 se corresponden al *encoder*, y las capas 3 y 4 se corresponden al *decoder*. La primera y la última capa disponen de 28 neuronas, pues se corresponde con el número de atributos.

La función de optimización utilizada durante el entrenamiento es *'Adam'* y la función de pérdida elegida es el *'Error cuadrático medio'*, el cual se calcula como el promedio de las diferencias al cuadrado entre el valor original de sus atributos y el valor reconstruido, es decir, el promedio de los errores de reconstrucción de todos sus atributos elevados al cuadrado.

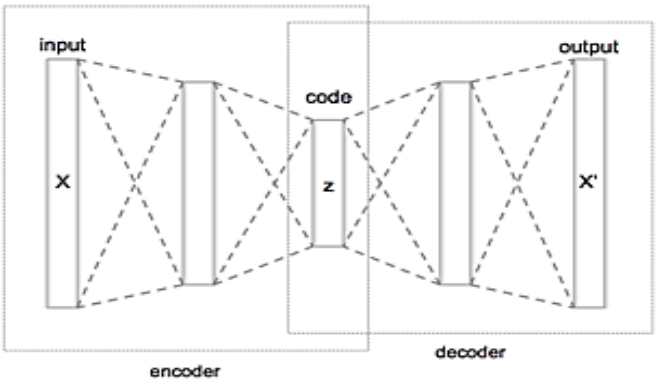


Figura 4.12. *Estructura diseñada para Autoencoder.*

4.6 Interpretación y evaluación

Luego de la etapa de entrenamiento, el siguiente paso es examinar el desempeño de los modelos de predicción obtenidos. Las métricas de evaluación de un modelo se utilizan para evaluar el poder pronóstico entre la salida de este y los datos, como así también, permitirán el contraste entre los modelos. A continuación, se presentan las métricas más utilizadas por los investigadores para llevar a cabo la evaluación de modelos:

4.6.1 Matriz de Confusión

La *"Matriz de confusión"* se usa para evaluar el rendimiento completo de un clasificador en donde la salida puede ser de dos o más clases. Esta métrica permite mostrar a través de una tabla los tipos de errores que se están cometiendo en el modelo. Su salida corresponde a una matriz de dimensión $N \times N$, en donde N es el número de clases que se están prediciendo y se realiza una comparación entre las predicciones obtenidas y los datos reales. Así, para el caso en que se tienen dos clases "A" y "B", el objetivo principal de la métrica es determinar el número de muestras de la clase "A" que fueron clasificadas dentro de la clase "B". En la Figura 4.13 se puede observar la forma estándar de una Matriz de confusión para N = 2, que se compone de: *verdaderos positivos (TP, por sus siglas en inglés), verdaderos negativos (TN, por sus siglas en inglés), falsos negativos (FN, por sus siglas en inglés) y falsos positivos (FP, por sus siglas en inglés).*

		PREDICCIÓN	
REAL		TP	FN
		FP	TN

Tabla 4.13. *Ejemplo de una matriz de confusión.*

Observar la Matriz de confusión una vez hechas las predicciones puede parecer algo insuficiente a la hora de medir la eficiencia de un modelo clasificando datos. A partir de esta matriz se pueden construir otras métricas como: *Accuracy*, *Recall*, *Precision*, *F1-Measure y espacio ROC*.

4.6.2 Exactitud (Accuracy)

Es la métrica más común para evaluar el desempeño de un modelo. Indica el número de muestras clasificadas correctamente en comparación con el número total de muestras. Tiene limitaciones, ya que no tiene un buen desempeño para conjuntos de datos desbalanceados, por lo que la Exactitud puede ser más alta de lo que realmente es. Puede calcularse a partir de la matriz de confusión como:

$$Accuracy = \frac{Predicciones\ correctas}{Muestras\ totales} = \frac{TP + TN}{TP + TN + FP + FN}$$

4.6.3 Precisión (Precision)

Representa la proporción de verdaderos positivos que son correctamente identificados en comparación con el número total de valores positivos que el modelo predijo. Una desventaja de la precisión es que, al igual que Accuracy, es sensible a los datos desbalanceados, es decir, si cambia la proporción de las muestras con una de

las clases, la Precisión informará un valor diferente cuando realmente el desempeño es el mismo. La expresión para la Precisión se obtiene como:

$$Precision = \frac{Verdaderos\ positivos}{Nº\ de\ positivos\ predichos} = \frac{TP}{TP + FP}$$

4.6.4 Sensibilidad (Recall)

Esta métrica indica la tasa de clasificación positiva correcta, es decir, la proporción de positivos que el modelo ha clasificado correctamente en función del número total de muestras positivas. Esta métrica tiene como ventaja que no es sensible a los datos desbalanceados. Por otra parte, es equivalente a la *Tasa de Verdaderos Positivos (TVP)*. La expresión para Sensibilidad se obtiene como:

$$Recall\ o\ TVP = \frac{Verdaderos\ positivos}{Positivos\ reales} = \frac{TP}{TP + FN}$$

La Precisión y Sensibilidad son dos medidas diferentes, pero con frecuencia se utilizan conjuntamente y se puede interpretar cuatro casos posibles para cada clase:

1. Alta Precisión y alta Sensibilidad: el modelo maneja perfectamente esa clase.

2. Alta Precisión y baja Sensibilidad: el modelo no detecta la clase muy bien, pero cuando lo hace es altamente confiable.

3. Baja Precisión y alta Sensibilidad: el modelo detecta bien la clase, pero también incluye muestras de otras clases.

4. Baja Precisión y baja Sensibilidad: el modelo no logra clasificar la clase correctamente.

Cuando se tiene un conjunto de datos desbalanceado, suele ocurrir que se obtiene un alto valor de Precisión en la clase mayoritaria y una baja Sensibilidad en la clase minoritaria.

4.6.5 F1 - Measure

El valor F se considera como una media armónica que combina los valores de Precisión y Sensibilidad, y que puede calcularse de forma general de la siguiente forma:

$$F_1 = (1 + \beta^2) \cdot \frac{Precisión \cdot Sensibilidad}{(\beta^2 \cdot Precisión) + Sensibilidad}$$

En este trabajo, la fórmula se reducirá a la siguiente expresión:

$$F_1 = 2 \cdot \frac{Precisión \cdot Sensibilidad}{Precisión + Sensibilidad}$$

en donde se toma β igual a uno, otorgándole la misma importancia o ponderación a Precisión y Sensibilidad. Si β es mayor que uno, se le daría mayor importancia a la Sensibilidad, mientras que si es menor que uno se le da más importancia a la

Precisión. El rango de valores de esta métrica de clasificación oscila entre 0 y 1. Un valor cercano a 1 representa un mejor desempeño del modelo.

4.6.6 Especificidad

La "*Especificidad"*, también llamada *"Tasa de Verdaderos Negativos (TVN)*", trata de las muestras negativas que el modelo ha clasificado correctamente, es decir, es la probabilidad de que un resultado negativo real sea negativo. Expresa cuán bien puede el modelo detectar esa clase. Su fórmula de cálculo es:

$$Especificidad = \frac{TN}{FP + TN}$$

4.6.7 Tasa de Falsos Positivos (TFP)

La "*Tasa de Falsos Positivos"* es la probabilidad de que se produzca una falsa alarma: que se dé un resultado positivo cuando el valor verdadero sea negativo. Su fórmula de cálculo es:

$$TFP = \frac{FP}{FP + TN}$$

4.6.8 Tasa de Falsos Negativos (TFN)

La "*Tasa de Falsos Negativos"*, también llamada "*tasa de error"*, es la probabilidad de que la prueba pase por alto un verdadero positivo. Se calcula como:

$$TFN = \frac{FN}{FN + TP}$$

4.6.9 Espacio ROC

Otra de las medidas más utilizadas en problemas de clasificación y que se caracteriza por ser algo más gráfica es calcular el valor bajo la *curva ROC* (Receiver Operating Characteristic), es decir, el *AUC (Area Under Curve)*. Para dibujar una curva ROC solo son necesarias las razones de verdaderos positivos (TVP) y de falsos positivos (TFP). Esta técnica genera curvas en lugar de una simple medida. Sin embargo, para comparar dos o más clasificadores se suele usar el área bajo la curva ROC como criterio de evaluación.

Un espacio ROC se define por TFP y TVP como ejes x e y respectivamente, y representa los intercambios entre verdaderos positivos (en principio, beneficios) y falsos positivos (en principio, costes). Dado que TVP es equivalente a Sensibilidad y TFP es igual a 1-Especificidad, el gráfico ROC también es conocido como la representación de Sensibilidad frente a (1-Especificidad). Cada resultado de predicción de la Matriz de confusión representa un punto en el espacio ROC (*Véase Figura 4.14*).

El mejor método posible de predicción se situaría en un punto en la esquina superior izquierda, o coordenada (0,1) del espacio ROC, representando un 100% de Sensibilidad (ningún falso negativo) y un 100% también de Especificidad (ningún falso positivo). A este punto (0,1) también se le llama una "*clasificación perfecta*". Por el contrario, una clasificación totalmente aleatoria daría un punto a lo largo de la línea diagonal, que se llama también "*línea de no-discriminación*", desde el extremo inferior izquierdo hasta la esquina superior derecha. La diagonal divide el espacio ROC. Los

puntos por encima de la diagonal representan buenos resultados de clasificación (mejor que el azar), y los puntos por debajo de la línea son resultados pobres (peor que al azar). Por ejemplo, en la Figura 4.14 se puede observar que los modelos A y C' son considerados buenos clasificadores. Sin embargo, B y C no tienen un buen rendimiento ya que B está sobre la línea de no-discriminación (azar) y C tiene un resultado insuficiente.

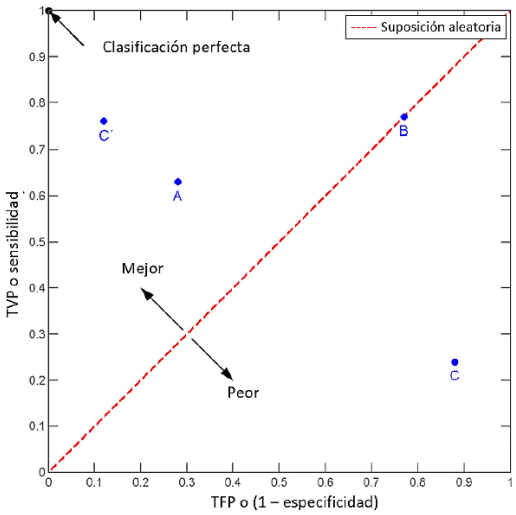


Figura 4.14. *El espacio ROC.*

Con el espacio ROC se concluye la explicación en cuanto a la evaluación de modelos de clasificación y en el siguiente capítulo se exhibirán los resultados obtenidos para cada uno de ellos. Tener en cuenta que cada métrica estará representada por el promedio general de 30 ejecuciones y también se presentarán los correspondientes desvíos estándar para cada una de ellas. En el Apéndice de este trabajo se pueden encontrar los resultados obtenidos en las 30 corridas para cada modelo.

Capítulo V
Resultados

En este capítulo se presentan los resultados de la experimentación. A su vez se puede notar cómo se comportan los distintos algoritmos al intentar clasificar con datos desbalanceados y balanceados. Se tomará como guía los resultados de la clasificación sin balancear para poder entender los obtenidos más adelante. Es importante señalar que todos los resultados presentados en este capítulo representan los *promedios* obtenidos al ejecutar 30 veces cada modelo. Adicionalmente, se ha calculado el *desvío estándar* para cada una de las métricas y va a mostrase su valor entre paréntesis. Por último, quiero recordar que se pueden encontrar en el Apéndice de este trabajo todos los resultados obtenidos en cada una de las ejecuciones para cada modelo, como así también el código correspondiente.

5.1 Sin balancear

Se realiza una comparación de dos algoritmos de Aprendizaje automático para identificar aquellos afiliados activos y bajas. Así mismo, el conjunto de datos fue proporcionado por la misma entidad de cobertura médica. Se realizó un preprocesamiento a los datos antes de aplicar dichos algoritmos, se removieron columnas que se consideraron innecesarias, no se apartaron muestras por valores faltantes, se removieron duplicados y se transformaron los atributos categóricos a numéricos. Cabe mencionar que después de este preprocesamiento, el conjunto de datos quedó desbalanceado, contando con:

- 62.619 muestras pertenecientes a la clase 0 (Activo)
- 18.016 pertenecientes a la clase 1 (Baja)

Los resultados obtenidos sin aplicar técnicas de balanceo son los siguientes:

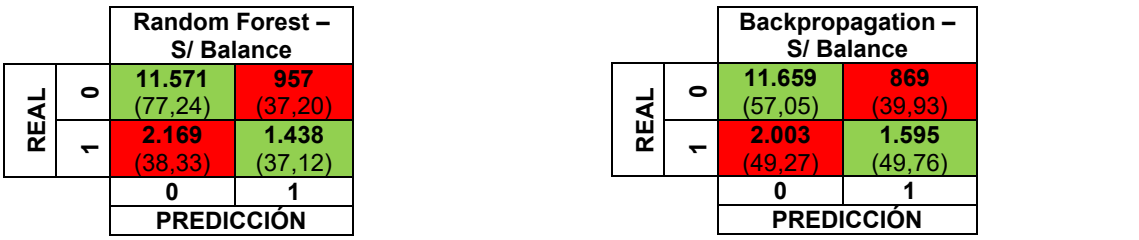
Algoritmo	**Validación**			
	Accuracy	**Precision**	**Recall**	**F1-Measure**
Random Forest	0,8066 (0,0048)	0,6006 (0,0048)	0,399 (0,0094)	0,4786 (0,0078)
Backpropagation	**0,82** (0,0026)	**0,6476** (0,0110)	**0,443** (0,0127)	**0,5263** (0,0093)

Tabla 5.1. *Resultados tras 30 ejecuciones con datos desbalanceados.*

En la Tabla 5.1, se puede observar en verde los mejores valores obtenidos para cada una de las métricas. Se puede ver que el *Accuracy* da la sensación de que todos los clasificadores hacen una clasificación similar y que, además, esta es bastante buena ya que ambos presentan aciertos en torno al 81%. Sin embargo, se puede ver que el *Recall* no supera el 45%, por lo que la clase minoritaria no es bien reconocida como tal. Este es el principal problema del desbalance de clases. Se obtiene buen *Accuracy* por la cantidad de muestras de clase mayoritaria que existen,

pero realmente está actuando de manera pobre con respecto a la clase minoritaria. Ninguno de estos modelos podría darse como bueno para una clasificación.

La Figura 5.2, presenta las *Matrices de confusión* correspondientes a cada modelo de la Tabla 5.1. Del mismo modo, los valores estarán representados por los promedios obtenidos tras 30 ejecuciones y entre paréntesis se muestran los desvíos estándar.



