Técnica de muestreo de valores extremos para el manejo de clases desbalanceadas para la identificación de operaciones bancarias fraudulentas usando machine learning

Jorge Saavedra Garrido

22 de diciembre del 2022

Profesor Guía:

Dr. Rodrigo José Salas Fuentes
Escuela de Ingeniería C. Biomédica, Universidad de Valparaíso

Profesor Co-Guía:

Dr. Harvey Jezlid Rosas Quintero
Instituto de Estadística, Universidad de Valparaíso

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN ESTADÍSTICA

Facultad de Ciencias
Instituto de Estadística (IDEUV)

2022

Técnica de muestreo de valores extremos para el manejo de clases desbalanceadas para la identificación de operaciones bancarias fraudulentas usando machine learning

Decana:
Dra. Marisol Tejos

Directora del Magíster en Estadística:
Dra. Kerlyns Martínez Rodríguez

Profesor Guía:
Dr. Rodrigo Jose Salas Fuentes
Escuela de Ingenieria C. Biomédica, Universidad de Valparaíso

Profesor Co-Guía:
Dr. Harvey Jezlid Rosas Quintero
Instituto de Estadística, Universidad de Valparaíso

Facultad de ciencias
Instituto de Estadistica IDEUV

2022

Resumen

El fraude con tarjetas de crédito ha sido un problema que ha afectado a entidades financieras durante años, causando grandes pérdidas monetarias. Para detectar comportamientos anómalos o acciones sospechosas que incurren en pérdidas, el desarrollo de tecnologías de Machine Learning ha sido de gran importancia. Sin embargo, los conjuntos de datos disponibles para problemas de fraude bancario a menudo están altamente desbalanceados, lo que dificulta el aprendizaje de patrones de la clase minoritaria. Para abordar este problema, se han utilizado técnicas de submuestreo y sobremuestreo para equilibrar las clases. No obstante, a menudo se hace un equilibrio de clases en la fase de preprocesamiento antes de separar los datos en conjuntos de entrenamiento y test, lo que puede generar una correlación entre los datos y un rendimiento engañoso al evaluar los modelos. Por lo tanto, el objetivo de este trabajo es identificar errores en la implementación de técnicas de submuestreo y sobremuestreo para equilibrar clases en conjuntos de datos altamente desbalanceados y proponer una nueva técnica de submuestreo que considera los valores extremos de ambas clases utilizando la distancia de Mahalanobis. Esta medida de distancia tiene en cuenta la variabilidad de los datos y se utiliza comúnmente en problemas de clasificación para medir la similitud entre dos grupos. Nuestros resultados demuestran una mejora significativa en el rendimiento en comparación con las técnicas de balanceo de clases Smote, NearMiss y Submuestreo Aleatorio, alcanzando una precisión del 97% y un recall del 88%.

Agradecimientos

A mis padres, Hugo y Elizabeth, por los valores inculcados, la comprensión y apoyo para alcanzar mis metas.

A mi abuelita Bernarda, que siempre confió en mí y me dio aliento para terminar este proceso.

A mi abuelito Jorge, hermana Monserrat, primo Jorge y tio Mario que siempre me acompañan donde quiera que vaya.

A mis hermanos Hugo y Anais por hacerme la vida mas feliz.

A mi polola Naileth por acompañarme en todo este proceso y hacerme este camino mas fácil.

A los Profesores Rodrigo Salas y Harvey Rosas que a pesar de sus multiples ocupaciones me han brindado su apoyo haciebdo posible que concluya esta tesis.

Sinceramente a todos, muchas gracias.

Facultad de ciencias
Instituto de Estadística IDEUV

Índice general

Capítulo 1

INTRODUCCIÓN

Debido a la creciente digitalización y el auge de las aplicaciones móviles, la cantidad de transacciones con tarjetas de crédito ha aumentado considerablemente, debido al uso masivo de pagos digitales. A medida que crece el número de usuarios, los casos de fraude también han aumentado causando pérdidas de miles de millones de dólares. The Nilson Report informa que en el 2020 los emisores, comerciantes y adquirentes de transacciones comerciales y de cajeros automáticos perdieron en conjunto $28,580 millones por fraude con tarjetas de crédito. Pronosticando que las pérdidas globales seguirán aumentando año tras año y posiblemente alcanzarán los $49,32 mil millones en el 2030 [1].

Las técnicas de aprendizaje de máquinas han generado diversas soluciones al problema de deteccción de fraudes [2], [3], [4], [5], [6]. Debido a que las transacciones genuinas superan con creces los fraudes, se presenta un alto desequilibrio de clases, siendo un desafio aprender de este tipo de conjuntos [7], pues el objetivo es identificar instancias de fraude con la mayor certeza posible [8] y que el modelo de aprendizaje automático logre generalizar dicha clase. Al respecto, han surgido dos enfoques para mitigar este problema, uno a nivel de algoritmo, donde se penaliza una de las clases al momento de entrenar el modelo [9], [10] y otro a nivel de datos, donde se plantea un equilibrio de clases. En [11] se propone la técnica de sobremuestreo SMOTE, que consiste en el sobremuestreo de la clase minoritaria creando instancias sintéticas. En [12] hacen un sobremuestreo en el límite entre la clase minoritaria y la clase mayoritaria del conjunto de datos. Otro tipo de muestreo sintético adaptativo (ADASYN) es presentado en [8], utilizan una distribución ponderada para diferentes instancias de las clases minoritarias según su nivel de dificultad en el aprendizaje, [13] presentan una técnica de sobremuestreo basada en la distancia de Mahalanobis (MDO) la cual genera muestras sintéticas que preservan la estructura de la varianza y reducen el riesgo de superposición. Por otra parte, [14] estudiaron varios métodos diferentes para reducir instancias de las clases mayoritarias, [15] proponen un submuestreo llamado NearMiss que elimina los ejemplos de las clases mayoritarias tomando en cuenta los k-vecinos.

La literatura ha mostrado que muchos autores utilizan técnicas de balanceo de clases antes de separar el conjunto de datos en entrenamiento y test. Sin embargo, esta práctica puede llevar a un sobreajuste de los modelos debido a la intervención del conjunto de prueba en la fase de preprocesamiento. Por otro lado, separar el conjunto de datos en entrenamiento y test antes de aplicar cualquier técnica de balanceo de clases *solo* al conjunto de entrenamiento, evita la intervención del conjunto de test y, por lo tanto, evita la generación de correlación entre ambos conjuntos. En esta tesis, se analiza esta práctica

mediante una comparación entre ambos enfoques utilizando técnicas de muestreo como el submuestreo aleatorio, NearMiss y SMOTE, y se observa la correlación generada en cada caso. Además, se propone un método de *submuestreo* que considera las distancias de Mahalanobis, el cual arroja resultados superiores en comparación con las técnicas mencionadas anteriormente, alcanzando un 88 % de recall y un 92 % de precisión.

1.1. Objetivos

1.1.1. Objetivo general

Desarrollar una nueva técnica de muestreo para el manejo de clases desbalanceadas para la identificación de operaciones bancarias fraudulentas usando machine learning.

1.1.2. Objetivos específicos

- Identificar las características más relevantes del conjunto de datos en el contexto de transacciones fraudulentas.
- Desarrollar una nueva técnica de submuestreo que considera los valores extremos en el contexto de la detección de transacciones fraudulentas.
- Comparar y contrastar diferentes técnicas de submuestreo y sobremuestreo utilizando métricas de desempeño como la precisión, el recall y la sensibilidad.

1.2. Pregunta e hipótesis de investigación

Los fraudes causados por las tarjetas de crédito han costado a los consumidores y bancos miles de millones de dólares en todo el mundo. Incluso después de numerosos mecanismos para detener el fraude, los estafadores intentan continuamente encontrar nuevas formas y trucos para incurrir en este delito. Por lo tanto, para identificar estos fraudes se necesita un potente sistema que los detecte antes de que estos se produzcan. Más aún, se debe hacer que los modelos aprendan de los fraudes cometidos en el pasado y sean capaces de adaptarse a futuros nuevos métodos de fraude. Es por esto que se plantea la siguiente pregunta de investigación:

- ¿Es posible mejorar el desempeño en clasificación de transacciones fraudulentas?
- La hipótesis de investigación propuesta es:

  Es posible implementar técnicas de submuestreo o sobremuestreo de clases para mejorar el desempeño en la clasificación de transacciones fraudulentas con tarjetas de crédito.

Capítulo 2

TÉCNICAS DE MACHINE LEARNING

2.1. Modelos lineales para regresión

El propósito de la regresión es predecir el valor de una o más variables objetivo t dado un vector D-dimensional $\mathbf{x}$ de las variables de entrada. El polinomio es un ejemplo específico de una amplia clase de funciones llamadas modelos de regresión lineal, que comparten la propiedad de ser funciones lineales de parámetros ajustables. Sin embargo, se puede obtener una clase de funciones mucho más útil tomando combinaciones lineales de un conjunto fijo de funciones no lineales de las variables de entrada, conocidas como *funciones base*.

Sea $\{x_n\}$ un conjunto de datos de entrenamiento que comprende N observaciones, junto con los correspondientes valores objetivo $\{t_n\}$. El propósito es estimar el valor de t para un nuevo valor de entrada $\mathbf{x}$. En el enfoque más simple, esto se puede hacer construyendo directamente una función apropiada $y(\mathbf{x})$ que toma en cuenta las variables de entrada $\mathbf{x}$ para predecir las variables objetivos t. De manera más general, desde una perspectiva probabilística, el objetivo es modelar la distribución predictiva $p(t|\mathbf{x})$, ya que esta expresa la incertidumbre sobre el valor de t. A partir de esta distribución condicional se puede hacer predicciones de t de tal manera que minimice el valor esperado de una función de pérdida elegida adecuadamente.

Definición 1 *El aprendizaje se dice supervisado cuando el objetivo es aprender una función a partir de ejemplos de sus entradas y salidas.*

El entrenamiento de modelos lineales para regresión y redes neuronales son ejemplos de aprendizaje supervisado.

Definición 2 *El aprendizaje se dice no supervisado cuando el objetivo es aprender a partir de patrones de entrada para los que no se especifican los valores de salida.*

2.1.1. Funciones base en modelos lineales

Si el resultado deseado para un problema de machine learning consta de una o más variables continuas, se llama regresión. Un ejemplo de un problema de regresión sería la predicción del rendimiento en un proceso de fabricación de productos químicos en el que las entradas consisten en las concentraciones de reactivos, la temperatura y la presión. Por otra parte, en algunos problemas de reconocimiento de patrones, los datos de entrenamiento consisten en un conjunto de vectores de entrada $\mathbf{x}$ sin ningún valor objetivo correspondiente, por ejemplo, descubrir grupos similares dentro de los datos, llamado *"clustering"*, o determinar la distribución de datos dentro del espacio de entrada. Dado esto surgen los conceptos de *aprendizaje supervisado* y *aprendizaje no supervisado.*

El modelo más simple para la regresión esta compuesto de una combinación lineal de las variables de entrada $\mathbf{x}$, tal que

$$y\left(\mathbf{x}, \mathbf{w}\right) = w_0 + w_1 x_1 + ... + w_D x_D, \tag{2.1}$$

donde $\mathbf{x} = \left(x_1, ..., x_D\right)^T$. La propiedad clave de este modelo es que es una función lineal de los parámetros $w_0, ..., w_D$. Sin embargo, también es una función lineal de las variables de entrada x_i. Este modelo, se puede extender considerando una combinación de funciones no lineales respecto a la variable de entrada, de la forma

$$y\left(\mathbf{x}, \mathbf{w}\right) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j\left(\mathbf{x}\right), \tag{2.2}$$

donde M es el número total de parámetros, los $\phi_j\left(\mathbf{x}\right)$ son conocidos como *funciones bases* y el parámetro w_0 recibe el nombre de sesgo, el cual permite desplazar los datos. A menudo, es conveniente definir una función *dummy* [1] $\phi_0\left(x\right) = 1$, tal que

$$y\left(\mathbf{x}, \mathbf{w}\right) = \sum_{j=0}^{M-1} w_j \phi_j\left(\mathbf{x}\right) = \mathbf{w}^T \boldsymbol{\phi}\left(x\right).$$

Usar funciones base no lineales, implica que $y(\mathbf{x}, \mathbf{w})$ sea una función no lineal del vector de entrada $\mathbf{x}$. Sin embargo, las funciones de la forma (2.2) se denominan modelos lineales porque son lineales en $\mathbf{w}$. Es esta linealidad en los parámetros la que simplifica enormemente el análisis de esta clase de modelos.

Ejemplo 3 *Una elección clásica es la función sigmoidal logística*

$$\sigma\left(\mathbf{x}\right) = \frac{1}{1 + e^{-\mathbf{x}}}. \tag{2.3}$$

El término "sigmoide" significa en forma de S, como se puede observar en la Figura 3.

[1] Una función dummy es una función matemática que toma uno o varios valores discretos y devuelve un valor binario (es decir, un valor que toma solo dos valores, 1 o 0). Esta función se utiliza a menudo en análisis estadísticos y de aprendizaje automático para representar variables categóricas (es decir, variables que toman un número limitado de valores discretos) de manera numérica.

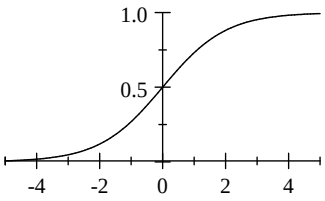


Figura 3. Función base sigmoidal.

Esta función transforma el intervalo $(-\infty, +\infty)$ *en el intervalo* $(0, 1)$*. Por otra parte, cuando* $|\mathbf{x}|$ *es pequeño, la ecuación (2.3) se puede aproximar por una función lineal.*

Ejemplo 4 *Una función base similar es la tangente hiperbólica*

$$\tanh \mathbf{x} = \frac{e^{\mathbf{x}} - e^{-\mathbf{x}}}{e^{\mathbf{x}} + e^{-\mathbf{x}}}, \tag{2.4}$$

que difiere de la función sigmoidal logística $\sigma(\mathbf{x})$ *solo por transformaciones lineales, de modo que*

$$2\sigma(2\mathbf{x}) - 1 = \tanh \mathbf{x}.$$

Por lo tanto, una combinación lineal de una función sigmoidal logística es equivalente a una combinación lineal de funciones $\tanh\mathbf{x}$*.*

Ejemplo 5 *Otra función base es introducida por McCulloch y Pitts (1943) para modelar el comportamiento de una neurona en un sistema nervioso llamada función escalonada de Heaviside*

$$g(a) = \begin{cases} 1 & si\ a \geq 0 \\ 0 & si\ a < 0 \end{cases} \tag{2.5}$$

donde $a = \mathbf{w}^T\mathbf{x} + \mathbf{w}_0$*. Las entradas* x_i *representan el nivel de actividad de cada neurona, los pesos* w_i *representan la fuerza de las conexiones sinápticas entre las neuronas,* w_0 *un umbral a partir del cual la neurona dispara un nuevo potencial de acción. Este modelo inspirado en la biología se usó para el reconocimiento estadístico de patrones. Rosenblatt (1962) estudió redes de unidades con función base escalonada que llamó perceptrones.*

Ejemplo 6 *Usando la ecuación (2.5), se puede representar funciones booleanas básicas. Esta fue una de las motivaciones de McCulloch y Pitts para el diseño de unidades individuales.*

$$\begin{aligned} AND(x_1, x_2) &= g(x_1 + x_2 - 1{,}5) \\ OR(x_1, x_2) &= g(x_1 + x_2 - 0{,}5). \end{aligned} \tag{2.6}$$

Esto quiere decir que usando estas unidades se puede construir una ***red neuronal*** *que calcule cualquier función booleana de las entradas.*

Ejemplo 7 *Rectificadora (ReLU - Rectified Linear Unit): Esta función de activación la introdujo por primera vez en el año* 2000 *Hahnloser, con motivaciones biológicas y matemáticas. Se viene usando en redes neuronales convolucionales por su eficiencia en el entrenamiento, siendo más utilizada que la ya ampliamente extendida función logística sigmoidal. La función rectificadora es, por tanto, una de las funciones de activación más populares para deep learning.*

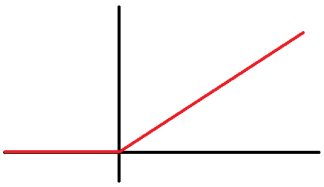


Figura 7. Función rectificadora dada por la función g del ejemplo 5.

2.1.2. Máxima verosimilitud y mínimos cuadrados

En esta sección se discute el enfoque de mínimos cuadrados y su relación con la máxima verosimilitud. Sea t una variable objetivo dada por una función determinista $y(x, \mathbf{w})$ con ruido aditivo Gaussiano de modo que

$$t = y\left(\mathbf{x}, \mathbf{w}\right) + \varepsilon, \tag{2.7}$$

donde ε es una variable aleatoria Gaussiana con media cero y precisión (varianza inversa) β. De esta manera, se tiene que

$$p\left(t|\mathbf{x}, \mathbf{w}, \beta\right) = \mathcal{N}\left(t|y\left(\mathbf{x}, \mathbf{w}\right), \beta^{-1}\right). \tag{2.8}$$

Al asumir una función de pérdida cuadrática, la predicción óptima para un nuevo valor de $\mathbf{x}$ será dada por la media condicional de la variable objetivo t. En el caso de una distribución normal de la forma (2.8), la media condicional es simplemente

$$\mathbb{E}\left(t|\mathbf{x}\right) = \int tp\left(t|\mathbf{x}\right) dt = y\left(\mathbf{x}, \mathbf{w}\right). \tag{2.9}$$

Sea $\mathbf{X} = \{\mathbf{x}_1, ..., x_N\}$ un conjunto de datos de entrada con sus correspondientes valores objetivo $\mathbf{t} = \{t_1, ..., t_N\}$. Si estos puntos de datos se extraen independientemente de la distribución (2.8), se obtiene la expresión para la función de verosimilitud, la cual se encuentra en función de los parámetros ajustables $\mathbf{w}$ y β en la forma

$$p\left(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta\right) = \prod_{n=1}^{N} \mathcal{N}\left(t_n|\mathbf{w}^T \phi\left(\mathbf{x}_n\right), \beta^{-1}\right). \tag{2.10}$$

Al aplicar logaritmo en la función de verosimilitud se obtiene

$$\begin{aligned}\ln p\left(\mathbf{t}|\mathbf{w},\beta\right) &= \sum_{n=1}^{N} \ln \mathcal{N}\left(t_n|\mathbf{w}^T\phi\left(\mathbf{x}_n\right),\beta^{-1}\right) \\ &= \frac{N}{2}\ln\beta - \frac{N}{2}\ln 2\pi - \frac{\beta}{2}\sum_{n=1}^{N}\left\{t_n - \mathbf{w}^T\phi\left(\mathbf{x}_n\right)\right\}^2.\end{aligned} \tag{2.11}$$

Definición 8 *Se define la función suma de error cuadrado como*

$$E_D\left(\mathbf{w}\right) = \frac{1}{2}\sum_{n=1}^{N}\left\{t_n - \mathbf{w}^T\phi\left(\mathbf{x}_n\right)\right\}^2, \tag{2.12}$$

la cual se extrae de (2.11).

Observación 9 *En lugar de maximizar el logaritmo de la verosimilitud, se puede minimizar de manera equivalente el logaritmo negativo de la máxima verosimilitud. En consecuencia, maximizar la verosimilitud es equivalente (respecto a la determinación de* $\mathbf{w}$*) a minimizar la función suma de error cuadrado definida por la ecuación (2.12).*

Una vez escrita la función de verosimilitud, se puede usar la máxima verosimilitud para estimar los valores de $\mathbf{w}$ y β. Considerando la maximización con respecto a $\mathbf{w}$, el gradiente de (2.11) toma la forma

$$\nabla \ln p\left(\mathbf{t}|\mathbf{w},\beta\right) = \sum_{n=1}^{N}\left\{t_n - \mathbf{w}^T\phi\left(\mathbf{x}_n\right)\right\}\phi^T\left(\mathbf{x}_n\right), \tag{2.13}$$

igualando a cero y despejando $\mathbf{w}$ se obtiene

$$\begin{aligned}\mathbf{w}_{MV} &= \left(\sum_{n=1}^{N}\phi\left(\mathbf{x}_n\right)\phi^T\left(\mathbf{x}_n\right)\right)^{-1}\sum_{n=1}^{N}t_n\phi^T\left(\mathbf{x}_n\right) \\ &= \left(\Phi^T\Phi\right)^{-1}\Phi^T\mathbf{t},\end{aligned} \tag{2.14}$$

las cuales son conocidas como, *ecuaciones normales* para el problema de mínimos cuadrados.

Definición 10 *Se define la matriz de diseño* Φ*, de orden* $N \times M$*, como*

$$\Phi = \begin{pmatrix} \phi_0\left(\mathbf{x}_1\right) & \phi_1\left(\mathbf{x}_1\right) & \cdots & \phi_{M-1}\left(\mathbf{x}_1\right) \\ \phi_0\left(\mathbf{x}_2\right) & \phi_1\left(\mathbf{x}_2\right) & \cdots & \phi_{M-1}\left(\mathbf{x}_2\right) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0\left(\mathbf{x}_N\right) & \phi_1\left(\mathbf{x}_N\right) & \cdots & \phi_{M-1}\left(\mathbf{x}_N\right) \end{pmatrix}, \tag{2.15}$$

cuyos elementos son dados por $\Phi_{nj} = \phi_j\left(\mathbf{x}_n\right)$.

Si se hace explícito el parámetro de sesgo, entonces la función de error (2.12) se convierte en

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}_0 - \sum_{j=1}^{M-1}\mathbf{w}_j^T\phi_j(\mathbf{x}_n)\}^2, \tag{2.16}$$

luego derivando con respecto a w_0 e igualando a cero, se obtiene

$$\mathbf{w}_0 = \frac{1}{N}\left(\sum_{n=1}^{N}t_n - \sum_{j=1}^{M-1}\sum_{n=1}^{N}\mathbf{w}_j^T\phi_j(\mathbf{x}_n)\right) = \bar{t} - \sum_{j=1}^{M-1}\mathbf{w}_j^T\overline{\phi}_j, \tag{2.17}$$

donde

$$\bar{t} = \frac{1}{N}\sum_{n=1}^{N}t_n \quad \text{y} \quad \bar{\phi}_j = \frac{1}{N}\sum_{n=1}^{N}\phi_j(\mathbf{x}_n). \tag{2.18}$$

Por lo tanto, el sesgo w_0 compensa la diferencia entre los promedios (sobre el conjunto de entrenamiento) de los valores objetivo t_n y la suma ponderada de los promedios de los valores de la función base en (2.17).

La maximización de la función de log-verosimilitud (2.11) con respecto al parámetro de precisión β es dada por

$$\frac{1}{\beta_{MV}} = \frac{1}{N}\sum_{n=1}^{N}\left\{t_n - \mathbf{w}_{MV}^T\phi(\mathbf{x}_n)\right\}^2. \tag{2.19}$$

2.1.3. Aprendizaje Secuencial

Las técnicas por lotes, como la solución de máxima verosimilitud (2.14), implica procesar todo el conjunto de entrenamiento de una sola vez, que computacionalmente puede ser costoso para grandes cantidades de datos. Si se tiene un conjunto de datos suficientemente grande, puede ser recomendable usar algoritmos secuenciales, también conocidos como algoritmos en línea, donde los puntos de datos se consideran uno a la vez, y los parámetros del modelo se van actualizando constantemente. El aprendizaje secuencial también es apropiado para aplicaciones en tiempo real, en que las observaciones de datos llegan en una secuencia continua, y las predicciones deben hacerse antes de que se vean todos los puntos de datos. Se puede obtener un algoritmo de aprendizaje secuencial aplicando la técnica de *descenso de gradiente estocástico*, también conocida como *descenso de gradiente secuencial*, de la siguiente manera.

Si la función de error comprende una suma sobre los puntos de datos

$$E = \sum_n E_n, \tag{2.20}$$

entonces después de la presentación del patrón n, el algoritmo de descenso de gradiente estocástico actualiza el vector de parámetros $\mathbf{w}$ usando

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - \eta\nabla E_n, \tag{2.21}$$

donde τ denota el número de iteraciones y η es la tasa de aprendizaje (cuanto afecta el gradiente a la actualización de los parámetros en cada iteración). El valor de $\mathbf{w}$ comienza con el vector $\mathbf{w}^{(0)}$. Para el caso de la función suma de error cuadrado de la ecuación (2.12) se tiene

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \eta\left(t_n - \mathbf{w}^{(\tau)T}\phi_n\right)\phi_n, \tag{2.22}$$

con $\phi_n = \phi_n(\mathbf{x}_n)$. Esto se conoce como *mínimos cuadrados medios* o el *algoritmo LMS*. El valor de η debe elegirse con cuidado para garantizar que el algoritmo converja, ya que un valor muy pequeño puede hacer que el proceso de optimización sea muy lento, pues los pasos de actualización de los parámetros serán muy pequeños. Por otro lado, un valor de tasa de aprendizaje muy grande puede hacer que el proceso sea inestable y puede hacer que el modelo no converja correctamente.

2.1.4. Mínimos cuadrados regularizados

En un conjunto de datos de entrada, muchas veces se introducen muestras atípicas o muestras que pueden no ser del todo representativas. Cuando se *"sobreentrena"* el modelo, el algoritmo estará considerando como válidos solo los datos idénticos a los del conjunto de entrenamiento, incluidos sus defectos, siendo incapaz de distinguir entradas buenas como fiables si se salen un poco de los rangos ya prestablecidos. Por otra parte, puede ocurrir que el modelo de *Machine Learning* sea demasiado simple y, en consecuencia, ocurre un ajuste insuficiente repercutiendo en la precisión. Esto suele suceder cuando se tiene pocos datos para construir un modelo preciso, o bien, cuando se intenta construir un modelo lineal con datos no lineales (ver Figura 2.1.4).

Definición 11 *Se dice sobreajuste o "overfitting" cuando un modelo memoriza los datos, pero no los generaliza, es decir, sólo se ajustará a aprender los casos particulares que se le enseña al modelo siendo incapaz de reconocer nuevos datos de entrada.*

Definición 12 *Se dice bajo ajuste o "underfitting" cuando un modelo de aprendizaje automático no puede ajustarse adecuadamente a los datos de entrenamiento.*

El bajo ajuste se debe a que el modelo es demasiado simple o no tiene suficiente capacidad para capturar la complejidad de los datos. Como resultado, el modelo tiene un rendimiento bajo tanto en los datos de entrenamiento como en los datos de prueba.

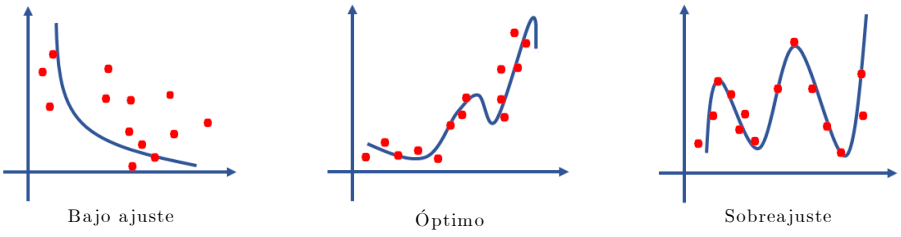


Figura 2.1.4. Ilustración de modelos con ajuste bajo, óptimo y sobreajustado.

La idea de agregar un término de regularización a una función de error es controlar el sobreajuste del modelo, de modo que la función de error total a minimizar tome la forma

$$E_D(\mathbf{w}) + \lambda E_W(\mathbf{w}), \tag{2.23}$$

donde λ es el coeficiente de regularización que controla la importancia relativa del error $E_D(\mathbf{w})$ dependiente de los datos y el término de regularización $E_W(\mathbf{w})$. Esta penalización mantiene los pesos pequeños, evitando que la red utilice pesos que no necesite realmente y así, reducir el sobre aprendizaje (sobreajuste) evitando que la red se ajuste demasiado al conjunto de entrenamiento. Una de las formas más simples de un regularizador viene dada por la suma de cuadrados de los elementos del vector peso

$$E_W(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}. \tag{2.24}$$

Si se considera la función suma de error cuadrado dado por

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\right)^2, \tag{2.25}$$

entonces la función de error total es

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\right)^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} = E_D(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}. \tag{2.26}$$

Si se calcula el vector gradiente de 2.26, se obtiene

$$\nabla\widetilde{E}(\mathbf{w}) = \nabla E_D(\mathbf{w}) + \lambda\mathbf{w}. \tag{2.27}$$

En consecuencia, utilizando descenso del gradiente, la regla para actualizar los sesgos no cambia, pero para los pesos pasa a ser

$$\begin{aligned}\mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - \eta\nabla\widetilde{E} \\ &= \mathbf{w}^{(\tau)} - \eta\nabla E_D - \eta\lambda\mathbf{w}^{(\tau)} \\ &= (1-\eta\lambda)\,\mathbf{w}^{(\tau)} - \eta\nabla E_D.\end{aligned} \tag{2.28}$$

Esto no es muy diferente a la regla usual usada en descenso de gradiente vista anteriormente, excepto que se escala los pesos $\mathbf{w}$ por el factor $(1-\eta\lambda)$ antes de actualizar, haciéndolos más pequeños. Es por esta razón, que $E_W(\mathbf{w})$ se conoce como *caída de peso o weight decay,* ya que este en los algoritmos de aprendizaje secuencial alienta a los valores de peso a decrecer.