TVP = 11.571 / (11.571 + 957) = **0,9236** (0,0030)
TFP = 2.169 / (2.169 + 1.438) = **0,6013** (0,0089)

TVP = 11.659 / (11.659+869) = **0,9306** (0,0031)
TFP = 2.003 / (2.003+1.595) = **0,5567** (0,0118)

Figura 5.2. *Matrices de confusión obtenidas tras 30 ejecuciones.*

A continuación, se presentan los resultados de las técnicas de balanceo de clases: *Random Oversampling*, *NearMiss*, *SMOTE* y *Autoencoders*. Los mismos serán contrastados con la Tabla 5.1 y del mismo modo, los resultados representan 30 ejecuciones.

5.2 Random Oversampling

Se aplicó la técnica *Random Oversampling* para balancear el conjunto de datos y tratar de obtener mejores resultados. Después de aplicar esta técnica, la estructura del conjunto de datos quedó de la siguiente manera:

- 62.619 muestras pertenecientes a la clase 0 (Activo)
- 62.619 pertenecientes a la clase 1 (Baja)

Estos son los resultados obtenidos:

Algoritmo	**Validación**			
	Accuracy	**Precision**	**Recall**	**F1-Measure**
Random Forest	**0,9136** (0,0049)	**0,8723** (0,0043)	**0,97** (0)	**0,92** (0)
Backpropagation	0,7213 (0,0035)	0,81 (0,0059)	0,581 (0,0071)	0,677 (0,0047)

Tabla 5.3. *Resultados tras 30 ejecuciones con Random Oversampling.*

En la Tabla 5.3 se muestran en verde los mejores valores para cada una de las métricas. Si se compara estas con la Tabla 5.1 se ve que se ha conseguido mejorar el rendimiento de la clasificación en todas las métricas y en ambos algoritmos. Se puede apreciar un valor de *Recall* más alto en todos los modelos donde se destaca *Random Forest*. Por consiguiente, este modelo presenta el mejor resultado hasta el momento. Aunque el *Accuracy* de *Backpropagation* se ve empeorado respecto a la Tabla 5.1, ha

sido en muy poca medida en comparación con la mejora que ha supuesto en la detección de la clase minoritaria que se obtiene con el *Recall*. Además, el *F1-Measure* brinda una información global de que ambos clasificadores han mejorado notablemente. A continuación, se presentan las *Matrices de confusión* correspondientes:

		Random Forest – Random Oversampling	
REAL	0	**10.751** (62,86)	**1.765** (43,37)
	1	**372** (25,81)	**12.158** (58,20)
		0	**1**
		PREDICCIÓN	

TVP = 10.751 / (10.751+1.765) = **0,8590** (0,0033)
TFP = 372 / (372+12.158) = **0,0296** (0,0020)

		Backpropagation – Random Oversampling	
REAL	0	**10.797** (90,89)	**1.715** (67,56)
	1	**5.244** (94,49)	**7.291** (102,15)
		0	**1**
		PREDICCIÓN	

TVP = 10.797 / (10.797+1.715) = **0,8628** (0,0054)
TFP = 5.244 / (5.244+7.291) = **0,4183** (0,0074)

Figura 5.4. *Matrices de confusión obtenidas tras 30 ejecuciones.*

5.3 NearMiss

Se aplicó la técnica de *Undersampling NearMiss* para balancear el conjunto de datos y tratar de obtener mejores resultados. Después de aplicar esta técnica, la estructura del conjunto de datos quedó de la siguiente manera:

- 18.016 muestras pertenecientes a la clase 0 (Activo)
- 18.016 pertenecientes a la clase 1 (Baja)

Estos son los resultados obtenidos:

Algoritmo	Validación			
	Accuracy	**Precision**	**Recall**	**F1-Measure**
Random Forest	0,7343 (0,0068)	0,7396 (0,0081)	0,726 (0,0077)	0,7316 (0,0065)
Backpropagation	**0,77** (0,0037)	**0,762** (0,0066)	**0,783** (0,0098)	**0,7723** (0,0050)

Tabla 5.5. *Resultados tras 30 ejecuciones con NearMiss.*

Como se viene realizando, la Tabla 5.5 muestra en verde los mejores valores para cada una de las métricas. Se observa que con esta técnica se obtienen resultados inferiores para *Random Forest* y una mejora considerable para *Backpropagation* respecto a los que se han visto en *Random Oversampling*. Sin embargo, ninguno supera los resultados de *ROS* con *Random Forest*. Por otra parte, se ve una notable mejora con respecto a los valores de la clasificación inicial sin aplicar técnicas (Tabla 5.1). Adicionalmente, visualizando la métrica de *F1-Measure* se puede apreciar también esta mejora. A continuación, se presentan las *Matrices de confusión* correspondientes:

		Random Forest – NearMiss	
REAL	0	**2.684** (36,14)	**921** (33,24)
	1	**987** (26,73)	**2.613** (33,78)
		0	**1**
		PREDICCION	

TVP = 2.684 / (2.684+921) = **0,7444** (0,0020)
TFP = 987 / (987+2.613) = **0,2742** (0,0071)

		Backpropagation – NearMiss	
REAL	0	**2.730** (34,87)	**876** (24,98)
	1	**781** (30,78)	**2.819** (41,86)
		0	**1**
		PREDICCION	

TVP = 2.730 / (2.730+876) = **0,7569** (0,0064)
TFP = 781 / (781+2.819) = **0,2170** (0,0084)

Figura 5.6. *Matrices de confusión obtenidas tras 30 ejecuciones.*

5.4 SMOTE

Se aplicó la técnica de *Oversampling SMOTE* para balancear el conjunto de datos y tratar de obtener mejores resultados. Después de aplicar esta técnica, la estructura del conjunto de datos quedó de la siguiente manera:

- 62.619 muestras pertenecientes a la clase 0 (Activo)
- 62.619 pertenecientes a la clase 1 (Baja)

Estos son los resultados obtenidos:

Algoritmo	Validación			
	Accuracy	Precision	Recall	F1-Measure
Random Forest	**0,84** (0)	0,8456 (00,50)	**0,837** (0,0048)	**0,8396** (0,0018)
Backpropagation	0,7936 (0,0049)	**0,8693** (0,0052)	0,6913 (0,0063)	0,7703 (0,0056)

Tabla 5.7. *Resultados tras 30 ejecuciones con SMOTE.*

Se vuelve a mostrar en verde los mejores resultados para cada métrica. Se observa que con esta técnica también se obtiene una mejora en la clasificación en la misma medida que se ha ido viendo con el resto de las técnicas. Se consigue un buen *Recall* para *Random Forest* y un incremento del rendimiento global reflejado en la métrica *F1-Measure*. A continuación, se presentan las *matrices de confusión* correspondientes:

		Random Forest – SMOTE	
REAL	0	**10.586** (83,02)	**1.925** (37,47)
	1	**2.058** (57,24)	**10.477** (77,52)
		0	**1**
		PREDICCION	

TVP = 10.586 / (10.586+1.925) = **0,8461** (0,0029)
TFP = 2.058 / (2.058+10.477) = **0,1642** (0,0042)

		Backpropagation – SMOTE	
REAL	0	**11.231** (96,45)	**1.299** (59,50)
	1	**3.860** (84,03)	**8.656** (106,85)
		0	**1**
		PREDICCION	

TVP = 11.231 / (11.231+1.299) = **0,8962** (0,0047)
TFP = 3.860 / (3.860+8.656) = **0,3084** (0,0067)

Figura 5.8. *Matrices de confusión obtenidas tras 30 ejecuciones.*

5.5 Autoencoders

Para analizar y presentar los resultados del modelo de Autoencoder, es necesario establecer un umbral para el error de reconstrucción y así poder determinar si la respuesta del modelo es de una clase o la otra. Su elección no es un proceso trivial, y una ligera variación puede ocasionar grandes cambios en las predicciones, debido a esto se han realizado 30 ejecuciones con distintos valores de umbrales para obtener diferentes métricas e identificar cual de todas entre ellas posee el valor óptimo de umbral. A continuación, se presentan los resultados al ejecutar el modelo:

Exc.	AUTOENCODER				
	Accuracy	**Precision**	**Recall**	**F1**	**Umbral**
1	0,73	0,34	0,21	0,26	1.000
2	0,57	0,27	0,56	0,37	3.000
3	0,52	0,26	0,62	0,37	5.000
4	0,5	0,26	0,65	0,37	6.000
5	0,49	0,26	0,69	0,37	8.000
6	0,47	0,26	0,74	0,38	10.000
7	0,46	0,26	0,76	0,39	11.000
8	0,45	0,26	0,79	0,39	13.000
9	0,44	0,26	0,81	0,39	15.000
10	0,43	0,26	0,83	0,39	17.000
11	**0,42**	**0,25**	**0,84**	**0,39**	**19.000**
12	0,41	0,25	0,85	0,39	20.000
13	0,4	0,25	0,86	0,39	21.000
14	0,39	0,25	0,88	0,39	23.000
15	0,387	0,25	0,9	0,39	25.000
16	0,35	0,24	0,92	0,38	27.000
17	0,33	0,24	0,94	0,38	30.000
18	0,32	0,24	0,95	0,38	32.000
19	0,3	0,23	0,95	0,38	34.000
20	0,3	0,23	0,96	0,38	36.000
21	0,29	0,23	0,96	0,37	38.000
22	0,28	0,23	0,96	0,37	40.000
23	0,28	0,23	0,97	0,37	42.000
24	0,28	0,23	0,97	0,37	44.000
25	0,27	0,23	0,97	0,37	46.000
26	0,24	0,22	0,98	0,37	60.000
27	0,23	0,22	0,99	0,36	80.000
28	0,22	0,22	0,99	0,36	90.000
29	0,22	0,22	0,99	0,36	95.000
30	0,22	0,22	1	0,36	100.000

Tabla 5.9. *Resultados tras 30 ejecuciones con distintos umbrales.*

Como se puede observar no son buenos los resultados. Sin embargo, se puede identificar que en torno a la ejecución número 11 rondan los mejores valores de estas 30 ejecuciones. A pesar de ello y con la finalidad de evaluar el desempeño de los Autoencoders para este conjunto de datos, se buscará un valor de umbral tal que por debajo de este valor quede una alta proporción de muestras minoritarias y una baja proporción de la clase mayoritaria. Por lo tanto, en base a la Tabla 5.9 el valor de umbral que se elegirá es 19.000. En los siguientes gráficos, se examinará para este valor de umbral los errores de reconstrucción tanto para la clase mayoritaria como la minoritaria:

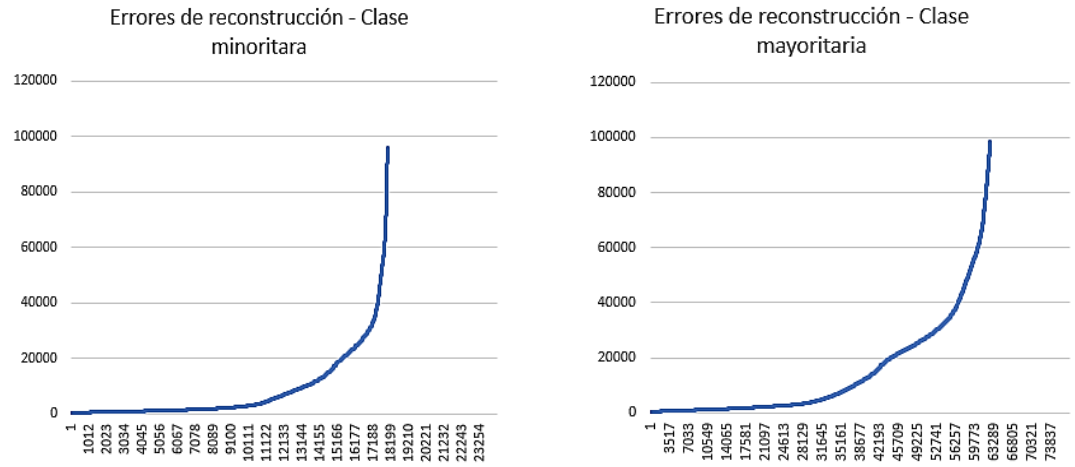


Gráfico 5.10. *Representación gráfica de errores de reconstrucción.*

Se observa que ambos presentan varias muestras con errores "bajos" y unas pocas muestras con errores "muy altos". Por consiguiente, siendo las muestras de la clase minoritaria un total de 18.016 y la clase mayoritaria 62.619, la matriz que permitirá evaluar el desempeño en el umbral es la siguiente:

Clase	Por debajo del umbral	Por encima del umbral
Minoritaria	**15.265** (85%)	**2.751** (15%)
Mayoritaria	**43.604** (70%)	**19.015** (30%)

Tabla 5.11. *Cantidad de muestras respecto al umbral.*

Como se puede ver, por debajo del umbral coexiste el 73% del total del conjunto de datos, 15.265 (85%) muestras de la clase minoritaria y 43.604 (70%) para la mayoritaria. Por lo cual, es difícil para el modelo poder identificar de forma correcta la clase. Por lo tanto, se puede concluir que el modelo conseguido no tiene la capacidad de poder discriminar entre una clase y la otra, de tal manera, deja de ser considerado como una opción para el tratamiento de datos desbalanceados con este conjunto de datos.

5.6 Resumen

La tabla 5.12 muestra un cuadro comparativo con todos los resultados obtenidos anteriormente. A diferencia de las tablas mostradas en las secciones anteriores, en esta ocasión no se presentan los desvíos estándar para cada métrica, solamente los promedios.

ALGORITMO ML		TÉCNICA	VALIDACIÓN			
			Accuracy	Precision	Recall	F1
Random Forest		Sin Balancear	0,8066	0,6006	0,399	0,4786
		Random Oversampling	**0.9136**	**0.8723**	**0.97**	**0.92**
		NearMiss	0.7343	0.7396	0.726	0.7316
		SMOTE	0,84	0,8456	0,837	0,8396
RNA	Backpropagation	Sin Balancear	0,82	0,6476	0,443	0,5263
		Random Oversampling	0,7213	0,81	0,581	0,677
		NearMiss	0,77	0,762	0,783	0,7723
		SMOTE	0,7936	0,8693	0,691	0,7703
		Autoencoders	0,42	0,25	0,84	0,39

Tabla 5.12. *Cuadro comparativo de los modelos implementados.*

Finalmente, se puede visualizar que el algoritmo de balanceo de mayor impacto en la clasificación fue *Random Oversampling* con el clasificador de *Random Forest.* Juntos obtuvieron los mejores valores respecto a las demás técnicas. A su vez, fue seguido por *SMOTE + Random Forest*. Uno de los pésimos resultados se obtuvo en *Random Oversampling* con el clasificador *Backpropagation*. A pesar de ello, se ve una mejora respecto a la clasificación sin balancear (Tabla 5.1). Por otra parte, la aplicación de *Autoencoders* no brindó buenos resultados por lo que es la técnica más defectuosa y menos recomendada de las expuestas para este conjunto de datos. Incluso se pueden visualizar mejores resultados en los modelos sin balancear que en el propio *Autoencoders*. Respecto a las demás técnicas, la diferencia numérica en cada uno de los resultados no es significativa como para que uno de ellos pueda ser considerado mejor que el otro. No obstante, superan los resultados de los modelos sin balancear.