La ventaja de este regularizador es que la función de error sigue siendo cuadrática, por lo que su minimizador exacto se puede encontrar en forma cerrada. Específicamente, al calcular el gradiente de (2.26) con respecto a $\mathbf{w}$, igualando a cero y despejando $\mathbf{w}$, se obtiene

$$\mathbf{w} = \left(\lambda I + \Phi^T\Phi\right)^{-1}\Phi^T\mathbf{t}, \tag{2.29}$$

la cual es una extensión de (2.14). Un regularizador más general toma la forma

$$\frac{1}{2}\sum_{n=1}^{N}\left(t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|\mathbf{w}_j|^q. \tag{2.30}$$

Cuando $q = 2$, se tiene el regularizador (2.26). La Figura **??** muestra los contornos de la función de regularización para diferentes valores de q.

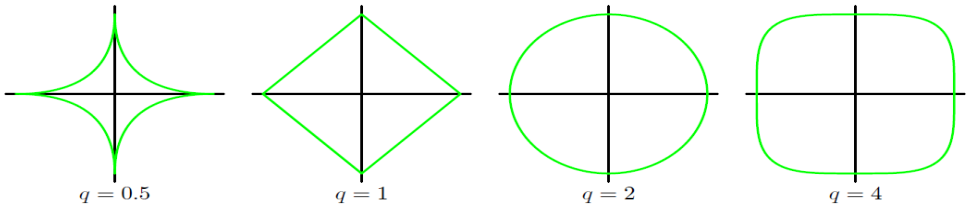


Figura **??**. Contornos del término de regularización para varios valores del parámetro q [16].

2.2. Modelos lineales para clasificación

El objetivo en problemas de clasificación es tomar un vector de entrada $\mathbf{x}$ y asignarlo a una de las K clases discretas $\mathcal{C}_k$, donde $k = 1, ..., K$. En el escenario más común, las clases se consideran disjuntas, de modo que cada entrada se asigna a una sola clase. El espacio de entrada se divide en *regiones de decisión* cuyas fronteras se denominan *límites de decisión* o *superficies de decisión*. En esta sección, se consideran modelos lineales de clasificación, donde las superficies de decisión son funciones lineales del vector de entrada $\mathbf{x}$ y, por lo tanto, están definidas por hiperplanos $(D - 1)$- dimensionales dentro del espacio de entrada D-dimensional. Para problemas de regresión, la variable objetivo t era simplemente un vector de números reales mientras que, en el caso de clasificación, existen diversas formas de utilizar valores objetivo para representar etiquetas de clase. Para los modelos probabilísticos lo más conveniente, en el caso de problemas de dos clases, es la representación binaria en la que hay una única variable objetivo $t \in \{0, 1\}$, tal que $t = 1$ representa la clase $\mathcal{C}_1$ y $t = 0$ representa la clase $\mathcal{C}_2$. El valor de t se puede interpretar como la probabilidad de que la clase sea $\mathcal{C}_1$ o $\mathcal{C}_2$tomando valores 0 o 1.

En el modelo de regresión lineal visto en la sección anterior, la predicción de $y(\mathbf{x}, \mathbf{w})$ estaba dada por una función lineal de los parámetros $\mathbf{w}$. En el caso más simple, el modelo también es lineal en las variables de entrada y, por lo tanto, toma la forma $y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \mathbf{w}_0$, de modo que y es un número real. Sin embargo, para problemas de clasificación, se desea predecir etiquetas de clase discretas o, en general, probabilidades a posteriori que se encuentran en el rango $(0, 1)$. Para lograr esto, se considera una generalización donde se transforma la función lineal de $\mathbf{w}$ usando una función no lineal $f(\cdot)$ de modo que

$$y(\mathbf{x}) = f\left(\mathbf{w}^T\mathbf{x} + \mathbf{w}_0\right). \tag{2.31}$$

En la literatura de aprendizaje automático o "machine learning" en inglés, $f(\cdot)$ se conoce como *función de activación*. Las superficies de decisión corresponden a $y(\mathbf{x}) =$*constante,* lo que implica que $\mathbf{w}^T\mathbf{x}+\mathbf{w}_0 =$ *constante* y, por lo tanto, las superficies de decisión son funciones lineales de $\mathbf{x}$, incluso si la función $f(\cdot)$ no es lineal. Sin embargo, en contraste con los modelos utilizados para la regresión, ya no son lineales en los parámetros debido a la presencia de la función no lineal $f(\cdot)$. Esto conducirá a propiedades analíticas

y computacionales más complejas que para los modelos de regresión lineal. Sin embargo, estos modelos siguen siendo relativamente simples en comparación con modelos no lineales más generales.

2.2.1. Función discriminante

Definición 13 *Un discriminante es una función que toma un vector de entrada* $\mathbf{x}$ *y lo asigna a una de las* K *clases* $\mathcal{C}_k$*.*

En esta Sección se prestará atención particularmente a los discriminantes lineales, es decir, aquellas funciones donde las superficies de decisión son hiperplanos. Primero, se analiza el caso de dos clases y luego se extiende a $K > 2$ clases.

Dos clases

Definición 14 *Se dice que los puntos de datos son linealmente separables si dos clases* $\mathcal{C}_j$ *y* $\mathcal{C}_k$ *pueden ser separadas por un hiperplano.*

Ejemplo 15 *Las funciones OR y AND son linealmente separables. En cambio la función XOR no es linealmente separable, como muestra la Figura 15 .*

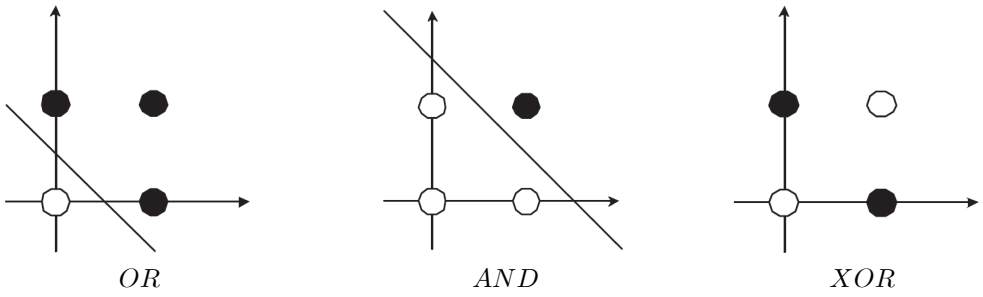


Figura 15. Funciones OR, AND y XOR [44].

La representación más simple de una función discriminante es obtenida tomando una función lineal de un vector de entrada, tal que

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \mathbf{w}_0, \tag{2.32}$$

donde $\mathbf{w}$ es llamado vector de peso y $\mathbf{w}_0$ sesgo (no confundir con el sesgo estadístico). Un vector de entrada $\mathbf{x}$ se asigna a la clase $\mathcal{C}_1$ si $y(\mathbf{x}) > 0$ y a la clase $\mathcal{C}_2$ si $y(\mathbf{x}) < 0$. Por lo tanto, la correspondiente frontera de decisión se define por la relación $y(\mathbf{x}) = 0$, que corresponde a un hiperplano (D-1)-dimensional dentro del espacio de entrada D-dimensional. Sean $\mathbf{x}_A$ y $\mathbf{x}_B$ dos puntos ubicados en la superficie de decisión. Dado que $y(\mathbf{x}_A) = y(\mathbf{x}_B) = 0$, se tiene que

$$\mathbf{w}^T(\mathbf{x}_A - \mathbf{x}_B) = 0, \tag{2.33}$$

por lo tanto, el vector $\mathbf{w}$ es ortogonal a todos los vectores que se encuentran dentro de la superficie de decisión, en consecuencia, $\mathbf{w}$ determina la orientación de la superficie de decisión. De manera similar, si $\mathbf{x}$ es un punto en la superficie de decisión, entonces $y(\mathbf{x}) = 0$, luego la distancia desde el origen hasta la superficie de decisión viene dada por

$$\frac{y\left(\mathbf{x}\right)}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T\mathbf{x} + \mathbf{w}_0}{\|\mathbf{w}\|} = 0, \tag{2.34}$$

donde

$$\frac{\mathbf{w}^T\mathbf{x}}{\|\mathbf{w}\|} = -\frac{\mathbf{w}_0}{\|\mathbf{w}\|}. \tag{2.35}$$

Con base en lo anterior, se observa que el parámetro de sesgo $\mathbf{w}_0$ determina la localización de la superficie de decisión como muestra la Figura 2.2.1 para $D = 2$. Además, se observa que el valor de $y(\mathbf{x})$ da una medida de la distancia perpendicular r desde el punto $\mathbf{x}$ a la superficie de decisión. Para ver esto, considerar un punto arbitrario $\mathbf{x}$ y sea $\mathbf{x}_\perp$ su proyección ortogonal sobre la superficie de decisión, de modo que

$$\mathbf{x} = \mathbf{x}_\perp + r\frac{\mathbf{w}}{\|\mathbf{w}\|}. \tag{2.36}$$

Al multiplicar ambos lados por $\mathbf{w}^T$, sumar $\mathbf{w}_0$ y notando que $y\left(\mathbf{x}_\perp\right) = \mathbf{w}^T\mathbf{x}_\perp + \mathbf{w}_0 = 0$, se obtiene

$$y\left(\mathbf{x}\right) = \mathbf{w}^T\mathbf{x} + \mathbf{w}_0 = \mathbf{w}^T\mathbf{x}_\perp + rw^T\frac{\mathbf{w}}{\|\mathbf{w}\|} + \mathbf{w}_0 = r\left\|\mathbf{w}\right\|, \tag{2.37}$$

de esta manera, la distancia perpendicular es dada por

$$r = \frac{y\left(\mathbf{x}\right)}{\|\mathbf{w}\|}. \tag{2.38}$$

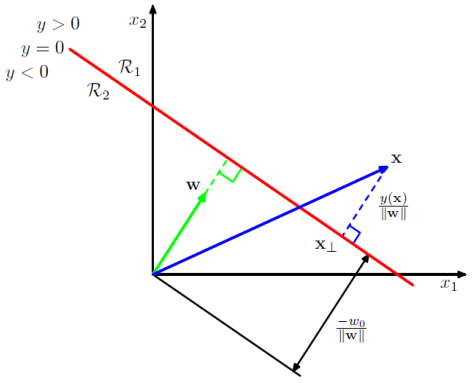


Figura 2.2.1. Ilustración geométrica de una función discriminante lineal en dos dimensiones. La superficie de decisión se muestra en rojo cuyo desplazamiento desde el origen está controlado por el parámetro w_0 [16].

Al igual que en los modelos de regresión lineal, a veces es conveniente usar una notación más compacta en la que se introduce un valor de entrada dummy $x_0 = 1$ y luego se define $\widetilde{\mathbf{w}} = (\mathbf{w}_0, \mathbf{w})$ y $\widetilde{x} = (x_0, x)$, de modo que

$$y(\mathbf{x}) = \widetilde{\mathbf{w}}^T \widetilde{x}. \tag{2.39}$$

En este caso, las superficies de decisión son hiperplanos D–dimensionales que pasan por el origen del espacio de entrada $(D+1)$-dimensional.

2.2.2. Regresión logística

Considerando el problema de la clasificación de dos clases $\mathcal{C}_1$ y $\mathcal{C}_2$ se cumple que la probabilidad a posteriori de la clase $\mathcal{C}_1$ puede escribirse como una función sigmoidal logística actuando sobre una función lineal del vector de características ϕ, tal que

$$p(\mathcal{C}_1|\phi) = y(\phi) = \sigma\left(\mathbf{w}^T\phi\right), \tag{2.40}$$

con $p(\mathcal{C}_2|\phi) = 1 - p(\mathcal{C}_1|\phi)$. En efecto, basta notar que la probabilidad a posteriori de la clase $\mathcal{C}_1$ se puede escribir como

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)\,p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)\,p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)\,p(\mathcal{C}_2)} = \frac{1}{1+\exp(-a)} = \sigma(a), \tag{2.41}$$

donde se ha definido

$$a = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)\,p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)\,p(\mathcal{C}_2)}. \tag{2.42}$$

En la terminología de estadística, este modelo se conoce como *regresión logística*, aunque debe enfatizarse que este es un modelo de clasificación en lugar de regresión.

Observación 16 *Para el caso de $K > 2$ clases, las probabilidades a posteriori se pueden escribir como*

$$\begin{aligned} p(\mathcal{C}_k|\mathbf{x}) &= \frac{p(\mathbf{x}|\mathcal{C}_k)\,p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_k)\,p(\mathcal{C}_k)} \\ &= \frac{\exp(a_k)}{\sum_j \exp(a_j)}, \end{aligned} \tag{2.43}$$

donde los a_k están definidos por

$$a_k = \ln\left(p(\mathbf{x}|\mathcal{C}_k)\,p(\mathcal{C}_k)\right). \tag{2.44}$$

Esta función es conocida como la exponencial normalizada o función softmax y se puede considerar como una generalización multiclase de la función sigmoidal logística. Notar que si $a_k \gg a_j$ para todo $j \neq k$, entonces $p(\mathcal{C}_k|\mathbf{x}) \simeq 1$ y $p(\mathcal{C}_j|\mathbf{x}) \simeq 0$.

Ahora, se utiliza la máxima verosimilitud para determinar los parámetros del modelo de la regresión logística. Para hacer esto, se hace uso de la derivada de la función sigmoidal logística, que puede expresarse de manera conveniente como

$$\frac{d\sigma}{da} = \sigma(a)\left(1 - \sigma(a)\right). \tag{2.45}$$

Para un conjunto de datos $\{\phi_n, t_n\}$, donde $t_n \in \{0,1\}$ y $\phi_n = \phi_n(x_n)$, la función de verosimilitud se puede escribir como

$$p(t|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n} \{1 - y_n\}^{1-t_n} \tag{2.46}$$

con $y_n = p(\mathcal{C}_1|\phi_n)$ y $t = (t_1, ..., t_N)^T$, pues la regresión logística supone que los datos obedecen a la distribución de Bernoulli. La función de error se puede definir tomando el logaritmo negativo de la verosimilitud, que da la función de error conocida como *entropía cruzada*

$$E(\mathbf{w}) = -\ln p(t|\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}, \tag{2.47}$$

con $y_n = \sigma(\mathbf{w}^T \phi_n)$. Tomando el gradiente de la función de error con respecto a $\mathbf{w}$ y tomando en cuenta la igualdad (2.45) se obtiene

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n) \phi_n. \tag{2.48}$$

En particular, la contribución al gradiente, desde los puntos de datos, viene dada por el "error" $y_n - t_n$ entre el valor objetivo y la predicción del modelo, multiplicado por el vector de función base ϕ_n. Por otra parte, al comparar con (2.13) se observa que toma exactamente la misma forma que el gradiente de la función suma de error cuadrado para el modelo de regresión lineal.

Observación 17 *Vale la pena señalar que la estimación por máxima verosimilitud puede exhibir problemas de sobreajuste severo para conjuntos de datos que son linealmente separables. Esto surge, porque la solución de máxima verosimilitud ocurre cuando el hiperplano correspondiente a* $\sigma = 0{,}5$ *o de forma equivalente* $\mathbf{w}^T \phi = 0$*, separa las dos clases y la magnitud de* $\mathbf{w}$ *tiende a infinito. En este caso, la pendiente de la función sigmoidal logística tiende a infinito en el espacio de características, de modo que a cada punto de entrenamiento, de cada clase* k*, se le asigna una probabilidad a posteriori* $p(\mathcal{C}_k|\mathbf{x}) = 1$*. Sin embargo, esta singularidad se puede evitar mediante la inclusión de un término de regularización a la función de error.*

2.2.3. Clasificación por mínimos cuadrados

En la sección anterior, se observó que la minimización de la función suma de error cuadrado conduce a una solución simple de forma cerrada para los valores de los parámetros. Dado esto, se aplicará el mismo formalismo a problemas de clasificación para ver si cumple la misma propiedad.

Definición 18 *Se llama esquema de codificación* $1-de-K$ *donde* $\mathbf{t}$ *es un vector de longitud* K*, de modo que si la clase correspondiente es* $\mathcal{C}_j$*, entonces todos los elementos* t_k *de* $\mathbf{t}$ *son cero, excepto el elemento* t_j *que toma el valor* 1*.*

Ejemplo 19 *Si se tiene* $K = 5$ *clases* $(\mathcal{C}_1, ..., \mathcal{C}_5)$*, entonces un patrón de la clase* $\mathcal{C}_2$ *recibe el vector objetivo*

$$\mathbf{t} = (0, 1, 0, 0, 0)^T.$$

El valor t_k *se puede interpretar como la probabilidad de que la clase sea* $\mathcal{C}_k$*.*

Una justificación para usar mínimos cuadrados en este contexto es que se aproxima a la esperanza condicional $\mathbb{E}[t|x]$ de los valores objetivo dado el vector de entrada. Para el esquema de codificación binaria, la esperanza condicional viene dada por el vector de probabilidad de la clase a posteriori. Sin embargo, estas probabilidades generalmente se aproximan bastante mal, de hecho, las aproximaciones pueden tener valores fuera del rango $(0, 1)$, debido a la flexibilidad limitada de un modelo lineal.

Cada clase $\mathcal{C}_k$ es descrita por su propio modelo lineal, tal que

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + \mathbf{w}_{k0}, \tag{2.49}$$

con $k = 1, ..., K$. Por otra parte, al igual que antes se puede agrupar en su forma vectorial como

$$y(\mathbf{x}) = \widetilde{W}^T \widetilde{x}, \tag{2.50}$$

donde $\widetilde{W}$ es una matriz cuya k^{th} columna comprende el vector $(D+1)$–dimensional $\widetilde{w} = (w_{k0}, \mathbf{w}_k^T)^T$ y $\widetilde{x}$ es el correspondiente vector de entrada $(1, \mathbf{x}^T)^T$ con una entrada dummy $x_0 = 1$. Una nueva entrada $\mathbf{x}$ es asignada a la clase donde la salida $y_k = \widetilde{w}_k^T \widetilde{x}$ es mayor.

La matriz de parámetros $\widetilde{W}$, se determina minimizando la función suma de error cuadrado al igual que en los modelos de regresión lineal. Para ver esto, se toma un conjunto de datos de entrenamiento $\{\mathbf{x}_n, t_n\}$ donde $n = 1, ..., N$ y una matriz T cuya n^{th} fila es dada por el vector t_n^T, junto con una matriz $\widetilde{X}$ cuya n^{th} fila es $\widetilde{x}_n^T$. Por consiguiente, la función suma de error cuadrado se puede escribir de la siguiente forma

$$E\left(\widetilde{W}\right) = \frac{1}{2} Tr\left\{\left(\widetilde{X}\widetilde{W} - T\right)^T \left(\widetilde{X}\widetilde{W} - T\right)\right\}. \tag{2.51}$$

Ahora, derivando con respecto a $\widetilde{W}$ e igualando a cero, se obtiene la solución para la matriz de parámetros

$$\widetilde{W} = \left(\widetilde{X}^T \widetilde{X}\right)^{-1} \widetilde{X} T. \tag{2.52}$$

Como resultado, la función discriminante es dada por

$$y(\mathbf{x}) = \widetilde{W}^T \widetilde{x} = T^T \left\{\left(\widetilde{X}^T \widetilde{X}\right)^{-1} \widetilde{X}\right\}^T \widetilde{x}. \tag{2.53}$$

Una propiedad interesante de las soluciones de mínimos cuadrados con múltiples variables objetivo es que si cada vector objetivo en un conjunto de entrenamiento satisface alguna restricción lineal

$$a^T t_n + b = 0 \tag{2.54}$$

para algunas constantes a y b, entonces la predicción del modelo para cualquier valor de $\mathbf{x}$ cumplirá la misma restricción

$$a^T y(\mathbf{x}) + b = 0. \tag{2.55}$$

Por lo tanto, si se usa un esquema de codificación $1 - de - K$ para K clases, entonces las predicciones hechas por el modelo tendrán la propiedad de que los elementos de $y(\mathbf{x})$ sumarán 1 para cualquier valor de $\mathbf{x}$. Sin embargo, esta restricción por sí sola no es suficiente para permitir que las salidas del modelo sean interpretadas como probabilidades, pues no están restringidas a mantenerse dentro del intervalo $(0, 1)$.

El enfoque de mínimos cuadrados proporciona una solución de forma cerrada para los parámetros de la función discriminante. Sin embargo, incluso con la función discriminante (la cual se usa para tomar decisiones directamente y prescindir de cualquier interpretación probabilística) sufre algunos problemas graves. Las soluciones con mínimos cuadrados carecen de robustez, de modo que a valores atípicos pueden producir resultados engañosos, como se ilustra en la Figura 2.2.3.

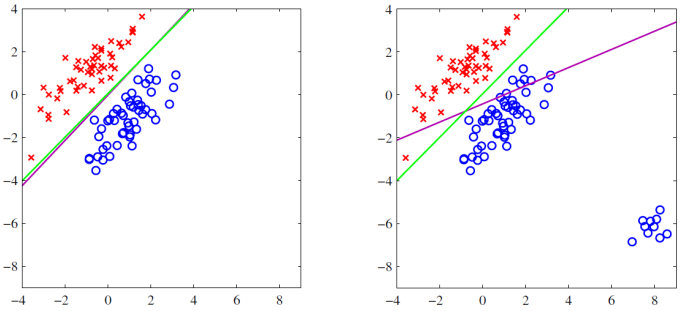


Figura 2.2.3. La gráfica izquierda muestra datos de dos clases, junto con la frontera de decisión encontrada por mínimos cuadrados (curva morada) y por el modelo de regresión logística (curva verde). El gráfico de la derecha muestra los resultados obtenidos cuando se agregan puntos de datos adicionales [16].

Al agregar puntos de datos a la derecha de la Figura 2.2.3 produce un cambio significativo en la ubicación del límite de decisión. Más aún, los problemas con mínimos cuadrados pueden ser más graves que la simple falta de robustez (cambios en los valores de entrada) como se puede observar en la Figura 2.2.3.

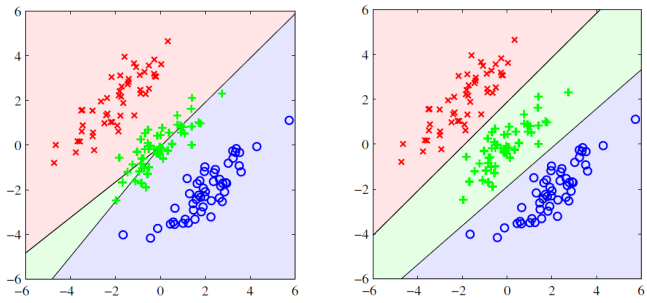


Figura 19. Conjunto de datos sintéticos de tres clases, con puntos de datos de entrenamiento indicados en rojo (×), verde (+) y azul (○) [16].

A la izquierda de la Figura 19b se utiliza el discriminante de mínimos cuadrados. Se puede observar que la región del espacio de entrada asignada a la clase verde es demasiado pequeña y, por lo tanto, la mayoría de

los puntos de esta clase están mal clasificados. Sin embargo, a la derecha se utilizó un modelo de regresión logística, el cual muestra la clasificación correcta de los datos de entrenamiento. Cabe recalcar, que bajo el supuesto de una distribución condicional Gaussina la estimación por mínimos cuadrados coincide con la de máxima verosimilitud, mientras que los vectores objetivo claramente tienen una distribución que está lejos del Gaussiana.

2.2.4. Algoritmo del perceptrón

El perceptrón de Rosenblatt es la forma más simple de una *red neuronal* usada para la clasificación de un tipo especial de patrones, los linealmente separables, es decir, patrones que se encuentran a ambos lados de un hiperplano. Básicamente, consiste en una neurona con pesos sinápticos ajustables, como se muestra la Figura **??**

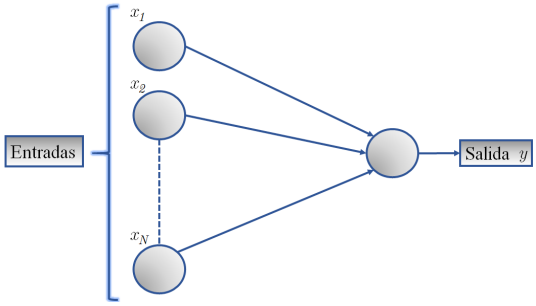


Figura 2.1.4. Representación gráfica de un perceptrón de una sola neurona.

El perceptrón es otro ejemplo de un modelo discriminante lineal que ocupa un lugar importante en la historia de los algoritmos de reconocimiento de patrones. Este corresponde a un modelo de dos clases en que el vector de entrada $\mathbf{x}$ se transforma primero usando una transformación no lineal fija para dar un vector de características $\phi(\mathbf{x})$ y luego se usa para construir un modelo lineal de la forma

$$y\left(\mathbf{x}\right) = f\left(\mathbf{w}^T \phi\left(\mathbf{x}\right)\right), \tag{2.56}$$

donde la función de activación no lineal $f(\cdot)$ viene dada por la siguiente *función de paso*

$$f\left(a\right) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}. \tag{2.57}$$

El vector $\phi\left(\mathbf{x}\right)$ incluye un componente de sesgo $\phi_0\left(\mathbf{x}\right) = 1$. En problemas de clasificación de dos clases, se ha trabajado con un esquema de codificación en el que $t \in \{0, 1\}$, que es apropiado en el contexto de modelos probabilísticos. Sin embargo, para el perceptrón, es más conveniente usar la variable objetivo $t = +1$ para la clase $\mathcal{C}_1$ y $t = -1$ cuando es de clase $\mathcal{C}_2$.

En los algoritmos de perceptrón generalmente se considera una función de error conocida como *criterio del perceptrón*. El objetivo es buscar un vector de peso $\mathbf{w}$, tal que los patrones x_n en la clase $\mathcal{C}_1$ cumplan

$\mathbf{w}^T\phi(x_n) > 0$, mientras que los patrones x_n en la clase C_2 sean $\mathbf{w}^T\phi(x_n) < 0$. Al usar un esquema de codificación $t \in \{-1, +1\}$ lo ideal sería que todos los patrones satisfagan $\mathbf{w}^T\phi(x_n)t_n > 0$. De este modo, el criterio de perceptrón asocia error cero a cualquier patrón que se clasifique correctamente, mientras que para un patrón mal clasificado x_n intenta minimizar la cantidad $-\mathbf{w}^T\phi(x_n)t_n$. Por lo tanto, el criterio del perceptrón viene dado por

$$E_p(\mathbf{w}) = -\sum_{n\epsilon\mathcal{M}} \mathbf{w}^T\phi_n t_n, \tag{2.58}$$

donde $\mathcal{M}$ denota el conjunto de todos los patrones mal clasificados. Al aplicar el algoritmo de descenso de gradiente estocástico a esta función de error se actualiza el vector de parámetros $\mathbf{w}$, tal que

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_p(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta\phi_n t_n, \tag{2.59}$$

donde η es el parámetro de *velocidad de aprendizaje* y τ es el número de iteraciones del algoritmo. El parámetro de velocidad de aprendizaje η se puede igualar a 1, sin pérdida de generalidad, ya que la función del perceptrón $y(\mathbf{x}, \mathbf{w})$ no cambia si se multiplica $\mathbf{w}$ por una constante. A medida que el vector de peso se actualiza mediante el entrenamiento, el conjunto de patrones mal clasificados irá cambiando.

El algoritmo de aprendizaje del perceptrón tiene la siguiente interpretación. Para cada patrón de entrada x_n se evalúa la función perceptrón (2.56). Si el patrón se clasifica correctamente, entonces el vector de peso permanece sin cambios, mientras que si se clasifica de forma errónea, entonces para la clase $\mathcal{C}_1$ se suma el vector $\phi(x_n)$ a la estimación del vector de peso $\mathbf{w}$, mientras que para la clase $\mathcal{C}_2$ se resta el vector $\phi(x_n)$ de $\mathbf{w}$ como se ilustra la Figura 2.2.4.