Para finalizar de examinar las métricas, se puede analizar la *Tasa de verdaderos positivos (TVP)* y *falsos positivos (TFP)*. Dichas métricas permiten visualizar cuál es el modelo de predicción que ha obtenido un mejor posicionamiento respecto al punto de coordenadas (0,1). De esta forma, se puede representar su nivel de *Sensibilidad* y *1-Especificidad*. A continuación, se presenta un resumen de los resultados de *TVP* y *TFP* calculados anteriormente en cada modelo:

ALGORITMO ML	TÉCNICA	TVP	TFP
Random Forest (RF)	S/ Balance	0,9236	0,6013
	Random Oversampling	**0,8590**	**0,0296**
	NearMiss	0,7444	0,2742
	SMOTE	0,8461	0,1642
Backpropagation (BCK)	S/ Balance	0,9306	0,5567
	Random Oversampling	0,8628	0,4183
	NearMiss	0,7569	0,2170
	SMOTE	0,8962	0,3084

Tabla 5.13. *Resultados de TVP y TFP de los modelos descritos.*

Estos resultados representan los puntos en el espacio de *Sensibilidad* y *1-Especificidad* y, como era de esperar, se logra evidenciar que el modelo de *Random Forest* con balanceo *Random Oversampling*, es el punto más próximo a la coordenada (0,1). En otras palabras, se puede observar que su desempeño es mejor que el resto y está representando por el 85,90% de *Sensibilidad* y un 2,96% de probabilidad de que se produzca una falsa alarma. A continuación, se puede ver la representación gráfica correspondiente:

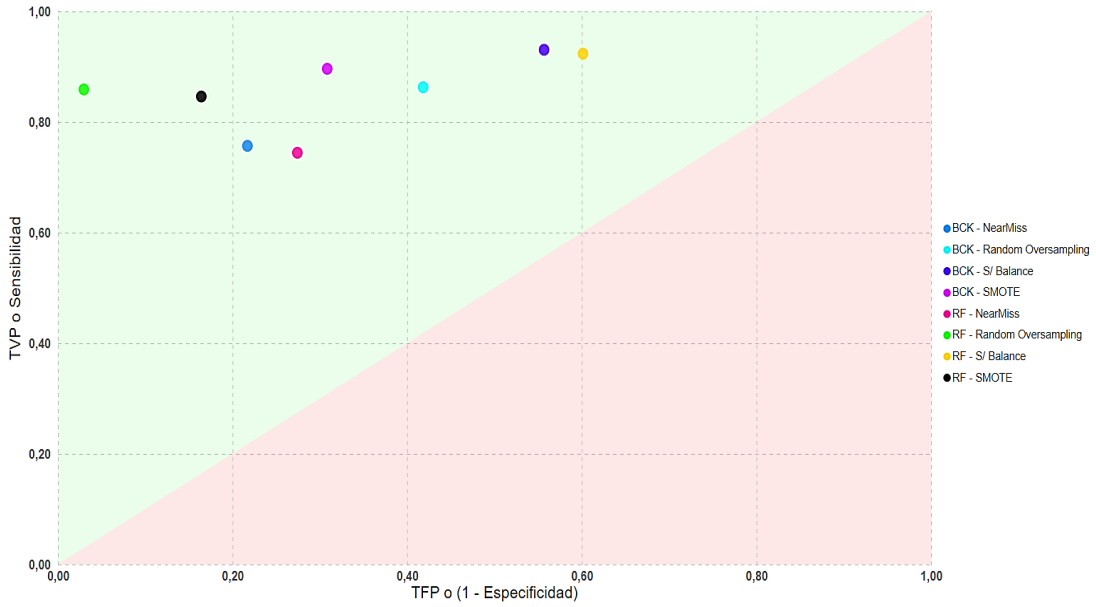


Figura 5.14. *Representación gráfica en el espacio TVP y TFP.*

Capítulo VI
Conclusiones

La investigación realizada en este trabajo se enfocó en el estudio y aplicación de diversas estrategias de balanceo de clases para mejorar el poder pronóstico de distintos clasificadores y, así mismo, demostrar lo importante que resulta disponer de un conjunto de datos equilibrado para la explotación de información. En base a ello, se buscó por medio de un trabajo experimental obtener diferentes enfoques para construir un cuadro comparativo que permitiera visualizar el rendimiento e impacto de los algoritmos de balanceo.

El proceso de experimentación se basó en 3 pilares fundamentales: conjunto de datos, algoritmos de balanceo y clasificadores. Para la obtención del conjunto de datos se utilizaron dominios reales proporcionados por una organización de cobertura médica; posteriormente se realizó un preprocesamiento y seguidamente se emplearon tres técnicas de balanceo de clases del método de Remuestreo, las cuales se basan en muestras aleatorias, sintéticas y cercanas. Cada una de ellas dio lugar a diferentes versiones del conjunto de datos original para la misma representación; luego, para cada variante del set de datos se dividió en 80% para el conjunto de entrenamiento, donde se aplicó un algoritmo de clasificación –Random Forest o Backpropagation- y el 20% restante corresponde al conjunto de prueba. Del mismo modo, se consideró la aplicación del conjunto de datos original, es decir, sin balancear, el cual fue utilizado como punto de partida para la comparación de los diversos enfoques. Adicionalmente, se diseñó un modelo basado en Autoencoders para contribuir una alternativa al problema del desbalance de clases y al método de Remuestreo.

Los resultados muestran que efectivamente dichas técnicas del método de Remuestreo mejoran el desempeño de clasificación con datos desbalanceados. Precisamente, se logró una mejoría considerable en las distintas métricas y un aumento global en el rendimiento de cada clasificador en referencia a los valores obtenidos sin balancear. Sin embargo, es necesario reconocer, que, así como se presentan casos muy alentadores, también hay algoritmos de balanceo de clases para los cuales el desempeño no es eficaz, debido a las características del conjunto de datos empleado. De todas maneras, estos resultados no deben quitarles mérito a los modelos obtenidos, puesto que en general los algoritmos de balanceo han demostrado tener un comportamiento más robusto a través de otro clasificador. Un claro ejemplo de esto es la técnica de Random Oversampling (ROS), donde se consigue el mejor y el peor resultado, Random Forest y Backpropagation respectivamente. Por otra parte, el modelo diseñado con Autoencoders no brindó buenos resultados por lo que es la técnica menos recomendada de las expuestas para utilizar con este conjunto de datos. En sí mismo, el modelo presentó grandes dificultades para discriminar entre una clase u otra, de tal manera que no se pudo demostrar (precisamente con este conjunto de datos) que los Autoencoders pueden ser una alternativa eficaz para afrontar el problema de desbalance de clases. A pesar de ello, no es factible extrapolar este resultado a todos los casos de clasificación ya que cada conjunto de datos es único.

Los modelos conseguidos pueden ser utilizados por la prestataria del servicio de coberturas médicas para un análisis más exhaustivo. Dichos modelos pueden ser empleados como fuente de conocimiento, por ejemplo, poder predecir cuándo un

beneficiario podría darse de baja. Existen muchas técnicas de pronóstico que permiten puntuar las muestras del conjunto de datos para determinar posibles beneficiarios a darse de baja. Un hecho simple es analizar los casos que son clasificados incorrectamente por los modelos; si un modelo predice un “Activo” como “Baja”, es un indicador que esa muestra "Activo" tiene similitudes con las muestras “Baja”, por lo cual tiene sentido examinar en más detalles esos casos para mitigar futuras pérdidas de beneficiarios.

Para concluir, es primordial tratar la problemática de desbalance de clases antes de entrenar un modelo de clasificación para no obtener resultados sesgados. Esto se puede observar en la experimentación, cuyos resultados, a excepción de Autoencoders, al tratar el desbalance de clases se lograron obtener modelos más confiables respecto a los conseguidos con el conjunto de datos desbalanceado.

En consecuencia, al finalizar este trabajo, se consigue un conocimiento sobre la clasificación con datos desbalanceados, su problemática y la necesidad de aplicar técnicas que mejoren sus resultados. Además, las razones de por qué un clasificador no debe ser medido por su nivel de Accuracy y por qué la misma puede ser engañosa.

Por otra parte, a trabajo futuro, existen diversos algoritmos que en esta investigación no se han estudiado pero que pueden ser de alto interés para la cuestión del desbalance de clases, por ejemplo, el modelo logístico, la profundización de Remuestreo híbridos o el intervalo de confianza de un modelo. Algunos de ellos constaban de desarrollos metodológicos más complejos y con un tiempo de procesado superior a los que se han visto, y por este tipo de razones, no han sido considerados en la elaboración de este trabajo final integrador.

Capítulo VII
Bibliografía

A. López Pineda. (2018). Algoritmos de balanceo de clases en problemas de clasificación binaria de conjuntos altamente desproporcionados. Tesis Maestría en Sistemas Inteligentes.

A. Palmer, J. Montaño & R. Jiménez. (2001). Tutorial sobre Redes Neuronales Artificiales: El Perceptrón Multicapa. Revista Electrónica de Psicología. Vol. 5, No. 2.

Autoencoder as a Classifier using Fashion-MNIST Dataset Tutorial (último acceso 31/03/2022).

B. Frenay, M. Verleysen. (2014). Classification in the presence of label noise: a survey, IEEE transactions on neural networks and learning systems, 25(5), p. 845-869.

C. Albon. (2018). Machine Learning with Python Cookbook. O'Reilly Media, Inc. ISBN: 978-1-491-98938-8.

C. Goutte & E. Gaussier. (2005). A Probabilistic Interpretation of Precision, Recall and F-score, with Implication for Evaluation.

C. Mera & J. M. Arrieta Ramos. (2015). Estudio Comparativo de Técnicas de Balanceo de Datos en el Aprendizaje de Múltiples Instancias. LACNE 2015 - Latin American Conference on Networking and Electronic Media At: Medellín. Colombia.

E. Garibay-Ruiz. (2018). Autoencoders para manejo de clases desbalanceadas en problemas de propensión de compra.

F. J. Pulgar, A. J. Rivera, F. Charte & J. De Jesús. (2018). Análisis del impacto de datos desbalanceados en el rendimiento predictivo de redes neuronales convolucionales. Instituto Andaluz Interuniversitario en Data Science and Computational Intelligence (DaSCI).

G. Hongyu & V. Herna. (2004). Learning from imbalanced data sets with boosting and data generation: The DataBoost-IM approach. SIGKDD Explorations. Newsl. 6, 1 pp. 30–39.

G. Lemaître, F. Nogueira & C. Aridas. (2017). Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. Journal of Machine.

I. Tomek. (1976). Two modifications of cnn. IEEE Trans. Syst. Man Cybern., 6:769-772.

Implementing Autoencoders in Keras: Tutorial (último acceso 15/04/2022).

J. Brownlee. (2020). SMOTE Oversampling for Imbalanced Classification with Python (último acceso 21/03/2022).

J. De Jesús; A. Guadalupe-Ramírez; J. Ambriz-Polo & E. López-González. (2018). Algoritmo de aprendizaje eficiente para tratar el problema del desbalance de múltiples clases. Research in Computing Science 147(5):143-157.

J. Han, M. Kamber & J. Pei. (2012). Data mining - Concepts and Techniques, 3rd Edition. Morgan Kaufmann. ISBN: 978-0-12-381479-1.

J. Hernández Orallo, M. J. Ramírez Quintana & C. Ferri Ramírez. (2004). Introducción a la Minería de Datos. Pearson Prentice Hall, ISBN: 84-205-4091-9.

J. Moreno, D. Rodríguez, M. Sicilia, J. Riquelme & Y. Ruiz. (2009). SMOTE-I: mejora del algoritmo SMOTE para balanceo de clases minoritarias. Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos, Vol. 3, No. 1.

J. Ross Quinlan. (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, Inc. ISBN: 1-55860-238-0.

J. Van Hulse, T. M. Khoshgoftaar & A. Napolitano. (2007). "Experimental perspectives on learning from imbalanced data," in ICML, pp. 935–942.

L. A. Caballero-Cruz, A. López-Chau & J. Bautista-López. (2015). Árbol de decisión C4.5 basado en entropía minoritaria para clasificación de conjuntos de datos no balanceados. Research in Computing Science 92(1):23-34.

L. Bojarczuk, H. Lopes; A. Freitas & E. Michalkiewicz. (2004). A constrained-syntax genetic programming system for discovering classification rules: Application to medical data sets. Artificial intelligence in medicine 30(1): 27-48.

L. Breiman. (2001). Random forests. Machine Learning (vol. 45, pp. 5-32).

L. Larcher & R. Costaguta. (2004). Una red neuronal backpropagation aplicada a la Micro Histolog. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial 8(23):105-109.

M. Buscema. (1998). Backpropagation Neural Networks. Substance use & misuse, 33(2): 233-270.

M. Galar, A. Fernández, E. Barrenechea, H. Bustince & F. Herrera. (2012). A review on ensembles for class imbalance problem: bagging, boosting and hybrid-based approaches, IEEE Transactions on Systems, Man, and Cybernetics, 42 (4), p. 463-484.

N. Japkowicz. (2000). The class imbalance problem: Significance and strategies. In Proceedings of the 2000 International Conference on Artificial Intelligence (IC-AI'2000), volume 1, pages 111-117.

N. V. Chawla, K. W. Bowyer, L.O. Hall & W.P. Kegelmeyer. (2002). SMOTE: Synthetic Minority Over-Sampling Technique. Journal of artificial intelligence research, p. 321-357.

N. Thai-Nghe, Z. Gantner & L. Schmidt-Thieme. (2010). "Cost-sensitive learning methods for imbalanced data," in IJCNN, pp. 1–8.

N. Yuan, L. Zhang, J.-T. Shi, X. Xia & G. Li. (2019). "Theories and applications of autoencoder neural networks: A literature survey," Jisuanji Xuebao/Chinese Journal of Computers, vol. 42, pp. 203–230, 01.