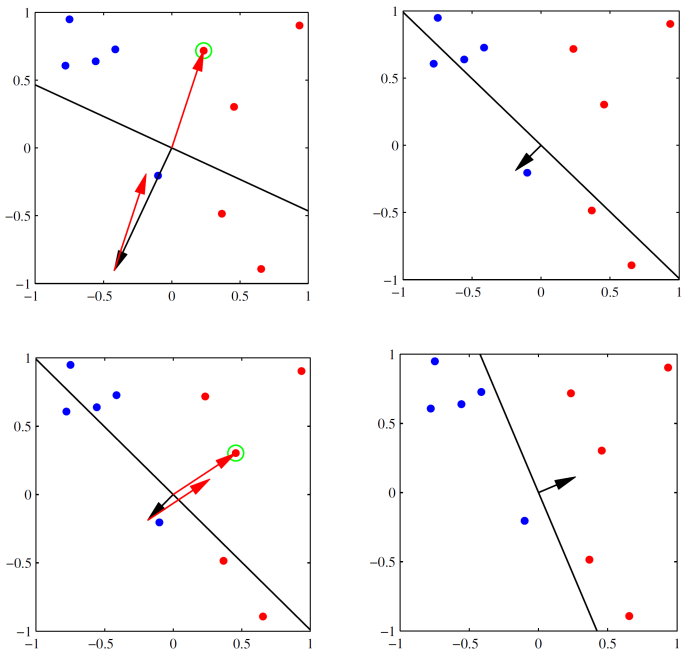


Figura 2.2.4. Ilustración de la convergencia del algoritmo de aprendizaje del perceptrón, mostrando puntos de datos de dos clases en un espacio de características bidimensional (ϕ_1, ϕ_2) [16].

La gráfica superior izquierda de la Figura 22 muestra el vector de parámetro inicial $\mathbf{w}$ con una flecha negra junto a la frontera de decisión correspondiente (línea negra). La flecha apunta hacia la región de decisión que se clasificó como perteneciente a la clase roja. El punto de datos encerrado en un círculo verde está mal clasificado y, por lo tanto, su vector de características se agrega al vector de peso actual, lo que proporciona una nueva frontera de decisión que se muestra en la gráfica superior derecha. La gráfica inferior izquierda muestra el siguiente punto mal clasificado a ser considerado, indicado por el círculo verde, y su vector de características se agrega nuevamente al vector de peso dando la frontera de decisión que se muestra en la gráfica inferior derecha, en la cual todos los puntos de datos están clasificados correctamente.

Si se considera el efecto de una única actualización en el algoritmo de aprendizaje del perceptrón, se observa que la contribución al error de un patrón mal clasificado será reducida, pues

$$-\mathbf{w}^{(\tau+1)T}\phi_n t_n = -\mathbf{w}^{(\tau)T}\phi_n t_n - (\phi_n t_n)^T \phi_n t_n < -\mathbf{w}^{(\tau)T}\phi_n t_n, \tag{2.60}$$

donde $\eta = 1$ y dado que $\|\phi_n t_n\|^2 > 0$ se cumple la desigualdad. Sin embargo, esto no implica que la contribución a la función de error de los otros patrones mal clasificados se habrá reducido. No obstante, el cambio en el vector de peso puede haber causado que algunos patrones previamente clasificados correctamente se clasifiquen de forma errónea. Por lo tanto, no se garantiza que la regla de aprendizaje del perceptrón reduzca la función de error total en cada etapa.

Rosenblatt (1962) demostró que, si los patrones usados para entrenar el perceptrón son extraídos de dos clases linealmente separables, entonces el algoritmo del perceptrón converge y toma como superficie de decisión un hiperplano entre estas dos clases. La prueba de convergencia del algoritmo es conocida como el teorema de convergencia del perceptrón.

Teorema 20 (*Convergencia del perceptrón*) *Si el conjunto de datos de entrenamiento es linealmente separable, entonces se garantiza que el algoritmo de aprendizaje del perceptrón encontrará una solución exacta en un número finito de pasos.*

Observación 21 *Incluso cuando el conjunto de datos es linealmente separable, puede haber muchas soluciones, y la que se encuentre dependerá de la inicialización de los parámetros y del orden de presentación de los puntos de datos.*

2.3. Redes neuronales

El término *"red neuronal"* tiene su origen en los intentos de encontrar representaciones matemáticas del procesamiento de información en sistemas biológicos [46], [47], [48]. De hecho, se ha utilizado de manera muy amplia para cubrir múltiples problemas. El estudio de las redes neuronales ha sido de gran interés para diversos grupos de estudio, en un principio con la idea de poder explicar el funcionamiento del cerebro y de los procesos neuronales asociados, pero posteriormente también con el objetivo de desarrollar sistemas "inteligentes"que fueran capaces de desarrollar diversas actividades.

El enfoque de este capítulo está en las redes neuronales como modelos eficientes para el reconocimiento de patrones estadísticos. Se considera la forma funcional del modelo de la red, incluida la parametrización específica de las funciones bases y luego se discute el problema de determinar los parámetros de la red dentro de un marco de máxima verosimilitud, que implica la solución de un problema de optimización no lineal. Esto requiere la evaluación de derivadas de la función de log-verosimilitud con respecto a los parámetros de la red, donde se utiliza la técnica de *retropropagación* (*"backpropagation"* en inglés) de errores para obtenerlos de manera eficiente.

2.3.1. Funciones Feed-Forward

Los modelos lineales de regresión y clasificación discutidos anteriormente se basan en combinaciones lineales de funciones bases no lineales fijas $\phi_j(\mathbf{x})$ que toman la forma

$$y\left(\mathbf{x}, \mathbf{w}\right) = f\left(\sum_{j=1}^{M} w_j \phi_j\left(\mathbf{x}\right)\right), \tag{2.61}$$

donde $f(\cdot)$ es una función de activación no lineal en el caso de la clasificación y la identidad en el caso de la regresión. El objetivo es extender este modelo haciendo que las funciones base $\phi_j\left(\mathbf{x}\right)$ dependan de los parámetros y luego permitir que estos parámetros sean ajustados, junto con los coeficientes $\{w_j\}$, durante el entrenamiento. Las redes neuronales utilizan funciones bases que siguen la misma forma que (2.61), de modo que cada función base es en sí misma una función no lineal de una combinación lineal de las entradas, donde los coeficientes en la combinación lineal son parámetros adaptativos. Esto conduce

al modelo básico de una red neuronal. Primero se construye M combinaciones lineales de las variables de entrada $x_1, ..., x_D$ de la forma

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \tag{2.62}$$

con $j = 1, ..., M$ y el exponente (1) indica que los parámetros correspondientes están en la primera *"capa"* de la red.

Definición 22 *Se definen los parámetros $w_{ji}^{(1)}$ y $w_0^{(1)}$ como pesos y sesgos, respectivamente. Las cantidades a_j se conocen como activaciones.*

Cada activación se transforma utilizando una función de activación no lineal, de modo que

$$z_j = h\left(a_j\right). \tag{2.63}$$

Estas cantidades reciben el nombre de unidades ocultas. Las funciones no lineales $h(\cdot)$ generalmente se eligen como funciones sigmoidales, por ejemplo, la función sigmoidal logística o la función tangente hiperbólica. Siguiendo (2.61), nuevamente se pueden hacer combinaciones lineales para dar activaciones de la unidad de salida

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}, \tag{2.64}$$

con $k = 1, ..., K$ donde K representa la cantidad total de salidas. Esta transformación corresponde a la *segunda capa* de la red con sus respectivos parámetros de sesgo $w_{k0}^{(2)}$. Finalmente, las activaciones de la unidad de salida se transforman utilizando una función de activación apropiada para proporcionar un conjunto de salidas de red y_k.

La elección de la función de activación está determinada por la naturaleza de los datos y la distribución asumida de las variables objetivo. Por lo tanto, para los problemas de regresión, la función de activación es la identidad, es decir $y_k = a_k$. Del mismo modo, para múltiples problemas de clasificación binaria, la activación de cada unidad de salida se puede transformar utilizando una función sigmoidal logística, tal que

$$y_k = \sigma\left(a_k\right). \tag{2.65}$$

Para problemas multiclase, se puede utilizar una función de activación softmax de la forma (2.43). Combinando lo anterior, se obtiene una salida sigmoidal

$$y_k\left(\mathbf{x}, \mathbf{w}\right) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right). \tag{2.66}$$

Definición 23 *Un modelo de red neuronal es una función no lineal de un conjunto de variables de entrada $\{x_i\}$ a un conjunto de variables de salida $\{y_k\}$ controladas por vectores de parámetros ajustables $\mathbf{w}$. La función no lineal (2.66) se representa en forma de diagrama de red en la siguiente figura.*

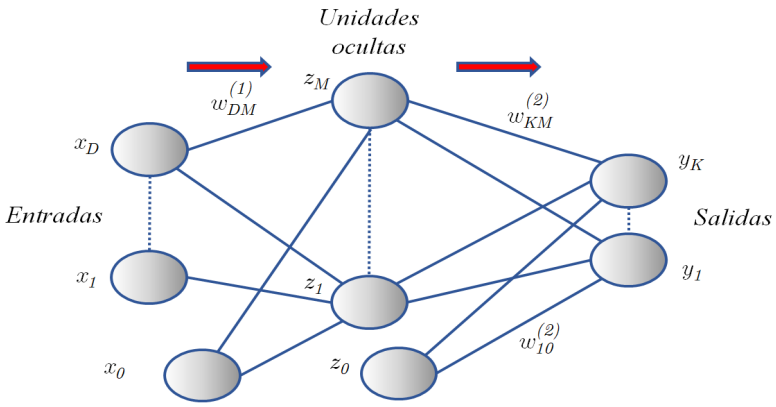


Figura 24. Diagrama de red neuronal de dos capas.

El proceso (2.66) puede interpretarse como una *propagación hacia adelante* (*"forward propagation"* en sus siglas en inglés) de información a través de la red. Al igual que antes, los parámetros de sesgo en (2.62) se pueden absorber en el conjunto de parámetros de peso definiendo una variable de entrada adicional $x_0 = 1$, de modo que (2.62) tome la forma

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{2.67}$$

y

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{ji}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right). \tag{2.68}$$

Como se observa en la Figura 24, el modelo de red neuronal comprende dos etapas de procesamiento, cada una de las cuales se asemeja al modelo de perceptrón y es por esta razón que la red neuronal también se conoce como perceptrón multicapa. Sin embargo, una diferencia clave en comparación con el perceptrón es que la red neuronal usa funciones sigmoides no lineales continuas en las unidades ocultas, mientras que el perceptrón usa una función de paso no lineal escalonada.

Si el número de unidades ocultas es menor que el número de unidades de entrada o salida, entonces las transformaciones que puede generar la red no son las transformaciones lineales más generales posibles de entradas a salidas, pues la información se pierde en la reducción de dimensionalidad en las unidades ocultas. Para evitar esto, es conveniente recurrir a conexiones de *capa de salto* o *"skip-layer"* en ingles, las cuales están asociadas a un parámetro adaptativo. Por ejemplo, en una red de dos capas estas conexiones irían directamente de entradas a salidas, como se ilustra en la Figura 26.

Definición 24 *A este tipo de arquitectura se les conoce como red neuronal feed-forward, de modo que las conexiones entre las unidades no forman un ciclo cerrado, garantizando que las salidas sean funciones deterministas de las entradas.*

Ejemplo 25 *Cada unidad (oculta o de salida) en una red feed-forward puede ser representada por*

$$z_k = h\left(\sum_j w_{kj} z_j\right), \tag{2.69}$$

donde la suma se ejecuta sobre todas las unidades que envían conexiones a la unidad k (incluyendo el parámetro de sesgo en la suma). Esto se ilustra en la Figura 2.3.1 para el caso de una red de dos capas con $k = 2$.

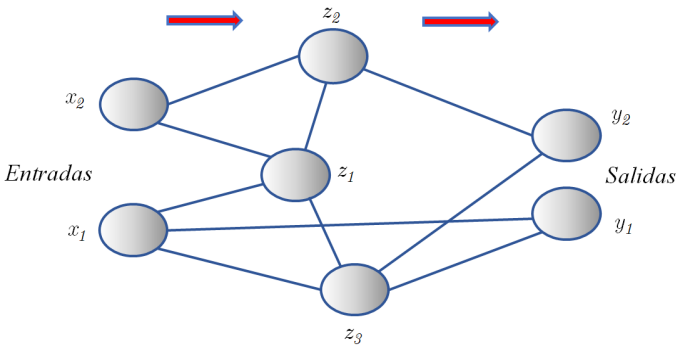


Figura 2.3.1. Red neuronal con topología feed-forward.

Teorema 26 (Teorema de aproximación universal, [49]) *Cualquier función continua sobre un conjunto compacto puede ser uniformemente aproximada (con precisión arbitraria) por una red de dos capas con salidas lineales, siempre que la red tenga un número suficientemente grande de unidades ocultas.*

Ejemplo 27 *La capacidad de una red de dos capas para modelar las funciones*

$$a)\ f(x) = x^2, \quad b)\ f(x) = \sin(x), \quad c)\ f(x) = |x| \quad y \quad d)\ f(x) = H(x),$$

donde $H(x)$ es la función de paso Heaviside, se ilustra en la siguiente figura.

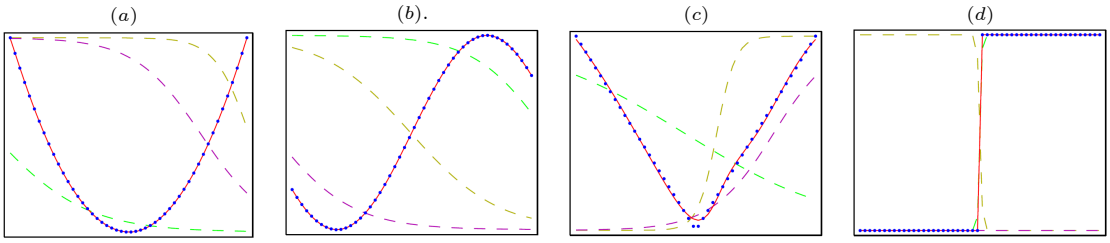


Figura 2.63. Ilustración de la capacidad de un perceptrón multicapa para aproximar funciones [16].

En la Figura 2.63 se consideran $N = 50$ puntos de datos, representados por los puntos azules. Estos se utilizan para entrenar una red de dos capas que tiene 3 *unidades ocultas con funciones de activación* tanh *y unidades de salida lineal. Las funciones de red resultantes se muestran mediante las curvas rojas, y las salidas de las tres unidades ocultas se muestran mediante las tres curvas discontinuas las cuales trabajan en colaboración para aproximar la función deseada.*

2.3.2. Entrenamiento de la red

Hasta ahora, se ha visto las redes neuronales como una clase general de funciones no lineales paramétricas desde un vector x de variables de entrada a un vector y de variables de salida. Un enfoque simple para el problema de determinar los parámetros de la red es minimizar la función suma de error de cuadrado

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N} \|y(x_n, w) - t_n\|^2 \tag{2.70}$$

donde el conjunto de entrenamiento comprende los vectores de entrada $\{x_n\}$, con $n = 1, ..., N$, junto con sus correspondientes vectores objetivo $\{t_n\}$.

Sin embargo, se puede proporcionar una visión más general del entrenamiento de la red dando una interpretación probabilística a las salidas, las cuales tienes diversas ventajas en la práctica. Se comienza discutiendo los problemas de regresión, considerando una única variable objetivo t que puede tomar cualquier valor real y suponiendo que sigue una distribución Gaussiana con una media dependiente de $\mathbf{x}$, dada por la salida de la red neuronal, de modo que

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right), \tag{2.71}$$

donde β es la precisión (varianza inversa) del ruido Gaussiano. Para la distribución dada por (2.71), es suficiente tomar la función de activación de la unidad de salida como la identidad, pues dicha red puede aproximar cualquier función continúa de $\mathbf{x}$ a y. Dado un conjunto de datos de N observaciones independientes e idénticamente distribuidas $\mathbf{X} = \{x_1, ..., x_N\}$, junto con los valores objetivo correspondientes $\mathbf{t} = \{t_1, ..., t_N\}$, se puede construir la función de verosimilitud

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|x_n, w, \beta).$$

Tomando el logaritmo negativo, se obtiene la función de error

$$\frac{\beta}{2}\sum_{n=1}^{N} \{y(x_n, w) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi), \tag{2.72}$$

que se puede usar para conocer los parámetros $\mathbf{w}$ y β.

Observación 28 *En la literatura de redes neuronales, es habitual considerar la minimización de la función de error en lugar de la maximización de la log-verosimilitud, ya que maximizar la función de verosimilitud es equivalente a minimizar la función suma de error cuadrado.*

Por lo tanto, por Observación 28, para determinar $\mathbf{w}$ se minimiza

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2. \tag{2.73}$$

El valor de $\mathbf{w}$ encontrado al minimizar $E(\mathbf{w})$ se denota $\mathbf{w}_{ML}$, donde ML significa *maximum likelihood,* que corresponde a la máxima verosimilitud. En la práctica, la no linealidad de la función de red $y(x_n, w)$ hace que el error $E(\mathbf{w})$ no sea *convexo*, esto implica que no hay garantia de encontrar los máximos locales de la verosimilitud (de forma equivalente los mínimos locales de la función de error). Por otra parte, β se puede encontrar minimizando la log-verosimilitud negativa para dar

$$-\nabla \log p(t|\mathbf{x}, \mathbf{w}, \beta) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2 - \frac{N}{2\beta},$$

igualando a cero y despejando β se obtiene

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}_{ML}) - t_n\}^2. \tag{2.74}$$

Si se tienen múltiples variables objetivo, y se asume que son independientes de $\mathbf{x}$ y $\mathbf{w}$, entonces la distribución condicional de los valores objetivo está dada por

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(\boldsymbol{t}|\mathbf{y}(\mathbf{x}, \mathbf{w}), \beta^{-1}I\right). \tag{2.75}$$

Siguiendo el mismo argumento que para una única variable objetivo, se tiene que la precisión del ruido viene dada por

$$\frac{1}{\beta_{ML}} = \frac{1}{NK}\sum_{n=1}^{N}\|\boldsymbol{y}(x_n, \mathbf{w}_{ML}) - \boldsymbol{t}_n\|^2, \tag{2.76}$$

donde K es el número de variables objetivo.

En el caso de regresión, la red tiene una función de activación de salida que es la identidad, de modo que $y_k = a_k$. La correspondiente función suma de error cuadrado tiene la propiedad que

$$\frac{\partial E}{\partial a_k} = y_k - t_k, \tag{2.77}$$

la cual se utiliza en la técnica de backpropagation que se discute más adelante.

Ahora, si se considera el caso de clasificación binaria donde se tiene una única variable objetivo t, tal que $t = 1$ denota la clase $\mathcal{C}_1$ y $t = 0$ denota la clase $\mathcal{C}_2$, cuya red tiene una única salida con una función de activación sigmoidal logística dada por

$$y(\mathbf{x}, \mathbf{w}) = \sigma(a) = \frac{1}{1 + \exp(-a)}. \tag{2.78}$$

Entonces, se puede interpretar $y(\mathbf{x}, \mathbf{w})$ como la probabilidad condicional $p(\mathcal{C}_1|\mathbf{x})$, con $p(\mathcal{C}_2|\mathbf{x})$ dada por $1 - y(\mathbf{x}, \mathbf{w})$. Por consiguiente, la distribución condicional de los objetivos corresponde a una distribución de Bernoulli de modo que

$$p(t|\mathbf{x}, \mathbf{w}) = y^t(\mathbf{x}, \mathbf{w})\{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}. \tag{2.79}$$

Si se considera un conjunto de entrenamiento de observaciones independientes, entonces la función de error dada por la log-verosimilitud negativa, es la función de error de entropía cruzada

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln (1 - y_n)\}. \tag{2.80}$$

Simard et al. (2003), demostraron que el uso de la función de error de entropía cruzada en lugar de la suma de cuadrados para un problema de clasificación conduce a un entrenamiento más rápido, así como a una mejor generalización.

En el caso de K clasificaciones binarias se puede usar una red con K salidas, las cuales están asociadas a una etiqueta de clase binaria $t_k \in \{0, 1\}$, donde $k = 1, ..., K$, cada una de las cuales puede tener una función de activación sigmoidal logística. Si se asume que las etiquetas de clase son independientes, dado el vector de entrada, entonces la distribución condicional de los objetivos es

$$p(\boldsymbol{t}|\mathbf{x}, \mathbf{w}) = \prod_{k=1}^{K} y_k^{t_k} \{1 - y_k\}^{1-t_k}. \tag{2.81}$$

Tomando el logaritmo negativo de la función de verosimilitud correspondiente, se obtiene la siguiente función de error

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} \{t_{nk} \ln y_{nk} + (1 - t_{nk}) \ln (1 - y_{nk})\} \tag{2.82}$$

donde $y_{nk} = y_k(x_n, w)$. Nuevamente, la derivada de la función de error con respecto a la activación para una unidad de salida particular toma la forma

$$\begin{aligned} \frac{\partial E}{\partial a_k} &= -\frac{1}{e^{-a_k} + 1} \left(t_{nk} + e^{-a_k} t_{nk} - 1\right) \\ &= \frac{1}{e^{-a_k} + 1} - t_{nk} \\ &= y_{nk} - t_{nk}, \end{aligned}$$

al igual que en el caso de regresión.

Observación 29 *Es interesante comparar la solución de la red neuronal basada en este problema con el enfoque correspondiente a un modelo de clasificación lineal. Suponiendo que se usa una red neuronal estándar de dos capas (ver Figura 24), se tiene que los parámetros de peso en la primera capa de la red se comparten entre las distintas salidas, mientras que en el modelo lineal cada problema de clasificación se resuelve de forma independiente. En consecuencia, se puede considerar que la primera capa de la red realiza una extracción de características no lineales, de modo que las características entre las diferentes salidas pueden conducir a una mejor generalización.*

Finalmente, se considera el problema de clasificación multiclase estándar en el que cada entrada se asigna a una de las K clases mutuamente excluyentes. Las variables objetivo binarias $t_k \in \{0, 1\}$ tienen

un esquema de codificación $1-de-K$ que indica la clase, y las salidas de la red se interpretan como $y_k(\mathbf{x},\mathbf{w}) = p(t_k = 1|\mathbf{x})$. Así la distribución condicional es dada por

$$p\left(\boldsymbol{t}|\mathbf{x},\mathbf{w}\right) = \prod_{k=1}^{K} y_k^{t_{nk}},$$

lo que lleva a la función de error

$$E\left(\mathbf{w}\right) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_k\left(x_n, w\right), \tag{2.83}$$

cuya función de activación de la unidad de salida viene dada por la función softmax

$$y_k(\mathbf{x},\mathbf{w}) = \frac{\exp\left(a_k\left(\mathbf{x},\mathbf{w}\right)\right)}{\sum_j \exp\left(a_j\left(\mathbf{x},\mathbf{w}\right)\right)}, \tag{2.84}$$

que satisface $0 \leq y_k \leq 1$, $\sum_k y_k = 1$ y al igual que antes, la derivada de la función de error con respecto a la activación de una unidad de salida sofmax toma la forma (2.77).

Observación 30 *Si una constante se suma a todas las activaciones de la unidad $a_k(\mathbf{x},\mathbf{w})$, los $y_k(\mathbf{x},\mathbf{w})$ no sufren cambios , lo que hace que la función de error sea constante en algunas direcciones en el espacio de los pesos. Esta degeneración se elimina si se agrega un término de regularización apropiado (Sección regularizadores) a la función de error.*

Resumen 31 *Existe una elección natural de la función de activación de la unidad de salida y de la función de error según el tipo de problema que se esté resolviendo.*

1) Problemas de regresión

Se puede utilizar la función suma de error cuadrado con salidas lineales.

2) Problemas de clasificaciones binarias

Se puede utilizar la función de error de entropía cruzada, con una función de activación sigmoide logísticas para las salidas.

3) Problemas de clasificación multiclase

Se puede utilizar la función de error de entropía cruzada multiclase, con una función de activación softmax para las salidas.

Para problemas de clasificación que involucran dos clases, se puede utilizar una única salida sigmoidal logística o alternativamente, una red con dos salidas que tengan una función de activación de salida softmax.

Optimización de parámetros

Ahora se trata de encontrar un vector de peso $\mathbf{w}$ que minimice la función $E(\mathbf{w})$. Una interpretación geométrica de la función de error, vista como una superficie sobre el espacio peso, se ilustra en la Figura 2.3.2, donde el punto w_A es un mínimo local, w_B es el mínimo global y para cualquier punto w_C el gradiente local de la superficie de error viene dado por el vector ∇E.

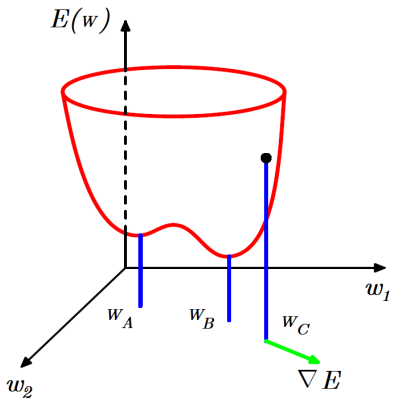


Figura 2.3.2. Función error $E(\mathbf{w})$ como una superficie sobre el espacio de los pesos [16].

Si se realiza un pequeño paso en el espacio peso de $\mathbf{w}$ a $\mathbf{w} + \delta w$, entonces el cambio en la función de error es

$$\delta E \simeq \delta w^T \nabla E(\mathbf{w}),$$

donde el vector $\nabla E(\mathbf{w})$ apunta en la dirección de mayor tasa de crecimiento de la función. Debido a que el error $E(\mathbf{w})$ es una función continúa suave, su valor más pequeño ocurrirá en un punto en el espacio de los pesos donde el gradiente de la función de error desaparece, de modo que

$$\nabla E\left(\mathbf{w}\right) = 0, \tag{2.85}$$

ya que, de lo contrario, se podría dar un paso en la dirección de $-\nabla E(\mathbf{w})$ y así reducir aún más el error.

Definición 32 *Los puntos donde el gradiente desaparece se denominan puntos estacionarios (o singularidades) y pueden clasificarse en mínimos, máximos y puntos de silla.*

El objetivo es encontrar un vector $\mathbf{w}$, tal que $E(\mathbf{w})$ tome su valor más pequeño, sin embargo, la función de error normalmente tiene una dependencia altamente no lineal de los parámetros de peso y sesgo, por lo que habrá muchos puntos en el espacio peso en los que el gradiente desaparece (o es numéricamente muy pequeño). De hecho, para cualquier punto $\mathbf{w}$ que sea un mínimo local, habrá otros puntos en el espacio de los pesos que serán mínimos. Para fines prácticos, en problemas que se utilizan redes neuronales puede que no sea necesario encontrar el mínimo global (y en general, no se sabrá si se ha encontrado el mínimo global), pero puede ser necesario comparar varios mínimos locales para encontrar una solución suficientemente buena.

Debido a que no es evidente encontrar una solución analítica a la ecuación $\nabla E(\mathbf{w}) = 0$, se utilizan procedimientos numéricos iterativos. La mayoría de las técnicas implican elegir algún valor inicial $\mathbf{w}^{(0)}$ para el vector de peso y luego moverse a través del espacio de peso en una sucesión de pasos de la forma

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta w^{(\tau)} \tag{2.86}$$

donde τ etiqueta el paso de cada iteración. Los diferentes algoritmos implican diferentes opciones para la actualización del vector de peso $\Delta w^{(\tau)}$. Muchos algoritmos utilizan información del gradiente y, por lo tanto, requieren que después de cada actualización se evalúe el valor de $\nabla E(\mathbf{w})$ en el nuevo vector de peso $\mathbf{w}^{(\tau+1)}$.

Optimización del descenso de gradiente

El enfoque más simple para usar la información del gradiente es elegir la actualización de los pesos

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta w^{(\tau)},$$

para incluir un pequeño paso en la dirección del gradiente negativo, de modo que

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E\left(\mathbf{w}^{(\tau)}\right), \tag{2.87}$$

donde el parámetro $\eta > 0$ se conoce como *tasa de aprendizaje*. Después de cada actualización de este tipo, el gradiente se vuelve a evaluar por el nuevo vector de peso y se repite el proceso.

En cada paso, el vector de peso se mueve en dirección de la mayor tasa de disminución de la función de error, por lo que este enfoque se conoce como descenso de gradiente. Dentro de esta técnica se pueden distinguir diferentes métodos, como

1. Descenso de gradiente

Esta técnica utiliza todo el conjunto de datos a la vez, en consecuencia, los parámetros se actualizan una vez que hayan pasado todos los ejemplos del entrenamiento $\{x_n, t_n\}$ a través de la red, es decir, si un conjunto de datos contiene T ejemplos de entrenamientos, los parámetros de la red neuronal se actualizan una vez.

Ventajas

a) Menos oscilaciones.

b) Puede beneficiarse de la vectorización que aumenta la velocidad de procesamiento de todas las muestras.

c) Estimación precisa del vector gradiente (es decir, la derivada de la función de error con respecto al vector de peso $\mathbf{w}$), garantizando así, en condiciones simples, la convergencia del método de descenso más pronunciado a un mínimo local.