P. Chan, W. Fan, A. Prodromidis & S. Stolfo. (1999). Distributed Data Mining in Credit Card Fraud Detection. IEEE Intelligent Systems and their Applications, vol. 14, no. 6, pp. 67-74.

P. Domingos. (1999). "MetaCost: a general method for making classifiers cost-sensitive," in SIGKDD, pp. 155–164.

Random Forest con Python (último acceso 31/03/2022).

S. Cateni, V. Colla & M. Vannucci. (2014). A method for resampling imbalanced datasets in binary classification tasks for real-world problems.

S. R. Timarán-Pereira, I. Hernández-Arteaga, S. J. Caicedo-Zambrano, A. Hidalgo-Troya & J. C. Alvarado-Pérez. (2016). El proceso de descubrimiento de conocimiento en bases de datos.

S. Wang. (2011). "Ensemble diversity for class imbalance learning," Ph.D. Thesis, The University of Birmingham.

T. G. Smith. (2018). Handle class imbalance intelligently by using variational autoencoders to generate synthetic observations of your minority class.

T. Hoens & N. Chawla. (2013). Imbalanced Datasets: From Sampling to Classifiers (último acceso 15/04/2022).

T. Jo & N. Japkowicz. (2004). Class imbalances versus small disjuncts. SIGKDD Explor. Newsi, 6(l):40-49.

T. Wang, G. Zeng, D.S. Yeung & J. Li. (2017). Dual Denoising Autoencoder Features for Imbalance Classification Problems. 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 312-317.

U. Fayyad, G. Piatetsky-Shapiro, & P. Smyth. (1996). From Data Mining to Knowledge Discovery in Databases.

W.W. Ng, W. Pedrycz, D.S. Yeung, G. Zeng & J. Zhang. (2016). Dual autoencoders features for imbalance classification problem. Pattern Recognition, 60, 875-889.

X.-Y. Liu, J. Wu & Z.-H Zhou. (2009). Exploratory undersampling for class-imbalance learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 39(2):539-550.

Y. Sun, A. Wong & M. S. Kamel. (2009). Classification of imbalanced data: a review. International Journal of Pattern Recognition and Artificial Intelligence. Vol. 23, No. 04, pp. 687-719.

Capítulo VIII
Apéndice

Antes de comenzar con las líneas de código se debe asegurar si se cuenta con los siguientes paquetes:

- **Scikit-learn:**
  a) Terminal Conda: conda install scikit-learn
  b) Terminal Windows: pip install scikit-learn
  c) Más info en https://scikit-learn.org/stable/install.html

- **Imbalanced-learn:**
  a) Terminal Conda: conda install -c conda-forge imbalanced-learn
  b) Terminal Windows: pip install -U imbalanced-learn
  c) Más info en https://imbalanced-learn.readthedocs.io/en/stable/install.html

- **Pandas:**
  a) Terminal Conda: conda install pandas
  b) Terminal Windows: pip install pandas
  c) Más info en https://pandas.pydata.org/docs/getting_started/index.html

- **Numpy:**
  a) Terminal Conda: conda install numpy
  b) Terminal Windows: pip install numpy
  c) Más info en https://numpy.org/install/

- **Matplotlib:**
  a) Terminal Conda: conda install matplotlib
  b) Terminal Windows: python -m pip install -U matplotlib
  c) Más info en https://scikit-learn.org/stable/install.html

- **Keras:**
  a) Terminal Conda: conda install keras
  b) Terminal Windows: pip install keras
  c) Más info en https://keras.io/api/

PREPROCESAMIENTO

Carga de archivo

```
>>> import pandas as pd
>>> df = pd.read_csv('./Datos/Afiliados.csv')
```

Información dataset

```
>>> df.info()
```

Datos faltantes

```
>>> print(df.isnull().values.any())
>>> df.isnull().sum()
```

Matriz de correlación

```
>>> #import matplotlib.pyplot as plt
>>> #import numpy as np
>>> import seaborn as sns
>>> #import os

>>> df2 = df[['edad','antig_mes','reclamo_tot','cant_capita','capita_baja']]
>>> #--Matriz de correlacion --
>>> print(df2.corr())
>>>
>>> sns.heatmap(df2.corr(), square=True, annot=True)
```

Eliminación de id_beneficiario

```
>>> df = df.drop(['id_beneficiario'], axis=1)
```

Numerizacion

```
>>> mapeo = {
>>>         "adhesion": {"I":0, "E":1},
>>>         "sexo": {"F":0, "M":1},
>>>         "discapacidad": {"NO":0, "SI":1},
>>>         "deuda": {"NO":0, "SI":1},
>>>         "aut_rech": {"NO":0, "SI":1},
>>>         "reint_rech": {"NO":0, "SI":1},
>>>         "excep_rech": {"NO":0, "SI":1},
>>>         "aut_emi": {"NO":0, "SI":1},
>>>         "reint_emi": {"NO":0, "SI":1},
>>>         "excep_emi": {"NO":0, "SI":1},
>>>         "aumento_cuota": {"NO":0, "SI":1},
>>>         "aumento_menor_5": {"NO":0, "SI":1},
>>>         "aumento_entre_5_10": {"NO":0, "SI":1},
>>>         "aumento_mayor_10": {"NO":0, "SI":1},
>>>         "capita_baja": {"NO":0, "SI":1}
>>>        }
>>>
>>> f.replace(mapeo, inplace=True)
```

Columnas dummies

```
>>> col = 'region'
>>> df = pd.concat([pd.get_dummies(df[col], prefix=col) if df[col].dtype == object else df[col] for col in df],
axis=1)
```

Exportación a archivo csv

```
>>> df.to_csv('./Datos/Data_Preprocemiento.csv', index=False, encoding='utf-8')
```

RANDOM FOREST

Random Forest - Sin balance

Lectura de datos y recuento

```
>>> # Importar paquete
>>> import pandas as pd

>>> # Carga del DataFrame
>>> df = pd.read_csv('./Datos/Data_Preprocemiento.csv')

>>> # Recuento
>>> data = df.drop(['capita_baja'], axis=1)
>>> target = df.loc[:,'capita_baja']
>>> baja = target.sum()
>>> activo = target.shape[0]- baja
>>> print('Bajas: ', baja, ' , Activos: ', activo )
```

Entrenamiento Random Forest

```
>>> #Import Random Forest Model
>>> from sklearn.ensemble import RandomForestClassifier

>>> #1. División de datos en conjunto de evaluación y conjunto de entrenamiento
>>> from sklearn.model_selection import train_test_split
>>> porc_test= 0.2 # 80% training y 20% test
>>> dataTrain, dataTest, targetTrain, targetTest = train_test_split(data, target, test_size= porc_test)
>>> print("\t %d datos para training" %dataTrain.shape[0])
>>> print("\t %d datos para testing" %dataTest.shape[0])>>>

>>> #2. Inicialización del modelo #Create a Gaussian Classifier
>>> criterion ='gini'
>>> max_depth = None
>>> max_features= 'auto'
>>> n_estimators= 100
>>> clf = RandomForestClassifier (n_estimators = n_estimators, criterion = criterion,max_depth = max_depth, max_features = max_features)

>>> #3. Entrenamiento del modelo
>>> clf.fit(dataTrain,targetTrain)

>>> #4. Predicción con el modelo
>>> targetPred = clf.predict(dataTest)
```

Evaluación del modelo

```
>>> # Importación de paquetes
>>> from sklearn.metrics import accuracy_score,recall_score, precision_score, f1_score
>>> import matplotlib.pyplot as plt
>>> from sklearn.metrics import confusion_matrix
>>> import numpy as np
>>> import itertools>>>

>>> # Accuracy, Precision, Recall, F1-Measure
>>> print("\nTest:")
>>> print("\t Accuracy: %.2f" % accuracy_score(targetTest, targetPred ))
>>> print("\t Precision: %.2f" % precision_score(targetTest, targetPred) )
>>> print("\t Recall: %.2f" % recall_score(targetTest, targetPred ))
>>> print("\t F1-Measure: %.2f" % f1_score(targetTest, targetPred))>>>

>>> # Matriz de confusión
>>> def plot_confusion_matrix(real_target, pred_target, classes, normalize=False, title='Matriz de Confusión', cmap=plt.cm.Reds):
>>>
>>>     cm= confusion_matrix(real_target, pred_target)
>>>
>>>     if normalize:
>>>         cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
>>>
>>>     plt.figure()
>>>     plt.imshow(cm, interpolation='nearest', cmap=cmap)
>>>     plt.title(title)
>>>     plt.colorbar()
>>>     tick_marks = np.arange(len(classes))
>>>     plt.xticks(tick_marks, classes, rotation=0)
>>>     plt.yticks(tick_marks, classes, rotation=0)

>>>     fmt = '.2f' if normalize else 'd'
>>>     thresh = cm.max() / 2.
>>>     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
>>>         plt.text(j, i, format(cm[i, j], fmt),
>>>                  horizontalalignment="center",
>>>                  color="white"
>>>                  if cm[i, j] > thresh else "black")

>>>     plt.tight_layout()
>>>     plt.ylabel('True')
```

```
>>>     plt.xlabel('Predicted')
>>>
>>> # Llamada a la función
>>> plot_confusion_matrix(targetTest, targetPred, ['0', '1'], False, 'Matriz de Confusión', plt.cm.Reds)
```

Resultados de 30 ejecuciones

Exc.	Random Forest - Sin balance									
	Acc.	Pre.	Rec.	F1	VP	FN	FP	VN	TVP	TFP
1	0,81	0,62	0,4	0,48	11590	886	2202	1449	0,9289	0,6031
2	0,81	0,6	0,42	0,49	11551	1005	2086	1485	0,9199	0,5841
3	0,8	0,6	0,42	0,49	11447	1028	2128	1524	0,9175	0,5826
4	0,81	0,61	0,4	0,49	11608	906	2166	1447	0,9276	0,5995
5	0,81	0,61	0,39	0,48	11563	919	2220	1425	0,9263	0,6090
6	0,8	0,59	0,38	0,46	11564	950	2242	1371	0,9240	0,6205
7	0,81	0,6	0,4	0,48	11598	952	2148	1429	0,9241	0,6005
8	0,81	0,62	0,41	0,49	11536	930	2165	1496	0,9253	0,5913
9	0,81	0,6	0,4	0,48	11609	955	2140	1423	0,9239	0,6006
10	0,81	0,61	0,4	0,48	11540	946	2178	1463	0,9242	0,5981
11	0,8	0,58	0,4	0,47	11493	1024	2180	1430	0,9181	0,6038
12	0,81	0,61	0,39	0,48	11583	914	2212	1418	0,9268	0,6093
13	0,8	0,59	0,41	0,48	11475	1025	2140	1487	0,918	0,5900
14	0,81	0,6	0,41	0,48	11602	963	2119	1443	0,9233	0,5948
15	0,8	0,58	0,39	0,47	11539	998	2184	1406	0,9203	0,6083
16	0,81	0,61	0,39	0,48	11545	937	2207	1438	0,9249	0,6054
17	0,81	0,58	0,39	0,47	11599	992	2149	1387	0,9212	0,6077
18	0,81	0,61	0,4	0,48	11542	934	2200	1451	0,9251	0,6025
19	0,8	0,58	0,39	0,47	11548	992	2191	1396	0,9208	0,6108
20	0,81	0,58	0,39	0,47	11640	988	2137	1362	0,9217	0,6107
21	0,8	0,6	0,39	0,47	11519	947	2248	1413	0,9240	0,6140
22	0,8	0,6	0,4	0,48	11552	965	2183	1427	0,9229	0,6047
23	0,81	0,61	0,41	0,49	11574	957	2113	1483	0,9236	0,5875
24	0,81	0,61	0,41	0,49	11563	940	2134	1490	0,9248	0,5888
25	0,81	0,61	0,4	0,48	11597	924	2156	1450	0,9262	0,5978
26	0,81	0,61	0,4	0,48	11581	932	2177	1437	0,9255	0,6023
27	0,8	0,59	0,39	0,47	11556	968	2181	1422	0,9227	0,6053
28	0,81	0,61	0,4	0,48	11543	922	2210	1452	0,9260	0,6034
29	0,8	0,59	0,4	0,47	11573	992	2154	1408	0,92105	0,6047
30	0,81	0,61	0,4	0,48	11919	925	2142	1441	0,9279	0,5978
Prom.	**0,8066**	**0,6006**	**0,399**	**0,4786**	**11571,63**	**957,2**	**2169,73**	**1438,43**	**0,9235**	**0,6013**
Desv.	**0,0048**	**0,0123**	**0,0094**	**0,0078**	**77,2405**	**37,2034**	**38,3360**	**37,1202**	**0,0030**	**0,0089**

Random Forest - RandomOverSampler

Lectura de datos y recuento

```
>>> # Importar paquete
>>> import pandas as pd
```

```
>>> # Carga del DataFrame
>>> df = pd.read_csv('./Datos/Data_Preprocemiento.csv')

>>> # Recuento
>>> data = df.drop(['capita_baja'], axis=1)
>>> target = df.loc[:,'capita_baja']
>>> baja = target.sum()
>>> activo = target.shape[0]- baja
>>> print('Bajas: ', baja, ' , Activos: ', activo )
```

Balanceo de datos – RandomOverSampler

```
>>> #Importamos los paquetes
>>> from imblearn.over_sampling import RandomOverSampler

>>> #Inicialización de los métodos de sobremuestreo
>>> ros = RandomOverSampler()

>>> #ROS. Duplica muestras de la clase menos representadas
>>> dataRos, targetRos = ros.fit_resample(data,target)
>>> baja = targetRos.sum()
>>> activo = targetRos.shape[0]- baja
>>> print('Bajas: ', baja, ' , Activos: ', activo )
```

Entrenamiento Random Forest

```
>>> #Import Random Forest Model
>>> from sklearn.ensemble import RandomForestClassifier

>>> #1. División de datos en conjunto de evaluación y conjunto de entrenamiento
>>> from sklearn.model_selection import train_test_split
>>> porc_test= 0.2 # 80% training y 20% test
>>> dataTrain, dataTest, targetTrain, targetTest = train_test_split(data, target, test_size= porc_test)
>>> print("\t %d datos para training" %dataTrain.shape[0])
>>> print("\t %d datos para testing" %dataTest.shape[0])>>>

>>> #2. Inicialización del modelo #Create a Gaussian Classifier
>>> criterion ='gini'
>>> max_depth = None
>>> max_features= 'auto'
>>> n_estimators= 100
>>> clf = RandomForestClassifier (n_estimators = n_estimators, criterion = criterion,max_depth =
max_depth, max_features = max_features)

>>> #3. Entrenamiento del modelo
>>> clf.fit(dataTrain,targetTrain)

>>> #4. Predicción con el modelo
>>> targetPred = clf.predict(dataTest)
```

Evaluación del modelo

```
>>> # Importación de paquetes
>>> from sklearn.metrics import accuracy_score,recall_score, precision_score, f1_score
>>> import matplotlib.pyplot as plt
>>> from sklearn.metrics import confusion_matrix
>>> import numpy as np
>>> import itertools>>>

>>> # Accuracy, Precision, Recall, F1-Measure
>>> print("\nTest:")
>>> print("\t Accuracy: %.2f" % accuracy_score(targetTest, targetPred ))
>>> print("\t Precision: %.2f" % precision_score(targetTest, targetPred) )
>>> print("\t Recall: %.2f" % recall_score(targetTest, targetPred ))
>>> print("\t F1-Measure: %.2f" % f1_score(targetTest, targetPred))>>>

>>> # Matriz de confusión
```

```
>>> def plot_confusion_matrix(real_target, pred_target, classes, normalize=False, title='Matriz de Confusión', cmap=plt.cm.Reds):
>>>
>>>     cm= confusion_matrix(real_target, pred_target)
>>>
>>>     if normalize:
>>>         cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
>>>
>>>     plt.figure()
>>>     plt.imshow(cm, interpolation='nearest', cmap=cmap)
>>>     plt.title(title)
>>>     plt.colorbar()
>>>     tick_marks = np.arange(len(classes))
>>>     plt.xticks(tick_marks, classes, rotation=0)
>>>     plt.yticks(tick_marks, classes, rotation=0)

>>>     fmt = '.2f' if normalize else 'd'
>>>     thresh = cm.max() / 2.
>>>     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
>>>         plt.text(j, i, format(cm[i, j], fmt),
>>>                  horizontalalignment="center",
>>>                  color="white"
>>>                  if cm[i, j] > thresh else "black")

>>>     plt.tight_layout()
>>>     plt.ylabel('True')
>>>     plt.xlabel('Predicted')
>>>
>>> # Llamada a la función
>>> plot_confusion_matrix(targetTest, targetPred, ['0', '1'], False, 'Matriz de Confusión', plt.cm.Reds)
```

Resultados de 30 ejecuciones

Exc.	Random Forest - RandomOverSampler									
	Acc.	Pre.	Rec.	F1	VP	FN	FP	VN	TVP	TFP
1	0,91	0,87	0,97	0,92	10817	1726	437	12068	0,8623	0,0349
2	0,92	0,88	0,97	0,92	10734	1722	350	12242	0,8617	0,0277
3	0,92	0,88	0,97	0,92	10854	1717	369	12108	0,8634	0,0295
4	0,91	0,87	0,97	0,92	10617	1761	370	12300	0,8577	0,0292
5	0,91	0,87	0,97	0,92	10706	1837	382	12123	0,8535	0,0305
6	0,91	0,87	0,97	0,92	10692	1777	384	12195	0,8574	0,0305
7	0,92	0,87	0,97	0,92	10780	1760	329	12179	0,8596	0,0263
8	0,92	0,88	0,97	0,92	10853	1634	396	12165	0,8691	0,0315
9	0,91	0,87	0,97	0,92	10704	1825	381	12138	0,8543	0,0304
10	0,91	0,87	0,97	0,92	10772	1799	398	12079	0,8568	0,0318
11	0,91	0,87	0,97	0,92	10733	1777	402	12136	0,8579	0,0320
12	0,92	0,87	0,97	0,92	10737	1785	320	12206	0,8574	0,0255
13	0,91	0,87	0,97	0,92	10823	1785	347	12093	0,8584	0,0278
14	0,91	0,87	0,97	0,92	10617	1807	400	12224	0,8545	0,0316
15	0,91	0,87	0,97	0,92	10814	1795	339	12100	0,8576	0,0272
16	0,91	0,87	0,97	0,92	10713	1747	406	12182	0,8597	0,0322
17	0,91	0,87	0,97	0,92	10761	1800	365	12122	0,8566	0,0292
18	0,91	0,87	0,97	0,92	10694	1764	367	12223	0,8584	0,0291
19	0,91	0,87	0,97	0,92	10695	1760	381	12212	0,8586	0,0302
20	0,91	0,87	0,97	0,92	10722	1771	379	12176	0,8582	0,0301

21	0,91	0,87	0,97	0,92	10737	1801	369	12141	0,8563	0,0294
22	0,91	0,87	0,97	0,92	10824	1788	373	12063	0,8582	0,0299
23	0,91	0,87	0,97	0,92	10676	1797	362	12213	0,8559	0,0287
24	0,92	0,88	0,97	0,92	10788	1734	381	12145	0,8615	0,0304
25	0,91	0,87	0,97	0,92	10755	1842	347	12104	0,8537	0,0278
26	0,92	0,88	0,97	0,92	10720	1704	386	12238	0,8628	0,0305
27	0,92	0,88	0,97	0,92	10790	1731	373	12154	0,8617	0,0297
28	0,92	0,87	0,97	0,92	10781	1753	374	12140	0,8601	0,0298
29	0,92	0,87	0,97	0,92	10840	1733	373	12102	0,8621	0,0298
30	0,92	0,88	0,97	0,92	10793	1740	320	12195	0,8611	0,0255
Prom.	**0,9136**	**0,8723**	**0,97**	**0,92**	**10751,4**	**1765,73**	**372**	**12158,87**	**0,8589**	**0,0296**
Desv.	**0,0049**	**0,0043**	**0**	**0**	**62,8647**	**43,3796**	**25,8190**	**58,2093**	**0,0033**	**0,0020**

Random Forest - NearMiss

Lectura de datos y recuento

```
>>> # Importar paquete
>>> import pandas as pd

>>> # Carga del DataFrame
>>> df = pd.read_csv('./Datos/Data_Preprocemiento.csv')

>>> # Recuento
>>> data = df.drop(['capita_baja'], axis=1)
>>> target = df.loc[:,'capita_baja']
>>> baja = target.sum()
>>> activo = target.shape[0]- baja
>>> print('Bajas: ', baja, ' , Activos: ', activo )
```

Balanceo de datos – NearMiss

```
>>> #Importamos los paquetes
>>> from imblearn.under_sampling import NearMiss

>>> #Inicialización de los métodos de submuestreo NearMiss
>>> nm = NearMiss()

>>> #NearMiss. Elimina las muestras más cercanas de la clase más representada
>>> dataNm, targetNm = nm.fit_resample(data,target)
>>> baja = targetNm.sum()
>>> activo = targetNm.shape[0]- baja
>>> print('Bajas: ', baja, ' , Activos: ', activo )
```

Entrenamiento Random Forest

```
>>> #Import Random Forest Model
>>> from sklearn.ensemble import RandomForestClassifier

>>> #1. División de datos en conjunto de evaluación y conjunto de entrenamiento
>>> from sklearn.model_selection import train_test_split
>>> porc_test= 0.2 # 80% training y 20% test
>>> dataTrain, dataTest, targetTrain, targetTest = train_test_split(data, target, test_size= porc_test)
>>> print("\t %d datos para training" %dataTrain.shape[0])
>>> print("\t %d datos para testing" %dataTest.shape[0])>>>

>>> #2. Inicialización del modelo #Create a Gaussian Classifier
>>> criterion ='gini'
>>> max_depth = None
>>> max_features= 'auto'
>>> n_estimators= 100
```

```
>>> clf = RandomForestClassifier (n_estimators = n_estimators, criterion = criterion,max_depth = max_depth, max_features = max_features)

>>> #3. Entrenamiento del modelo
>>> clf.fit(dataTrain,targetTrain)

>>> #4. Predicción con el modelo
>>> targetPred = clf.predict(dataTest)
```

Evaluación del modelo

```
>>> # Importación de paquetes
>>> from sklearn.metrics import accuracy_score,recall_score, precision_score, f1_score
>>> import matplotlib.pyplot as plt
>>> from sklearn.metrics import confusion_matrix
>>> import numpy as np
>>> import itertools>>>

>>> # Accuracy, Precision, Recall, F1-Measure
>>> print("\nTest:")
>>> print("\t Accuracy: %.2f" % accuracy_score(targetTest, targetPred ))
>>> print("\t Precision: %.2f" % precision_score(targetTest, targetPred) )
>>> print("\t Recall: %.2f" % recall_score(targetTest, targetPred ))
>>> print("\t F1-Measure: %.2f" % f1_score(targetTest, targetPred))>>>

>>> # Matriz de confusión
>>> def plot_confusion_matrix(real_target, pred_target, classes, normalize=False, title='Matriz de Confusión', cmap=plt.cm.Reds):
>>>
>>>     cm= confusion_matrix(real_target, pred_target)
>>>
>>>     if normalize:
>>>         cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
>>>
>>>     plt.figure()
>>>     plt.imshow(cm, interpolation='nearest', cmap=cmap)
>>>     plt.title(title)
>>>     plt.colorbar()
>>>     tick_marks = np.arange(len(classes))
>>>     plt.xticks(tick_marks, classes, rotation=0)
>>>     plt.yticks(tick_marks, classes, rotation=0)

>>>     fmt = '.2f' if normalize else 'd'
>>>     thresh = cm.max() / 2.
>>>     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
>>>         plt.text(j, i, format(cm[i, j], fmt),
>>>                  horizontalalignment="center",
>>>                  color="white"
>>>                  if cm[i, j] > thresh else "black")

>>>     plt.tight_layout()
>>>     plt.ylabel('True')
>>>     plt.xlabel('Predicted')
>>>
>>> # Llamada a la función
>>> plot_confusion_matrix(targetTest, targetPred, ['0', '1'], False, 'Matriz de Confusión', plt.cm.Reds)
```

Resultados de 30 ejecuciones

Exc.	Random Forest - NearMiss									
	Acc.	Pre.	Rec.	F1	VP	FN	FP	VN	TVP	TFP
1	0,73	0,73	0,73	0,73	2683	958	956	2610	0,7368	0,2680
2	0,74	0,74	0,73	0,73	2720	927	960	2600	0,7458	0,2696
3	0,73	0,73	0,73	0,73	2609	957	993	2648	0,7316	0,2727
4	0,73	0,74	0,73	0,74	2621	916	998	2672	0,7410	0,2719

5	0,73	0,74	0,72	0,73	2692	930	994	2591	0,7432	0,2772
6	0,74	0,75	0,74	0,74	2691	900	954	2662	0,7493	0,2638
7	0,75	0,75	0,74	0,74	2729	878	945	2655	0,7565	0,2625
8	0,74	0,74	0,73	0,73	2689	923	979	2616	0,7444	0,2723
9	0,73	0,74	0,73	0,73	2645	934	983	2645	0,7390	0,2709
10	0,72	0,73	0,71	0,72	2649	969	1029	2560	0,7321	0,2867
11	0,72	0,73	0,72	0,72	2647	969	1017	2574	0,7320	0,2832
12	0,73	0,74	0,73	0,73	2643	939	985	2640	0,7378	0,2717
13	0,73	0,74	0,72	0,73	2685	909	1005	2608	0,7470	0,2781
14	0,74	0,75	0,72	0,74	2722	864	1001	2620	0,7590	0,2764
15	0,74	0,74	0,73	0,73	2720	906	975	2606	0,7501	0,2722
16	0,73	0,72	0,74	0,73	2672	1004	927	2604	0,7268	0,2625
17	0,74	0,75	0,72	0,74	2672	890	1005	2640	0,7501	0,2757
18	0,74	0,75	0,73	0,74	2674	901	965	2667	0,7479	0,2656
19	0,74	0,75	0,73	0,74	2672	900	969	2666	0,7480	0,2665
20	0,74	0,75	0,73	0,74	2737	882	971	2617	0,7562	0,2706
21	0,74	0,74	0,72	0,73	2746	901	984	2576	0,7529	0,2764
22	0,74	0,75	0,73	0,74	2723	882	988	2614	0,7553	0,2742
23	0,73	0,73	0,72	0,72	2700	946	1005	2556	0,7405	0,2822
24	0,73	0,73	0,72	0,73	2644	953	1003	2607	0,7350	0,2778
25	0,73	0,74	0,71	0,72	2733	880	1055	2539	0,7564	0,2935
26	0,74	0,74	0,73	0,73	2695	934	972	2606	0,7426	0,2716
27	0,73	0,73	0,72	0,73	2634	961	1000	2612	0,7326	0,2768
28	0,73	0,74	0,73	0,73	2693	931	985	2598	0,7431	0,2749
29	0,73	0,74	0,72	0,73	2677	906	1031	2593	0,7471	0,2844
30	0,74	0,74	0,72	0,73	2708	895	994	2610	0,7515	0,2758
Prom.	**0,7343**	**0,7396**	**0,726**	**0,7316**	**2684,16**	**921,5**	**987,6**	**2613,73**	**0,7444**	**0,2742**
Desv.	**0,0068**	**0,0081**	**0,0077**	**0,0065**	**36,1416**	**33,2418**	**26,7396**	**33,7832**	**0,0087**	**0,0071**

Random Forest - SMOTE

Lectura de datos y recuento

```
>>> # Importar paquete
>>> import pandas as pd

>>> # Carga del DataFrame
>>> df = pd.read_csv('./Datos/Data_Preprocemiento.csv')

>>> # Recuento
>>> data = df.drop(['capita_baja'], axis=1)
>>> target = df.loc[:,'capita_baja']
>>> baja = target.sum()
>>> activo = target.shape[0]- baja
>>> print('Bajas: ', baja, ' , Activos: ', activo )
```

Balanceo de datos – SMOTE

```
>>> # Importación de librería
>>> from imblearn.over_sampling import SMOTE

>>> # Inicialización del método de sobremuestreo SMOTE
```

```
>>> smote = SMOTE()

>>> # SMOTE. Genera nuevas muestras sintéticas
>>> dataSmote, targetSmote = smote.fit_resample(data,target)
>>> bajaSmote = targetSmote.sum()
>>> activoSmote = targetSmote.shape[0]- bajaSmote
>>> print('Bajas: ', bajaSmote, ' , Activos: ', activoSmote )
```

Entrenamiento Random Forest

```
>>> #Import Random Forest Model
>>> from sklearn.ensemble import RandomForestClassifier

>>> #1. División de datos en conjunto de evaluación y conjunto de entrenamiento
>>> from sklearn.model_selection import train_test_split
>>> porc_test= 0.2 # 80% training y 20% test
>>> dataTrain, dataTest, targetTrain, targetTest = train_test_split(data, target, test_size= porc_test)
>>> print("\t %d datos para training" %dataTrain.shape[0])
>>> print("\t %d datos para testing" %dataTest.shape[0])>>>

>>> #2. Inicialización del modelo #Create a Gaussian Classifier
>>> criterion ='gini'
>>> max_depth = None
>>> max_features= 'auto'
>>> n_estimators= 100
>>> clf = RandomForestClassifier (n_estimators = n_estimators, criterion = criterion,max_depth =
max_depth, max_features = max_features)

>>> #3. Entrenamiento del modelo
>>> clf.fit(dataTrain,targetTrain)

>>> #4. Predicción con el modelo
>>> targetPred = clf.predict(dataTest)
```