Desventajas

a) No permite el aprendizaje on-line debido a que la actualización del gradiente se produce sobre el conjunto entero.

b) Bajo rendimiento y velocidad en el caso de ser utilizado en tiempo real.

c) Desde una perspectiva práctica, el aprendizaje por lotes es bastante exigente en términos de requisitos de almacenamiento.

2. Descenso de gradiente estocástico o secuencial

Es una versión on-line del descenso del gradiente que ha resultado útil en la práctica para entrenar redes neuronales en grandes conjuntos de datos (Le Cun et al., 1989). Las funciones de error basadas en la máxima verosimilitud para un conjunto de observaciones independientes comprenden una suma de términos, uno para cada punto de dato

$$E\left(\mathbf{w}\right) = \sum_{n=1}^{N} E_n\left(\mathbf{w}\right). \tag{2.88}$$

El descenso de gradiente on-line, también conocido como *descenso de gradiente secuencial* o *descenso de gradiente estocástico*, actualiza el vector de peso en función de un punto a la vez o por lotes, de modo que

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n\left(\mathbf{w}^{(\tau)}\right). \tag{2.89}$$

Esta actualización se repite recorriendo los datos en secuencia o seleccionando puntos al azar con reemplazo.

Ventajas

a) Una ventaja de los métodos on-line en comparación con los métodos por lotes es que los primeros manejan la redundancia (es decir, la muestra de entrenamiento contiene varias copias del mismo ejemplo) en los datos de manera mucho más eficiente.

b) Otra propiedad del descenso de gradiente estocástico es la posibilidad de escapar de los mínimos locales, ya que un punto estacionario con respecto a la función de error generalmente no será un punto estacionario para cada punto de datos individualmente.

c) Requiere mucho menos almacenamiento que el aprendizaje por lotes.

d) Capacidad para rastrear pequeños cambios en los datos de entrenamiento, particularmente cuando el entorno responsable de generar los datos no es estacionario.

Desventajas

a) Debido a las actualizaciones frecuentes los pasos dados hacia los mínimos son muy ruidosos. Esto puede conducir al descenso de gradiente en otras direcciones.

b) Puede llevar más tiempo lograr la convergencia a los mínimos de la función de error.

c) Pierde la ventaja de operaciones vectorizadas, ya que trata con solo un ejemplo a la vez.

Observación 33 *Si un conjunto de entrenamiento contiene T ejemplos, los parámetros se actualizan $M \leq T$ veces.*

Resumen 34 *A pesar de las desventajas, el aprendizaje on-line es muy popular para resolver problemas de clasificación de patrones por dos importantes razones prácticas:*

- *Sencillo de implementar.*
- *Proporciona soluciones eficaces a problemas de clasificación de patrones difíciles y de gran escala.*

Sin embargo, es recomendable combinar ambos métodos para aprovechar al máximo sus ventajas y mitigar sus carencias.

2.3.3. Error de backpropagation

El objetivo en esta sección es encontrar un método eficiente para evaluar el gradiente de una función de error $E(\mathbf{w})$ para una red neuronal feed-forward. Esto se puede lograr utilizando un esquema donde la información se envía alternativamente hacia adelante y hacia atrás a través de la red, lo que se conoce como backpropagation. El término backpropagation también se utiliza para describir el entrenamiento de un perceptrón multicapa mediante el descenso de gradiente. La mayoría de los algoritmos de entrenamiento involucran un procedimiento iterativo para minimizar una función de error, con ajustes a los pesos que se realizan en una secuencia de pasos. En cada uno de estos pasos se puede distinguir dos etapas distintas.

1. En la primera etapa, se deben evaluar las derivadas de la función de error con respecto a los pesos, cuya contribución consiste en proporcionar un método computacionalmente eficiente para evaluar tales derivadas. Debido a que es en esta etapa donde los errores se propagan hacia atrás a través de la red, se utilizará el término backpropagation específicamente para describir la evaluación de derivadas.

2. En la segunda etapa, las derivadas se utilizan para calcular los ajustes que se realizarán a los pesos. La técnica más simple de este tipo, y la que originalmente consideraron Rumelhart et al. (1986), involucra el descenso del gradiente.

Esto se puede utilizar en muchos otros tipos de red y no solo al perceptrón multicapa. También se puede aplicar a funciones de error distintas a la suma de cuadrados, y a la evaluación de otras derivadas, por ejemplo, la matriz Hessiana.

Evaluación de las derivadas de la función de error

Ahora se considera un algoritmo de backpropagation para una red general con topología feed-forward arbitraria, funciones de activación diferenciables, no lineales y una amplia clase de funciones de error. Las fórmulas resultantes serán ilustradas usando una estructura de red simple que tiene sola una capa de unidades ocultas sigmoidales junto con una función suma de error cuadrado.

Muchas funciones de error de interés práctico, por ejemplo, las definidas por la máxima verosimilitud para un conjunto de datos independientes e idénticamente distribuidos ($i.i.d$), comprenden una suma de términos, uno para cada punto de datos en el conjunto de entrenamiento, de modo que

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}). \tag{2.90}$$

Aquí se considera el problema de evaluar el gradiente $\nabla E_n(\mathbf{w})$ para uno de esos términos en la función de error. Esto puede ser usado directamente por la optimización secuencial o los resultados pueden ser acumulados sobre el conjunto de entrenamiento (métodos por lotes).

Primero, se considera un modelo lineal simple donde las salidas y_k son combinaciones lineales de las variables de entrada x_i, de modo que

$$y_k = \sum_i w_{ki} x_i, \tag{2.91}$$

junto con una función de error que, para un patrón de entrada particular n, toma la forma

$$E_n = \frac{1}{2} \sum_k \left(y_{nk} - t_{nk}\right)^2, \tag{2.92}$$

donde $y_{nk} = y_k\left(x_n, w\right)$. El gradiente de esta función de error con respecto a un peso w_{ji} viene dado por

$$\frac{\partial E_n}{\partial w_{ji}} = \left(y_{nj} - t_{nj}\right) x_{ni}. \tag{2.93}$$

En la Sección 2.2.2, se observó cómo surge una fórmula similar con la función de activación sigmoidal logística junto con la función de error de entropía cruzada, y de manera similar, para la función de activación softmax junto con su correspondiente función de error de entropía cruzada. Esto se puede extender a un entorno más complejo de redes multicapas feed-forward. En general, en una red feed-forward, cada unidad calcula una suma ponderada de sus entradas de la forma

$$a_j = \sum_i w_{ji} z_i, \tag{2.94}$$

donde z_i es la activación de una unidad y w_{ji} es el peso asociado con esa conexión. La suma en (2.94) se transforma mediante una función de activación no lineal $h(\cdot)$ para dar la activación z_j de la unidad j en la forma

$$z_j = h\left(a_j\right). \tag{2.95}$$

Observación 35 *Notar que, una o más de las variables z_i en (2.94) podría ser una entrada y de manera similar, la unidad j en (2.95) podría ser una salida.*

Definición 36 *Se dice proceso de propagación hacia adelante o "forward propagation" en inglés, cuando el flujo de información va hacia adelante a través de la red, es decir, para cada patrón del conjunto de entrenamiento se calcula las activaciones de todas las unidades ocultas y de salida de la red mediante la aplicación sucesiva de (2.94) y (2.95).*

La función de error E_n depende del peso w_{ji} solo a través de la entrada a_j correspondiente a la unidad j. Por lo tanto, se puede aplicar la regla de la cadena para derivadas parciales, tal que

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}. \tag{2.96}$$

Se introduce la notación

$$\delta_j = \frac{\partial E_n}{\partial a_j}, \tag{2.97}$$

donde los δ_j a menudo se denominan errores. Usando (2.94), se puede escribir

$$\frac{\partial a_j}{\partial w_{ji}} = z_i. \tag{2.98}$$

Al sustituir (2.97) y (2.98) en (2.96), se obtiene

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i, \tag{2.99}$$

de este modo, para evaluar las derivadas solo se necesita calcular el valor de los δ_j para cada unidad oculta y de salida en la red, y luego aplicar (2.99). Para las unidades de salida se tiene que

$$\delta_k = \frac{\partial E_n}{\partial a_k} = y_k - t_k \tag{2.100}$$

siempre que se utilice la identidad como función de activación de la unidad de salida. Para evaluar los δ's de las unidades ocultas, nuevamente se hace uso de la regla de la cadena para derivadas parciales

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}, \tag{2.101}$$

donde la suma recorre todas las k unidades a las que la unidad j envía conexiones. La disposición de unidades y pesos se ilustra en la Figura 2.3.3.

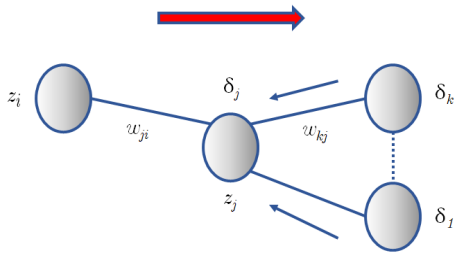


Figura 2.3.3. Ilustración del cálculo de δ_j para la unidad oculta j por backpropagation. Las flechas indican la dirección del flujo de información durante la propagación.

Observación 37 *Al escribir (2.101) se hace uso del hecho de que las modificaciones en a_j dan lugar a cambios en la función de error solo a través de variaciones en las variables a_k.*

Haciendo uso de (2.100) y notando que

$$\begin{aligned}\frac{\partial a_k}{\partial a_j} &= \frac{\partial}{\partial a_j}\sum_j w_{kj} z_j \\ &= \frac{\partial}{\partial a_j}\sum_j w_{kj} h\left(a_j\right) \\ &= w_{kj} h'\left(a_j\right),\end{aligned} \tag{2.102}$$

se obtiene la siguiente fórmula de backpropagation al sustituir estos resultados en (2.101)

$$\delta_j = h'\left(a_j\right)\sum_k w_{kj}\delta_k, \tag{2.103}$$

donde el valor de δ para una unidad oculta particular puede obtenerse propagando los δ's hacia atrás desde unidades más arriba en la red, como se ilustra en la Figura 36.

Observación 38 *Notar que la sumatoria en (2.103) se toma sobre el primer índice en w_{kj} (correspondiente a la propagación hacia atrás de información a través de la red), mientras que en la ecuación de propagación hacia adelante (2.69) se toma sobre el segundo índice j.*

Debido a que ya se conocen los valores de los $\delta^{'}s$ para las unidades de salida, aplicando recursivamente (2.103) se puede evaluar los $\delta^{'}s$ para todas las unidades ocultas en una red feed-forward, independiente de su topología. Por tanto, el procedimiento de backpropagation se puede resumir como sigue.

Error de Backpropagation

1. Aplicar un vector de entrada x_n a la red y propagar hacia adelante a través de la red usando (2.94) y (2.95) para encontrar las activaciones de todas las unidades ocultas y de salida.
2. Evaluar δ_k para todas las unidades de salida usando (2.100).
3. Retropropagar los δ's usando (2.103) y así obtener los δ_j para cada unidad oculta de la red.
4. Utilizar (2.99) para evaluar las derivadas requeridas.

Para los métodos por lotes, la derivada del error total E se puede obtener repitiendo los pasos anteriores para cada patrón en el conjunto de entrenamiento y luego sumar todos los patrones, es decir

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}}. \tag{2.104}$$

Ejemplo 39 *Considerando una red de dos capas de la forma ilustrada en la Figura 2.61, junto con un error de suma de cuadrados, en la cual las unidades de salida tienen funciones de activación lineal, de modo que $y_k = a_k$, mientras que las unidades ocultas tienen funciones de activación tangente hiperbólica dadas por*

$$h\left(a\right) = \tanh\left(a\right), \tag{2.105}$$

donde

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}, \tag{2.106}$$

cuya derivada puede ser expresada como

$$h'(a) = 1 - h(a)^2. \tag{2.107}$$

También se considera la función estándar suma de error cuadrado, de modo que para el patrón, n, el error viene dado por

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2, \tag{2.108}$$

donde y_k es la activación de la unidad de salida k y t_k el correspondiente objetivo para un particular patrón de entrada x_n.

Para cada patrón en el conjunto de entrenamiento, se realiza primero una forward propagation usando

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{2.109}$$

$$z_j = \tanh(a_j) \tag{2.110}$$

$$y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j. \tag{2.111}$$

Se pueden calcular los δ's para cada unidad de salida usando

$$\delta_k = y_k - t_k. \tag{2.112}$$

Entonces, se hace backpropagation para obtener los δ's para las unidades ocultas

$$\delta_j = \left(1 - z_j^2\right)\sum_{k=1}^{K} w_{kj}\delta_k. \tag{2.113}$$

Finalmente, las derivadas de la primera y segunda capa respecto a los pesos son dadas por

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i \quad y \quad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j. \tag{2.114}$$

2.4. Máquina de vectores de soporte

Las *máquinas de vectores soporte* (*SVM*, del inglés *"Support Vector Machines"*) tienen su origen en los trabajos sobre la teoría del aprendizaje estadístico y fueron introducidas en los años 90 por Vapnik y sus colaboradores (Boser et al., 1992, Cortes & Vapnik, 1995). Aunque originariamente las SVMs fueron pensadas para resolver problemas de clasificación binaria, actualmente se utilizan para resolver otros tipos de problemas (por ejemplo, regresión o agrupamiento). También son diversos los campos en los que han

sido utilizadas con éxito, tales como visión artificial, reconocimiento de caracteres, categorización de texto e hipertexto, clasificación de proteínas, procesamiento de lenguaje natural, análisis de series temporales, clasificación de fraudes, etc. Las máquinas de vectores de soporte realizan una clasificación no lineal utilizando lo que se llama el *"truco del kernel"*, mapeando implícitamente sus entradas en espacios de características de alta dimensión.

2.4.1. Kernel y representación dual

En los capítulos anteriores se han considerado modelos paramétricos lineales para regresión y clasificación donde la forma del mapeo $y(\mathbf{x}, \mathbf{w})$ desde la entrada $\mathbf{x}$ hasta la salida y se rige por un vector $\mathbf{w}$ de parámetros adaptativos. Durante la etapa de aprendizaje, se utiliza un conjunto de datos de entrenamiento para obtener una estimación puntual del vector de parámetros o para determinar una distribución posterior sobre este vector. A continuación, se descartan los datos de entrenamiento y las predicciones de nuevas entradas se basan exclusivamente en el vector $\mathbf{w}$ de parámetros aprendidos. Muchos modelos paramétricos lineales se pueden convertir en una representación dual equivalente donde las predicciones también se basan en combinaciones lineales de una *función kernel* evaluada en los puntos de datos de entrenamiento. Para los modelos que se basan en un mapeo espacial de características no lineales fijas $\phi(\mathbf{x})$, la función kernel viene dada por la siguiente relación

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') \tag{2.115}$$

De esta definición, se observa que el kernel es una función simétrica de modo que $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$. El ejemplo más simple de una función kernel se obtiene al considerar el mapeo identidad para el espacio de características (2.115), donde $\phi(\mathbf{x}) = \mathbf{x}$ y en cuyo caso $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$ recibe el nombre de kernel lineal. El concepto de un kernel formulado como un producto interno en un espacio de características permite construir extensiones interesantes de muchos algoritmos conocidos haciendo uso del *truco del kernel*.