Evaluación del modelo

```
>>> # Importación de paquetes
>>> from sklearn.metrics import accuracy_score,recall_score, precision_score, f1_score
>>> import matplotlib.pyplot as plt
>>> from sklearn.metrics import confusion_matrix
>>> import numpy as np
>>> import itertools>>>

>>> # Accuracy, Precision, Recall, F1-Measure
>>> print("\nTest:")
>>> print("\t Accuracy: %.2f" % accuracy_score(targetTest, targetPred ))
>>> print("\t Precision: %.2f" % precision_score(targetTest, targetPred) )
>>> print("\t Recall: %.2f" % recall_score(targetTest, targetPred ))
>>> print("\t F1-Measure: %.2f" % f1_score(targetTest, targetPred))>>>

>>> # Matriz de confusión
>>> def plot_confusion_matrix(real_target, pred_target, classes, normalize=False, title='Matriz de
Confusión', cmap=plt.cm.Reds):
>>>
>>>     cm= confusion_matrix(real_target, pred_target)
>>>
>>>     if normalize:
>>>         cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
>>>
>>>     plt.figure()
>>>     plt.imshow(cm, interpolation='nearest', cmap=cmap)
>>>     plt.title(title)
>>>     plt.colorbar()
>>>     tick_marks = np.arange(len(classes))
>>>     plt.xticks(tick_marks, classes, rotation=0)
>>>     plt.yticks(tick_marks, classes, rotation=0)

>>>     fmt = '.2f' if normalize else 'd'
>>>     thresh = cm.max() / 2.
```

```
>>>     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
>>>         plt.text(j, i, format(cm[i, j], fmt),
>>>                  horizontalalignment="center",
>>>                  color="white"
>>>                  if cm[i, j] > thresh else "black")

>>>     plt.tight_layout()
>>>     plt.ylabel('True')
>>>     plt.xlabel('Predicted')
>>>
>>> # Llamada a la función
>>> plot_confusion_matrix(targetTest, targetPred, ['0', '1'], False, 'Matriz de Confusión', plt.cm.Reds)
```

Resultados de 30 ejecuciones

Exc.	Random Forest - SMOTE									
	Acc.	Pre.	Rec.	F1	VP	FN	FP	VN	TVP	TFP
1	0,84	0,84	0,84	0,84	10591	1933	2045	10479	0,8456	0,1632
2	0,84	0,85	0,84	0,84	10584	1912	2005	10547	0,8469	0,1597
3	0,84	0,85	0,84	0,84	10580	1911	2063	10494	0,8470	0,1642
4	0,84	0,85	0,84	0,84	10718	1903	2015	10412	0,8492	0,1621
5	0,84	0,84	0,84	0,84	10520	1960	2069	10499	0,8429	0,1646
6	0,84	0,84	0,84	0,84	10450	1976	2044	10578	0,8409	0,1619
7	0,84	0,84	0,83	0,84	10426	1970	2106	10546	0,8410	0,1664
8	0,84	0,84	0,83	0,84	10533	1935	2121	10459	0,8448	0,1686
9	0,84	0,85	0,83	0,84	10551	1868	2186	10443	0,8495	0,1730
10	0,84	0,85	0,83	0,84	10611	1903	2110	10424	0,8479	0,1683
11	0,84	0,84	0,84	0,84	10500	1994	2035	10519	0,8404	0,1620
12	0,84	0,84	0,84	0,84	10651	1966	1983	10448	0,8441	0,1595
13	0,84	0,84	0,84	0,84	10596	1955	2001	10496	0,8442	0,1601
14	0,84	0,85	0,84	0,84	10550	1878	2045	10575	0,8488	0,1620
15	0,84	0,85	0,83	0,84	10529	1904	2152	10463	0,8468	0,1705
16	0,84	0,85	0,84	0,84	10584	1897	2063	10504	0,8480	0,1641
17	0,84	0,85	0,84	0,84	10426	1943	2064	10615	0,8429	0,1627
18	0,84	0,85	0,84	0,84	10547	1903	2030	10568	0,8471	0,1611
19	0,84	0,85	0,83	0,84	10600	1901	2096	10451	0,8479	0,1670
20	0,84	0,84	0,84	0,84	10693	1991	1985	10379	0,8430	0,1605
21	0,84	0,85	0,83	0,84	10521	1879	2158	10490	0,8484	0,1706
22	0,84	0,85	0,84	0,84	10679	1869	2019	10481	0,8510	0,1615
23	0,84	0,85	0,84	0,84	10634	1872	2036	10506	0,8503	0,1623
24	0,84	0,84	0,84	0,84	10642	1946	2022	10438	0,8454	0,1622
25	0,84	0,85	0,83	0,84	10695	1881	2107	10365	0,8504	0,1689
26	0,84	0,85	0,84	0,84	10663	1917	1983	10485	0,8476	0,1590
27	0,84	0,84	0,84	0,84	10654	1970	1969	10455	0,8439	0,1584
28	0,84	0,85	0,84	0,84	10568	1919	2018	10543	0,8463	0,1606
29	0,84	0,84	0,83	0,84	10539	1953	2088	10468	0,8436	0,1662
30	0,84	0,84	0,83	0,83	10762	1941	2140	10205	0,8472	0,1733
Prom.	**0,84**	**0,8456**	**0,837**	**0,8396**	**10586,57**	**1925**	**2058,6**	**10477,83**	**0,8461**	**0,1642**
Desv.	**0**	**0,0050**	**0,0048**	**0,0018**	**83,0213**	**37,4700**	**57,2403**	**77,5247**	**0,0029**	**0,0042**

BACKPROPAGATION

Backpropagation - Sin balance

Lectura de datos y recuento

```
>>> # Importar paquete
>>> import pandas as pd

>>> # Carga del DataFrame
>>> df = pd.read_csv('./Datos/Data_Preprocemiento.csv')

>>> # Recuento
>>> data = df.drop(['capita_baja'], axis=1)
>>> target = df.loc[:,'capita_baja']
>>> baja = target.sum()
>>> activo = target.shape[0]- baja
>>> print('Bajas: ', baja, ' , Activos: ', activo )
```

Entrenamiento Backpropagation

```
# Importación paquete
>>> from sklearn.neural_network import MLPClassifier

>>> #1. División de datos en conjunto de evaluación y conjunto de entrenamiento
>>> from sklearn.model_selection import train_test_split
>>> porc_test = 0.2 # 80% training y 20% test
>>> dataTrain, dataTest, targetTrain, targetTest = train_test_split(data, target, test_size= porc_test)
>>> print("\t %d datos para training" %dataTrain.shape[0])
>>> print("\t %d datos para testing" %dataTest.shape[0])

>>> #2. Normalización/Estandarización de los datos
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> scaler.fit(dataTrain)
>>> dataTrain = scaler.transform(dataTrain)
>>> dataTest = scaler.transform(dataTest)

>>> #3. Inicialización del modelo
>>> solver = 'sgd'
>>> alpha = 0.0001
>>> activation = 'relu'
>>> hidden_layer_sizes = (100,)
>>> max_iter = 200
>>> tol = 1e-4
>>> verbose = False
>>> modelMLP = MLPClassifier(solver=solver, alpha=alpha, activation=activation,
hidden_layer_sizes=hidden_layer_sizes, max_iter=max_iter, tol=tol, verbose=verbose)

>>> #4. Entrenamiento del modelo
>>> modelMLP.fit(dataTrain, targetTrain)
>>> print("El modelo tardó %d iteraciones en entrenar" % modelMLP.n_iter_)

>>> #5. Predicción con el modelo
>>> targetPred = modelMLP.predict(dataTest)
>>> print("El modelo tardó %d iteraciones en entrenar" % modelMLP.n_iter_)
```

Evaluación del modelo

```
>>> # Importación de paquetes
>>> from sklearn.metrics import accuracy_score,recall_score, precision_score, f1_score
>>> import matplotlib.pyplot as plt
>>> from sklearn.metrics import confusion_matrix
>>> import numpy as np
>>> import itertools>>>

>>> # Accuracy, Precision, Recall, F1-Measure
>>> print("\nTest:")
```

```
>>> print("\t Accuracy: %.2f" % accuracy_score(targetTest, targetPred ))
>>> print("\t Precision: %.2f" % precision_score(targetTest, targetPred) )
>>> print("\t Recall: %.2f" % recall_score(targetTest, targetPred ))
>>> print("\t F1-Measure: %.2f" % f1_score(targetTest, targetPred))>>>

>>> # Matriz de confusión
>>> def plot_confusion_matrix(real_target, pred_target, classes, normalize=False, title='Matriz de Confusión', cmap=plt.cm.Reds):
>>>
>>>     cm= confusion_matrix(real_target, pred_target)
>>>
>>>     if normalize:
>>>         cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
>>>
>>>     plt.figure()
>>>     plt.imshow(cm, interpolation='nearest', cmap=cmap)
>>>     plt.title(title)
>>>     plt.colorbar()
>>>     tick_marks = np.arange(len(classes))
>>>     plt.xticks(tick_marks, classes, rotation=0)
>>>     plt.yticks(tick_marks, classes, rotation=0)

>>>     fmt = '.2f' if normalize else 'd'
>>>     thresh = cm.max() / 2.
>>>     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
>>>         plt.text(j, i, format(cm[i, j], fmt),
>>>                  horizontalalignment="center",
>>>                  color="white"
>>>                  if cm[i, j] > thresh else "black")

>>>     plt.tight_layout()
>>>     plt.ylabel('True')
>>>     plt.xlabel('Predicted')
>>>
>>> # Llamada a la función
>>> plot_confusion_matrix(targetTest, targetPred, ['0', '1'], False, 'Matriz de Confusión', plt.cm.Reds)
```

Resultados de 30 ejecuciones

Exc.	Backpropagation - Sin balance									
	Acc.	Pre.	Rec.	F1	VP	FN	FP	VN	TVP	TFP
1	0,82	0,63	0,42	0,51	11660	882	2072	1513	0,9296	0,5779
2	0,82	0,63	0,43	0,51	11764	886	1992	1485	0,9299	0,5729
3	0,82	0,64	0,45	0,53	11638	907	1957	1625	0,9277	0,5463
4	0,82	0,66	0,45	0,54	11588	874	2005	1660	0,9298	0,5470
5	0,81	0,64	0,42	0,51	11592	866	2120	1549	0,9304	0,5778
6	0,82	0,66	0,44	0,53	11676	816	2045	1590	0,9346	0,5625
7	0,82	0,65	0,45	0,53	11672	875	1982	1598	0,9302	0,5536
8	0,82	0,65	0,43	0,52	11702	834	2030	1561	0,9334	0,5653
9	0,82	0,66	0,46	0,54	11613	886	1942	1686	0,9291	0,5352
10	0,82	0,64	0,45	0,53	11576	923	2006	1622	0,9261	0,5529
11	0,82	0,65	0,45	0,53	11712	859	1972	1584	0,9316	0,5545
12	0,82	0,65	0,44	0,52	11676	865	2012	1574	0,9310	0,5610
13	0,82	0,65	0,44	0,53	11703	854	1982	1588	0,9319	0,5551
14	0,82	0,64	0,44	0,52	11705	874	1986	1562	0,9305	0,5597
15	0,82	0,65	0,43	0,52	11643	836	2080	1568	0,9330	0,5701
16	0,82	0,64	0,43	0,51	11744	840	2023	1520	0,9332	0,5709

17	0,82	0,66	0,46	0,54	11637	853	1980	1657	0,9317	0,5444
18	0,82	0,63	0,45	0,53	11652	940	1936	1599	0,9253	0,5476
19	0,82	0,67	0,44	0,53	11672	806	2035	1614	0,9354	0,5576
20	0,82	0,65	0,43	0,52	11662	834	2062	1569	0,9332	0,5678
21	0,82	0,63	0,45	0,53	11515	965	1988	1659	0,9226	0,5451
22	0,82	0,65	0,43	0,52	11681	848	2042	1556	0,9323	0,5675
23	0,82	0,65	0,44	0,52	11692	848	2020	1567	0,9323	0,5631
24	0,82	0,66	0,45	0,53	11657	823	2020	1627	0,9340	0,5538
25	0,83	0,64	0,45	0,53	11792	858	1929	1548	0,9321	0,5547
26	0,82	0,64	0,46	0,53	11609	906	1966	1646	0,9276	0,5442
27	0,82	0,64	0,45	0,53	11645	918	1954	1610	0,9269	0,5482
28	0,82	0,67	0,43	0,52	11667	795	2080	1585	0,9362	0,5675
29	0,82	0,65	0,47	0,54	11599	911	1933	1684	0,9271	0,5344
30	0,82	0,65	0,46	0,54	11639	899	1948	1641	0,9282	0,5427
Prom.	**0,82**	**0,6476**	**0,443**	**0,5263**	**11659,43**	**869,3667**	**2003,3**	**1594,9**	**0,9306**	**0,5567**
Desv.	**0,0026**	**0,0110**	**0,0127**	**0,0093**	**57,0563**	**39,9305**	**49,2706**	**49,7624**	**0,0031**	**0,0118**

Backpropagation - RandomOverSampler

Lectura de datos y recuento

```
>>> # Importar paquete
>>> import pandas as pd

>>> # Carga del DataFrame
>>> df = pd.read_csv('./Datos/Data_Preprocemiento.csv')

>>> # Recuento
>>> data = df.drop(['capita_baja'], axis=1)
>>> target = df.loc[:,'capita_baja']
>>> baja = target.sum()
>>> activo = target.shape[0]- baja
>>> print('Bajas: ', baja, ' , Activos: ', activo )
```

Balanceo de datos – RandomOverSampler

```
>>> #Importamos los paquetes
>>> from imblearn.over_sampling import RandomOverSampler

>>> #Inicialización de los métodos de sobremuestreo
>>> ros = RandomOverSampler()

>>> #ROS. Duplica muestras de la clase menos representadas
>>> dataRos, targetRos = ros.fit_resample(data,target)
>>> baja = targetRos.sum()
>>> activo = targetRos.shape[0]- baja
>>> print('Bajas: ', baja, ' , Activos: ', activo )
```

Entrenamiento Backpropagation

```
# Importación paquete
>>> from sklearn.neural_network import MLPClassifier

>>> #1. División de datos en conjunto de evaluación y conjunto de entrenamiento
>>> from sklearn.model_selection import train_test_split
>>> porc_test = 0.2 # 80% training y 20% test
>>> dataTrain, dataTest, targetTrain, targetTest = train_test_split(data, target, test_size= porc_test)
>>> print("\t %d datos para training" %dataTrain.shape[0])
>>> print("\t %d datos para testing" %dataTest.shape[0])
```

```
>>> #2. Normalización/Estandarización de los datos
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> scaler.fit(dataTrain)
>>> dataTrain = scaler.transform(dataTrain)
>>> dataTest = scaler.transform(dataTest)

>>> #3. Inicialización del modelo
>>> solver = 'sgd'
>>> alpha = 0.0001
>>> activation = 'relu'
>>> hidden_layer_sizes = (100,)
>>> max_iter = 200
>>> tol = 1e-4
>>> verbose = False
>>> modelMLP = MLPClassifier(solver=solver, alpha=alpha, activation=activation, hidden_layer_sizes=hidden_layer_sizes, max_iter=max_iter, tol=tol, verbose=verbose)

>>> #4. Entrenamiento del modelo
>>> modelMLP.fit(dataTrain, targetTrain)
>>> print("El modelo tardó %d iteraciones en entrenar" % modelMLP.n_iter_)

>>> #5. Predicción con el modelo
>>> targetPred = modelMLP.predict(dataTest)
>>> print("El modelo tardó %d iteraciones en entrenar" % modelMLP.n_iter_)
```

Evaluación del modelo

```
>>> # Importación de paquetes
>>> from sklearn.metrics import accuracy_score,recall_score, precision_score, f1_score
>>> import matplotlib.pyplot as plt
>>> from sklearn.metrics import confusion_matrix
>>> import numpy as np
>>> import itertools>>>

>>> # Accuracy, Precision, Recall, F1-Measure
>>> print("\nTest:")
>>> print("\t Accuracy: %.2f" % accuracy_score(targetTest, targetPred ))
>>> print("\t Precision: %.2f" % precision_score(targetTest, targetPred) )
>>> print("\t Recall: %.2f" % recall_score(targetTest, targetPred ))
>>> print("\t F1-Measure: %.2f" % f1_score(targetTest, targetPred))>>>

>>> # Matriz de confusión
>>> def plot_confusion_matrix(real_target, pred_target, classes, normalize=False, title='Matriz de Confusión', cmap=plt.cm.Reds):
>>>
>>>     cm= confusion_matrix(real_target, pred_target)
>>>
>>>     if normalize:
>>>         cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
>>>
>>>     plt.figure()
>>>     plt.imshow(cm, interpolation='nearest', cmap=cmap)
>>>     plt.title(title)
>>>     plt.colorbar()
>>>     tick_marks = np.arange(len(classes))
>>>     plt.xticks(tick_marks, classes, rotation=0)
>>>     plt.yticks(tick_marks, classes, rotation=0)

>>>     fmt = '.2f' if normalize else 'd'
>>>     thresh = cm.max() / 2.
>>>     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
>>>         plt.text(j, i, format(cm[i, j], fmt),
>>>                  horizontalalignment="center",
>>>                  color="white"
>>>                  if cm[i, j] > thresh else "black")
```

```
>>>     plt.tight_layout()
>>>     plt.ylabel('True')
>>>     plt.xlabel('Predicted')
>>>
>>> # Llamada a la función
>>> plot_confusion_matrix(targetTest, targetPred, ['0', '1'], False, 'Matriz de Confusión', plt.cm.Reds)
```

Resultados de 30 ejecuciones

Exc.	Backpropagation - RandomOverSampler									
	Acc.	Pre.	Rec.	F1	VP	FN	FP	VN	TVP	TFP
1	0,72	0,81	0,58	0,67	10840	1747	5241	7220	0,8612	0,4205
2	0,72	0,81	0,58	0,68	10781	1732	5239	7296	0,8615	0,4179
3	0,72	0,81	0,58	0,68	10853	1664	5277	7254	0,8670	0,4211
4	0,72	0,81	0,59	0,68	10771	1760	5139	7378	0,8595	0,4105
5	0,72	0,81	0,58	0,68	10841	1736	5216	7255	0,8619	0,4182
6	0,72	0,81	0,58	0,68	10773	1706	5256	7313	0,8632	0,4181
7	0,72	0,81	0,58	0,67	10692	1737	5310	7309	0,8602	0,4207
8	0,72	0,8	0,58	0,68	10722	1807	5200	7319	0,8557	0,4153
9	0,72	0,8	0,58	0,68	10801	1774	5182	7291	0,8589	0,4154
10	0,72	0,82	0,58	0,68	10780	1651	5293	7324	0,8671	0,4195
11	0,72	0,82	0,57	0,67	10945	1580	5380	7143	0,8738	0,4296
12	0,73	0,81	0,59	0,68	10826	1787	5054	7381	0,8583	0,4064
13	0,72	0,82	0,58	0,68	10813	1655	5289	7291	0,8672	0,4204
14	0,73	0,81	0,59	0,68	10705	1750	5135	7458	0,8594	0,4077
15	0,72	0,81	0,58	0,68	10787	1709	5286	7266	0,8632	0,4211
16	0,72	0,81	0,57	0,67	10857	1657	5374	7160	0,8675	0,4287
17	0,72	0,81	0,58	0,68	10786	1708	5219	7335	0,8632	0,4157
18	0,72	0,81	0,57	0,67	10857	1657	5374	7160	0,8675	0,4287
19	0,73	0,81	0,59	0,68	10840	1778	5070	7360	0,8590	0,4078
20	0,72	0,81	0,57	0,67	10827	1651	5368	7202	0,8676	0,4270
21	0,72	0,81	0,59	0,68	10692	1772	5187	7397	0,8578	0,4121
22	0,72	0,81	0,58	0,68	10799	1689	5250	7310	0,8647	0,4179
23	0,72	0,8	0,58	0,67	10827	1761	5259	7201	0,8601	0,4220
24	0,72	0,81	0,59	0,68	10642	1743	5226	7437	0,8592	0,4126
25	0,72	0,81	0,57	0,67	10994	1631	5384	7039	0,8708	0,4333
26	0,72	0,8	0,59	0,68	10541	1847	5144	7516	0,8509	0,4063
27	0,72	0,8	0,59	0,68	10720	1829	5108	7391	0,8542	0,4086
28	0,72	0,82	0,57	0,67	10806	1630	5404	7208	0,8689	0,4284
29	0,73	0,82	0,58	0,68	10955	1594	5277	7222	0,8729	0,4221
30	0,72	0,81	0,59	0,68	10833	1726	5178	7311	0,8625	0,4146
Prom.	**0,7213**	**0,81**	**0,581**	**0,677**	**10796,87**	**1715,6**	**5243,96**	**7291,56**	**0,8628**	**0,4183**
Desv.	**0,0035**	**0,0059**	**0,0071**	**0,0047**	**90,8935**	**67,5623**	**94,4916**	**102,1529**	**0,0054**	**0,0074**

Backpropagation - NearMiss

Lectura de datos y recuento

```
>>> # Importar paquete
>>> import pandas as pd

>>> # Carga del DataFrame
>>> df = pd.read_csv('./Datos/Data_Preprocemiento.csv')

>>> # Recuento
>>> data = df.drop(['capita_baja'], axis=1)
>>> target = df.loc[:,'capita_baja']
>>> baja = target.sum()
>>> activo = target.shape[0]- baja
>>> print('Bajas: ', baja, ' , Activos: ', activo )
```

Balanceo de datos – NearMiss

```
>>> #Importamos los paquetes
>>> from imblearn.under_sampling import NearMiss

>>> #Inicialización de los métodos de submuestreo NearMiss
>>> nm = NearMiss()

>>> #NearMiss. Elimina las muestras más cercanas de la clase más representada
>>> dataNm, targetNm = nm.fit_resample(data,target)
>>> baja = targetNm.sum()
>>> activo = targetNm.shape[0]- baja
>>> print('Bajas: ', baja, ' , Activos: ', activo )
```

Entrenamiento Backpropagation

```
# Importación paquete
>>> from sklearn.neural_network import MLPClassifier

>>> #1. División de datos en conjunto de evaluación y conjunto de entrenamiento
>>> from sklearn.model_selection import train_test_split
>>> porc_test = 0.2 # 80% training y 20% test
>>> dataTrain, dataTest, targetTrain, targetTest = train_test_split(data, target, test_size= porc_test)
>>> print("\t %d datos para training" %dataTrain.shape[0])
>>> print("\t %d datos para testing" %dataTest.shape[0])

>>> #2. Normalización/Estandarización de los datos
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> scaler.fit(dataTrain)
>>> dataTrain = scaler.transform(dataTrain)
>>> dataTest = scaler.transform(dataTest)

>>> #3. Inicialización del modelo
>>> solver = 'sgd'
>>> alpha = 0.0001
>>> activation = 'relu'
>>> hidden_layer_sizes = (100,)
>>> max_iter = 200
>>> tol = 1e-4
>>> verbose = False
>>> modelMLP = MLPClassifier(solver=solver, alpha=alpha, activation=activation,
hidden_layer_sizes=hidden_layer_sizes, max_iter=max_iter, tol=tol, verbose=verbose)

>>> #4. Entrenamiento del modelo
>>> modelMLP.fit(dataTrain, targetTrain)
>>> print("El modelo tardó %d iteraciones en entrenar" % modelMLP.n_iter_)

>>> #5. Predicción con el modelo
>>> targetPred = modelMLP.predict(dataTest)
>>> print("El modelo tardó %d iteraciones en entrenar" % modelMLP.n_iter_)
```

Evaluación del modelo

```
>>> # Importación de paquetes
>>> from sklearn.metrics import accuracy_score,recall_score, precision_score, f1_score
>>> import matplotlib.pyplot as plt
>>> from sklearn.metrics import confusion_matrix
>>> import numpy as np
>>> import itertools>>>

>>> # Accuracy, Precision, Recall, F1-Measure
>>> print("\nTest:")
>>> print("\t Accuracy: %.2f" % accuracy_score(targetTest, targetPred ))
>>> print("\t Precision: %.2f" % precision_score(targetTest, targetPred) )
>>> print("\t Recall: %.2f" % recall_score(targetTest, targetPred ))
>>> print("\t F1-Measure: %.2f" % f1_score(targetTest, targetPred))>>>

>>> # Matriz de confusión
>>> def plot_confusion_matrix(real_target, pred_target, classes, normalize=False, title='Matriz de Confusión', cmap=plt.cm.Reds):
>>>
>>>     cm= confusion_matrix(real_target, pred_target)
>>>
>>>     if normalize:
>>>         cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
>>>
>>>     plt.figure()
>>>     plt.imshow(cm, interpolation='nearest', cmap=cmap)
>>>     plt.title(title)
>>>     plt.colorbar()
>>>     tick_marks = np.arange(len(classes))
>>>     plt.xticks(tick_marks, classes, rotation=0)
>>>     plt.yticks(tick_marks, classes, rotation=0)

>>>     fmt = '.2f' if normalize else 'd'
>>>     thresh = cm.max() / 2.
>>>     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
>>>         plt.text(j, i, format(cm[i, j], fmt),
>>>                  horizontalalignment="center",
>>>                  color="white"
>>>                  if cm[i, j] > thresh else "black")

>>>     plt.tight_layout()
>>>     plt.ylabel('True')
>>>     plt.xlabel('Predicted')
>>>
>>> # Llamada a la función
>>> plot_confusion_matrix(targetTest, targetPred, ['0', '1'], False, 'Matriz de Confusión', plt.cm.Reds)
```

Resultados de 30 ejecuciones

Exc.	Backpropagation - NearMiss									
	Acc.	Pre.	Rec.	F1	VP	FN	FP	VN	TVP	TFP
1	0,77	0,76	0,78	0,77	2755	860	805	2787	0,7621	0,2241
2	0,77	0,76	0,79	0,77	2738	885	762	2822	0,7557	0,2126
3	0,78	0,77	0,79	0,78	2737	862	756	2852	0,7604	0,2095
4	0,77	0,77	0,77	0,77	2713	860	820	2814	0,7593	0,2256
5	0,77	0,76	0,78	0,77	2730	896	770	2811	0,7528	0,2150
6	0,77	0,76	0,77	0,77	2768	872	809	2758	0,7604	0,2268
7	0,77	0,76	0,78	0,77	2745	870	798	2794	0,7593	0,2221
8	0,76	0,75	0,77	0,76	2739	899	813	2756	0,7528	0,2277
9	0,77	0,76	0,79	0,77	2721	896	770	2820	0,7522	0,2144
10	0,77	0,77	0,78	0,77	2644	875	804	2884	0,7513	0,2180

11	0,77	0,76	0,77	0,77	2794	847	813	2753	0,7673	0,2279
12	0,77	0,77	0,78	0,78	2720	844	785	2858	0,7631	0,2154
13	0,77	0,77	0,77	0,77	2720	849	821	2817	0,7621	0,2256
14	0,77	0,77	0,79	0,78	2729	863	766	2849	0,7597	0,2118
15	0,77	0,77	0,78	0,77	2704	848	814	2841	0,7612	0,2227
16	0,77	0,76	0,78	0,77	2760	873	773	2801	0,7597	0,2162
17	0,77	0,75	0,79	0,77	2739	914	752	2802	0,7497	0,2115
18	0,77	0,77	0,77	0,77	2717	837	825	2828	0,7644	0,2258
19	0,77	0,76	0,8	0,78	2656	906	747	2898	0,7456	0,2049
20	0,77	0,76	0,78	0,77	2809	850	784	2764	0,7676	0,2209
21	0,77	0,76	0,79	0,78	2711	887	760	2849	0,7534	0,2105
22	0,77	0,77	0,78	0,77	2766	861	772	2808	0,7626	0,2156
23	0,77	0,76	0,77	0,77	2770	863	808	2766	0,7624	0,2260
24	0,77	0,75	0,8	0,77	2704	946	715	2842	0,7408	0,2010
25	0,77	0,76	0,78	0,77	2726	868	805	2808	0,7584	0,2228
26	0,77	0,76	0,78	0,77	2743	888	804	2772	0,7554	0,2248
27	0,78	0,77	0,79	0,78	2716	873	747	2871	0,7567	0,2064
28	0,77	0,76	0,8	0,78	2711	906	716	2874	0,7495	0,1994
29	0,76	0,75	0,78	0,77	2733	907	787	2780	0,7508	0,2206
30	0,77	0,76	0,8	0,78	2680	895	743	2889	0,7496	0,2045
Prom.	**0,77**	**0,762**	**0,783**	**0,7723**	**2729,93**	**876,66**	**781,46**	**2818,93**	**0,7569**	**0,2170**
Desv.	**0,0037**	**0,0066**	**0,0098**	**0,0050**	**34,8780**	**24,9887**	**30,7815**	**41,8601**	**0,0064**	**0,0084**

Backpropagation - SMOTE

Lectura de datos y recuento

```
>>> # Importar paquete
>>> import pandas as pd

>>> # Carga del DataFrame
>>> df = pd.read_csv('./Datos/Data_Preprocemiento.csv')

>>> # Recuento
>>> data = df.drop(['capita_baja'], axis=1)
>>> target = df.loc[:,'capita_baja']
>>> baja = target.sum()
>>> activo = target.shape[0]- baja
>>> print('Bajas: ', baja, ' , Activos: ', activo )
```

Balanceo de datos – SMOTE

```
>>> # Importación de librería
>>> from imblearn.over_sampling import SMOTE

>>> # Inicialización del método de sobremuestreo SMOTE
>>> smote = SMOTE()

>>> # SMOTE. Genera nuevas muestras sintéticas
>>> dataSmote, targetSmote = smote.fit_resample(data,target)
>>> bajaSmote = targetSmote.sum()
>>> activoSmote = targetSmote.shape[0]- bajaSmote
>>> print('Bajas: ', bajaSmote, ' , Activos: ', activoSmote )
```

Entrenamiento Backpropagation

```
# Importación paquete
>>> from sklearn.neural_network import MLPClassifier

>>> #1. División de datos en conjunto de evaluación y conjunto de entrenamiento
>>> from sklearn.model_selection import train_test_split
>>> porc_test = 0.2 # 80% training y 20% test
>>> dataTrain, dataTest, targetTrain, targetTest = train_test_split(data, target, test_size= porc_test)
>>> print("\t %d datos para training" %dataTrain.shape[0])
>>> print("\t %d datos para testing" %dataTest.shape[0])

>>> #2. Normalización/Estandarización de los datos
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> scaler.fit(dataTrain)
>>> dataTrain = scaler.transform(dataTrain)
>>> dataTest = scaler.transform(dataTest)

>>> #3. Inicialización del modelo
>>> solver = 'sgd'
>>> alpha = 0.0001
>>> activation = 'relu'
>>> hidden_layer_sizes = (100,)
>>> max_iter = 200
>>> tol = 1e-4
>>> verbose = False
>>> modelMLP = MLPClassifier(solver=solver, alpha=alpha, activation=activation,
hidden_layer_sizes=hidden_layer_sizes, max_iter=max_iter, tol=tol, verbose=verbose)

>>> #4. Entrenamiento del modelo
>>> modelMLP.fit(dataTrain, targetTrain)
>>> print("El modelo tardó %d iteraciones en entrenar" % modelMLP.n_iter_)

>>> #5. Predicción con el modelo
>>> targetPred = modelMLP.predict(dataTest)
>>> print("El modelo tardó %d iteraciones en entrenar" % modelMLP.n_iter_)
```

Evaluación del modelo

```
>>> # Importación de paquetes
>>> from sklearn.metrics import accuracy_score,recall_score, precision_score, f1_score
>>> import matplotlib.pyplot as plt
>>> from sklearn.metrics import confusion_matrix
>>> import numpy as np
>>> import itertools>>>

>>> # Accuracy, Precision, Recall, F1-Measure
>>> print("\nTest:")
>>> print("\t Accuracy: %.2f" % accuracy_score(targetTest, targetPred ))
>>> print("\t Precision: %.2f" % precision_score(targetTest, targetPred) )
>>> print("\t Recall: %.2f" % recall_score(targetTest, targetPred ))
>>> print("\t F1-Measure: %.2f" % f1_score(targetTest, targetPred))>>>

>>> # Matriz de confusión
>>> def plot_confusion_matrix(real_target, pred_target, classes, normalize=False, title='Matriz de
Confusión', cmap=plt.cm.Reds):
>>>
>>>     cm= confusion_matrix(real_target, pred_target)
>>>
>>>     if normalize:
>>>         cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
>>>
>>>     plt.figure()
>>>     plt.imshow(cm, interpolation='nearest', cmap=cmap)
>>>     plt.title(title)
>>>     plt.colorbar()
>>>     tick_marks = np.arange(len(classes))
>>>     plt.xticks(tick_marks, classes, rotation=0)
```

```
>>>     plt.yticks(tick_marks, classes, rotation=0)

>>>     fmt = '.2f' if normalize else 'd'
>>>     thresh = cm.max() / 2.
>>>     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
>>>         plt.text(j, i, format(cm[i, j], fmt),
>>>                  horizontalalignment="center",
>>>                  color="white"
>>>                  if cm[i, j] > thresh else "black")

>>>     plt.tight_layout()
>>>     plt.ylabel('True')
>>>     plt.xlabel('Predicted')
>>>
>>> # Llamada a la función
>>> plot_confusion_matrix(targetTest, targetPred, ['0', '1'], False, 'Matriz de Confusión', plt.cm.Reds)
```

Resultados de 30 ejecuciones

Exc.	Backpropagation - SMOTE									
	Acc.	Pre.	Rec.	F1	VP	FN	FP	VN	TVP	TFP
1	0,79	0,87	0,69	0,77	11320	1335	3819	8574	0,8945	0,3081
2	0,8	0,87	0,7	0,78	11317	1258	3777	8696	0,8999	0,3028
3	0,79	0,87	0,69	0,77	11260	1303	3858	8627	0,8962	0,3090
4	0,8	0,87	0,7	0,77	11338	1310	3759	8641	0,8964	0,3031
5	0,8	0,87	0,69	0,77	11381	1265	3828	8574	0,8999	0,3086
6	0,79	0,87	0,68	0,76	11365	1214	4011	8458	0,9034	0,3216
7	0,79	0,86	0,69	0,77	11097	1417	3869	8665	0,8867	0,3086
8	0,79	0,86	0,69	0,76	11311	1374	3878	8485	0,8916	0,3136
9	0,8	0,88	0,69	0,77	11383	1156	3900	8609	0,9078	0,3117
10	0,79	0,87	0,68	0,77	11214	1244	3886	8704	0,9001	0,3086
11	0,8	0,87	0,69	0,77	11064	1401	3719	8864	0,8876	0,2955
12	0,8	0,86	0,7	0,78	11111	1382	3812	8743	0,8893	0,3036
13	0,79	0,86	0,7	0,77	11175	1333	3943	8597	0,8934	0,3144
14	0,79	0,87	0,69	0,77	11418	1226	3771	8633	0,9030	0,3040
15	0,79	0,87	0,69	0,77	11152	1333	3932	8631	0,8932	0,3129
16	0,8	0,87	0,7	0,78	11128	1314	3727	8879	0,8943	0,2956
17	0,79	0,86	0,69	0,77	11256	1341	3884	8567	0,8935	0,3119
18	0,79	0,87	0,7	0,77	11135	1334	3817	8762	0,8930	0,3034
19	0,79	0,87	0,69	0,77	11125	1323	3874	8726	0,8937	0,3074
20	0,8	0,87	0,69	0,77	11263	1263	3826	8696	0,8991	0,3055
21	0,8	0,88	0,69	0,77	11186	1235	3898	8729	0,9005	0,3087
22	0,79	0,87	0,68	0,76	11268	1287	3985	8508	0,8974	0,3189
23	0,79	0,87	0,68	0,76	11297	1253	4043	8455	0,9001	0,3234
24	0,79	0,87	0,69	0,77	11184	1332	3885	8647	0,8935	0,3100
25	0,79	0,88	0,69	0,77	11122	1205	3977	8744	0,9022	0,3126
26	0,79	0,87	0,69	0,77	11181	1301	3894	8672	0,8957	0,3098
27	0,8	0,87	0,7	0,78	11237	1308	3731	8772	0,8957	0,2984
28	0,79	0,87	0,69	0,77	11182	1290	3907	8669	0,8965	0,3106
29	0,79	0,87	0,69	0,77	11298	1324	3840	8586	0,8951	0,3090

30	0,8	0,87	0,7	0,78	11173	1332	3757	8786	0,8934	0,2995
Prom.	**0,7936**	**0,8693**	**0,691**	**0,7703**	**11231,37**	**1299,76**	**3860,23**	**8656,63**	**0,8962**	**0,3084**
Desv.	**0,0049**	**0,0052**	**0,0063**	**0,0056**	**96,4513**	**59,5041**	**84,0384**	**106,8571**	**0,0047**	**0,0067**

AUTOENCODERS

Carga de datos

```
>>> from keras.layers import Dense, Input
>>> from keras import Model
>>> import numpy as np
>>> import pandas as pd

>>> def load_data():
>>>     # Muestras de la clase minoritaria
>>>     df = pd.read_csv('./Datos/Data_Preprocemiento.csv')
>>>     x_train = df[df['capita_baja'] == 1]
>>>     row_count, column_count = np.shape(x_train)

>>>     # Archivo con las muestras de la clase mayoritaria
>>>     x_test = df[df['capita_baja'] == 0]

>>>     return x_train,x_test,column_count
```

Coder y Decoder

```
>>> def DenseAutoencoder(input_dim,latent_dim):

>>>   def generate_encoder():
>>>     encoder_input = Input(shape=(input_dim,), name='encoder_input')
>>>     layer = encoder_input
>>>     if type(latent_dim ) == list:
>>>       for l in latent_dim:
>>>         layer = Dense(l, activation= "tanh", name='latent_vector'+str(l))(layer)
>>>         code = layer
>>>     else:
>>>       code = Dense(latent_dim, activation= "tanh", name='latent_vector')(encoder_input)
>>>     encoder = Model(encoder_input, code, name='encoder')
>>>     return encoder,encoder_input
>>>
>>>   def generate_decoder():
>>>     if type(latent_dim ) == list:
>>>       latent_input = Input(shape=(latent_dim[-1]), name='decoder_input')
>>>       layer = latent_input
>>>       for l in range(len(latent_dim)-2, -1, -1):
>>>         layer = Dense(latent_dim[l], activation= "tanh", name='decoder_vector'+str(l))(layer)
>>>     else:
>>>       latent_input = Input(shape=(latent_dim,), name='decoder_input')
>>>       layer = latent_input
>>>     decode = Dense(input_dim,activation= "tanh",name='decoder_output')(layer)
>>>     decoder = Model(latent_input, decode, name='decoder')
>>>     return decoder
>>>
>>>   encoder,encoder_input = generate_encoder()
>>>   decoder = generate_decoder()
>>>   # Crea el modelo del decodificador
>>>   autoencoder = Model(encoder_input, decoder(encoder(encoder_input)), name='autoencoder')
>>>
>>>   return autoencoder,encoder,decoder
```

Entrenamiento Autoencoders

```
>>> x_train, x_test, input_dim = load_data()

>>> # Tamaño capa intermedia
```

```
>>> latent_dim = [15,10]

>>> epochs = 10  #Número de épocas para entrenar el modelo.

>>> autoencoder, encoder, decoder = DenseAutoencoder(input_dim, latent_dim)
>>> autoencoder.compile(loss='mean_squared_error', optimizer='Adam')
>>> autoencoder.fit(x_train, x_train, validation_data=(x_test, x_test), epochs=epochs, batch_size=128)
```

Análisis de scores de reconstrucción

```
>>> x_decoded = autoencoder.predict(x_train)
>>> dif = (x_decoded - x_train)**2
>>> scores_train = np.sum(dif,1)
>>> print("Min train score: ", np.min(scores_train))
>>> print("Max train score: ", np.max(scores_train))

>>> x_decoded = autoencoder.predict(x_test)
>>> dif = (x_decoded - x_test)**2
>>> scores_test = np.sum(dif,1)
>>> print("Min test score: ", np.min(scores_test))
>>> print("Max test score: ", np.max(scores_test))

>>> root_path = './Datos/'

>>> salida = open(root_path + "Importantes_scores.txt", "w+")
>>> for s in scores_train:
>>>   salida.write(str(s) + "\n")
>>> salida.close()

>>> salida = open(root_path + "NoImportantes_scores.txt", "w+")
>>> for s in scores_test:
>>>   salida.write(str(s) + "\n")
>>> salida.close()
```

Obtención de las métricas

```
>>> umbral = 19000

>>> TP = 0; FN = 0; FP = 0; TN = 0
>>> for s in scores_train:
>>>   if s < umbral:
>>>     TP+= 1
>>>   else:
>>>     FN+= 1

>>> for s in scores_test:
>>>   if s < umbral:
>>>     FP+= 1
>>>   else:
>>>     TN+= 1

>>> print("Accuracy: " + str((TP+TN) / (TP+FN+FP+TN)))
>>> precision = TP / (TP+FP)
>>> print("Precision: " + str(precision))
>>> recall = TP / (TP+FN)
>>> print("Recall: " + str(recall))
>>> print("F1-Measure: " + str(2 * precision * recall / (precision + recall)))
```

Resultados de 30 ejecuciones

Exc.	Autoencoders				
	Accuracy	**Precision**	**Recall**	**F1**	**Umbral**
1	0,73	0,34	0,21	0,26	1.000
2	0,57	0,27	0,56	0,37	3.000
3	0,52	0,26	0,62	0,37	5.000

4	0,5	0,26	0,65	0,37	6.000
5	0,49	0,26	0,69	0,37	8.000
6	0,47	0,26	0,74	0,38	10.000
7	0,46	0,26	0,76	0,39	11.000
8	0,45	0,26	0,79	0,39	13.000
9	0,44	0,26	0,81	0,39	15.000
10	0,43	0,26	0,83	0,39	17.000
11	0,42	0,25	0,84	0,39	19.000
12	0,41	0,25	0,85	0,39	20.000
13	0,4	0,25	0,86	0,39	21.000
14	0,39	0,25	0,88	0,39	23.000
15	0,387	0,25	0,9	0,39	25.000
16	0,35	0,24	0,92	0,38	27.000
17	0,33	0,24	0,94	0,38	30.000
18	0,32	0,24	0,95	0,38	32.000
19	0,3	0,23	0,95	0,38	34.000
20	0,3	0,23	0,96	0,38	36.000
21	0,29	0,23	0,96	0,37	38.000
22	0,28	0,23	0,96	0,37	40.000
23	0,28	0,23	0,97	0,37	42.000
24	0,28	0,23	0,97	0,37	44.000
25	0,27	0,23	0,97	0,37	46.000
26	0,24	0,22	0,98	0,37	60.000
27	0,23	0,22	0,99	0,36	80.000
28	0,22	0,22	0,99	0,36	90.000
29	0,22	0,22	0,99	0,36	95.000
30	0,22	0,22	1	0,36	100.000
Promedio	**0,373233**	**0,245667**	**0,85**	**0,373**	**33033,33**