Representación dual

Muchos modelos lineales de regresión y clasificación pueden reformularse en términos de una representación dual donde la función kernel surge naturalmente. Se considera un modelo de regresión lineal cuyos parámetros se determinan minimizando una función suma de error cuadrado regularizada, de la forma

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^T\phi(x_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}, \tag{2.116}$$

donde $\lambda \geq 0$. Si se iguala a cero el gradiente de $J(\mathbf{w})$ con respecto a $\mathbf{w}$, se observa que la solución para $\mathbf{w}$ toma la forma de una combinación lineal de los vectores $\phi(x_n)$ con coeficientes que son funciones de $\mathbf{w}$, tal que

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\left\{\mathbf{w}^T\phi(x_n) - t_n\right\}\phi(x_n) = \sum_{n=1}^{N} a_n\phi(x_n) = \Phi^T a \tag{2.117}$$

donde Φ es la matriz de diseño, cuya enésima fila está dada por $\phi(x_n)^T$ con $a = (a_1, ..., a_N)^T$ y se define

$$a_n = -\frac{1}{\lambda}\left\{\mathbf{w}^T\phi(x_n) - t_n\right\}. \tag{2.118}$$

En lugar de trabajar con el vector de parámetros $\mathbf{w}$, ahora se puede reformular el algoritmo de mínimos cuadrados en términos del vector de parámetros a, dando lugar a una *representación dual*. Si se sustituye $\mathbf{w} = \Phi^T a$ en $J(\mathbf{w})$, se obtiene

$$J(a) = \frac{1}{2}a^T\Phi\Phi^T\Phi\Phi^T a - a^T\Phi\Phi^T t + \frac{1}{2}t^T t + \frac{\lambda}{2}a^T\Phi\Phi^T a, \tag{2.119}$$

donde $t = (t_1, ..., t_N)^T$. Ahora se define la *matriz de Gram,* $K = \Phi\Phi^T$, la cual es simétrica de orden $N \times N$ cuyos elementos son dados por

$$K_{nm} = \phi^T(x_n)\,\phi(x_m) = k(x_n, x_m), \tag{2.120}$$

lo que introduce a la *función kernel* $k(\mathbf{x}, \mathbf{x}')$ definida en (2.115). En términos de la matriz de Gram, la función suma de error cuadrado se puede escribir como

$$J(a) = \frac{1}{2}a^T KKa - a^T Kt + \frac{1}{2}t^T t + \frac{\lambda}{2}a^T Ka. \tag{2.121}$$

Estableciendo el gradiente de $J(a)$ con respecto a la variable a e igualando a cero, se obtiene la siguiente solución

$$a = (K + \lambda I_N)^{-1} t. \tag{2.122}$$

Si se sustituye esto nuevamente en el modelo de regresión lineal, se obtiene la siguiente predicción para una nueva entrada $\mathbf{x}$

$$y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = a^T\Phi\phi(\mathbf{x}) = \mathrm{k}(\mathbf{x})^T(K + \lambda I_N)^{-1} t, \tag{2.123}$$

donde se a definido el vector $\mathrm{k}(\mathbf{x})$ con elementos $k_n(\mathbf{x}) = k(x_n, \mathbf{x})$. De esta manera, la formulación dual permite que la solución del problema de mínimos cuadrados se exprese completamente en términos de la función kernel $k(\mathbf{x}, \mathbf{x}')$.

Observación 40 *Se conoce como formulación dual porque la solución para a puede expresarse como combinación lineal de los elementos de* $\phi(\mathbf{x})$*, recuperando la formulación original en términos del vector de parámetros* $\mathbf{w}$*. La ventaja de la formulación dual, es que se expresa completamente en términos de la función kernel* $k(\mathbf{x}, \mathbf{x}')$*. Por tanto, se puede trabajar directamente en términos del kernel y evitar la introducción explícita del vector de características* $\phi(\mathbf{x})$*, lo que permite utilizar implícitamente espacios de características de dimensionalidad alta, incluso infinita.*

2.4.2. Clasificación del margen máximo

Se comienza la discusión de las SVM volviendo al problema de clasificación de dos clases utilizando el modelo lineal

$$y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) + b \tag{2.124}$$

donde $\phi(\mathbf{x})$ denota una transformación espacio-características fija, con parámetro de sesgo b explicito. El conjunto de datos de entrenamiento comprende N vectores de entrada $x_1, ..., x_N$, con los valores objetivo correspondientes $t_1, ..., t_N$ donde $t_n \in \{-1, 1\}$, y los nuevos puntos de datos $\mathbf{x}$ se clasifican de acuerdo con el signo de $y(\mathbf{x})$.

Se supone, por el momento, que el conjunto de datos de entrenamiento es linealmente separable en el espacio de características, de modo que, por definición, existe al menos una elección de los parámetros $\mathbf{w}$ y b tal que una función de la forma (2.124) satisface $y(x_n) > 0$ para puntos que tienen $t_n = +1$ e $y(x_n) < 0$ para puntos que tienen $t_n = -1$, de modo que $t_n y(x_n) > 0$ para todos los puntos de datos de entrenamiento que están correctamente clasificados.

Existen muchas de estas soluciones que separan las clases exactamente. En la Sección 2.2.4, se describió el algoritmo del perceptrón que garantiza encontrar una solución en un número finito de pasos. La solución que se encuentre, sin embargo, dependerá de los valores iniciales (arbitrarios) elegidos para $\mathbf{w}$ y b, así como del orden en el que se presentan los puntos de datos. Si hay varias soluciones que clasifican exactamente el conjunto de datos de entrenamiento, entonces se debería intentar encontrar la que arroje el *error de generalización* más pequeño. La máquina de vectores de soporte aborda este problema mediante el concepto de *margen*.

Definición 41 *El margen se define como la distancia perpendicular entre la frontera de decisión y los puntos de datos más cercanos.*

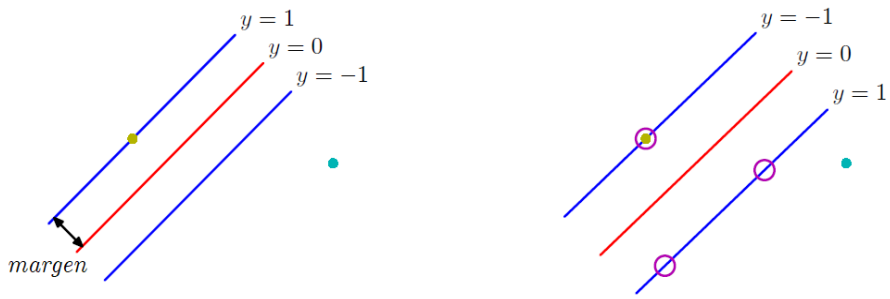


Figura 2.4.2. Ilustración del margen y la frontera de decisión.

Maximizar el margen conduce a una elección particular de la frontera de decisión, como se muestra a la derecha de la Figura 2.4.2. La ubicación de este límite está determinada por un subconjunto de puntos de datos, conocidos como *vectores de soporte*, que están indicados por los círculos.

Como se observó en la Figura 2.2.1, la distancia perpendicular de un punto $\mathbf{x}$ a un hiperplano definido por $y(\mathbf{x}) = 0$, donde $y(\mathbf{x})$ toma la forma (2.124), está dada por $|y(\mathbf{x})|/\|\mathbf{w}\|$. Sin embargo, el interés radica en las soluciones donde los puntos de datos están clasificados correctamente, es decir, cuando $t_n y(x_n) > 0$ para todo n. Así, la distancia de un punto x_n a la superficie de decisión está dada por

$$\frac{t_n y_n}{\|\mathbf{w}\|} = \frac{t_n \left(\mathbf{w}^T \phi_n (x_n) + b\right)}{\|\mathbf{w}\|}. \tag{2.125}$$

El margen viene dado por la distancia perpendicular al punto más cercano x_n del conjunto de datos, cuyo enfoque es optimizar los parámetros $\mathbf{w}$ y b con el fin de maximizar esta distancia. Por lo tanto, la solución de margen máximo se encuentra resolviendo

$$\arg \max_{w,b} \left\{ \frac{1}{\|w\|} \min_n \left[t_n \left(w^T \phi (x_n) + b \right) \right] \right\}. \tag{2.126}$$

La solución directa de este problema de optimización no es trivial, es por esto que se trata de transformar en un problema equivalente mucho más fácil de resolver. Para hacer esto, notar que si se reescala $\mathbf{w} \to \kappa w$ y $b \to \kappa b$, entonces la distancia desde cualquier punto x_n a la superficie de decisión, dada por $t_n|y(x)|/\|\mathbf{w}\|$, no cambia. Entonces, se puede utilizar esto de forma conveniente para establecer la siguiente igualdad

$$t_n\left(\phi_n\left(x_n\right)\mathbf{w}^T + b\right) = 1, \tag{2.127}$$

para el punto más cercano a la superficie. En este caso, todos los puntos de datos cumplen la restricción

$$t_n\left(\phi_n\left(x_n\right)\mathbf{w}^T + b\right) \geq 1, \tag{2.128}$$

con $n = 1, ..., N$. Esto se conoce como la representación canónica del hiperplano de decisión.

Definición 42 *En el caso donde se mantiene la igualdad en (2.128), se dice que las restricciones están activas, mientras que para el resto se dice que están inactivas.*

Observación 43 *Por definición, siempre habrá al menos una restricción activa, pues siempre habrá un punto más cercano a la superficie y una vez que el margen se haya maximizado habrá al menos dos restricciones activas.*

El problema de optimización requiere simplemente que se maximice $\|\mathbf{w}\|^{-1}$, o de forma equivalente minimizar $\|\mathbf{w}\|$. Sin embargo, para simplificar ciertos cálculos es conveniente minimizar $\|\mathbf{w}\|^2$. Por consiguiente, se tiene que resolver el siguiente problema de optimización

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \tag{2.129}$$

sujeto a la restricción dada por (2.128). El factor 1/2 en (2.129) se agrega por conveniencia para más adelante. Este es un ejemplo de un problema de *programación cuadrática* donde se trata de minimizar una función cuadrática sujeta a un conjunto de restricciones de desigualdad lineal.

Para resolver este problema de optimización restringida, se introducen *multiplicadores de Lagrange* $a_n \geq 0$, con un multiplicador para cada una de las restricciones en (2.128), dando la función

$$L\left(\mathbf{w}, b, a\right) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n\left\{t_n\left(\mathbf{w}^T\phi\left(x_n\right) + b\right) - 1\right\}, \tag{2.130}$$

donde $a = (a_1, ..., a_N)^T \in \mathbb{R}^N$.

Observación 44 *Se agrega el signo menos delante del multiplicador de Lagrange, porque estamos minimizando con respecto a* $\mathbf{w}$ *y* b*, y maximizando con respecto al vector* a*.*

Estableciendo las derivadas de $L(\mathbf{w}, b, a)$ con respecto a $\mathbf{w}$ y b e igualando a cero, se obtiene las siguientes dos condiciones

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi\left(x_n\right) \tag{2.131}$$

$$0 = \sum_{n=1}^{N} a_n t_n. \tag{2.132}$$

Respecto a las condiciones (2.131) y (2.132), se obtiene la *representación dual* del problema del margen máximo en el que se maximiza

$$\widetilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m), \tag{2.133}$$

con respecto a a sujeto a las restricciones

$$a_n \geq 0, \tag{2.134}$$

$$\sum_{n=1}^{N} a_n t_n = 0. \tag{2.135}$$

Aquí la función kernel se define por $k(x, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$. Esto toma la forma de un problema de programación cuadrática en el que se optimiza una función cuadrática de a sujeta a un conjunto de restricciones de desigualdad.

Para clasificar nuevos puntos de datos usando el modelo entrenado, se evalúa el signo de $y(\mathbf{x})$ definido por (2.124). Esto se puede expresar en términos de los parámetros $\{a_n\}$ y la función kernel sustituyendo $\mathbf{w}$, para dar

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, x_n) + b. \tag{2.136}$$

Una optimización restringida de esta forma satisface las condiciones de Karush-Kuhn-Tucker ($\mathcal{KKT}$), que en este caso requieren que las siguientes tres propiedades se cumplan

$$a_n \geq 0 \tag{2.137}$$

$$t_n y(x_n) - 1 \geq 0 \tag{2.138}$$

$$a_n \{t_n y(x_n) - 1\} = 0. \tag{2.139}$$

Por lo tanto, para cada punto de datos se tiene que $a_n = 0$ o bien $t_n y(x_n) = 1$. Los puntos donde $a_n = 0$, no estarán representados en la suma de (2.136) y, por lo tanto, no desempeñan ningún papel en la realización de nuevas predicciones.

Definición 45 *Los puntos de datos donde $a_n \neq 0$ en (2.136) se denominan vectores de soporte.*

Debido a que los vectores de soporte satisfacen $t_n y(x_n) = 1$, estos se encontrarán en los hiperplanos de margen máximo en el espacio de características como se ilustra en la Figura 41.

Observación 46 *La propiedad recién mencionada es fundamental para la aplicabilidad práctica de las máquinas de vectores de soporte, ya que una vez que se entrena el modelo se puede descartar una proporción significativa de puntos de datos y solo se pueden conservar los vectores de soporte.*

Habiendo resuelto el problema de programación cuadrática y encontrado un valor para a, se puede determinar el valor del parámetro b notando que cualquier vector de soporte x_n satisface $t_n y(x_n) = 1$. Usando (2.136), se obtiene

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(x_n, x_m) + b \right) = 1 \tag{2.140}$$

donde $\mathcal{S}$ denota el conjunto de índices de los vectores de soporte. Se obtiene una solución numéricamente más estable al multiplicar primero por t_n, haciendo uso de $t_n^2 = 1$, luego se promedia estas ecuaciones sobre todos los vectores de soporte y por último se despeja b para obtener

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k\left(x_n, x_m\right) \right) \tag{2.141}$$

donde $N_{\mathcal{S}}$ es el número total de vectores de soporte.

Para una comparación posterior con modelos alternativos, se puede expresar el clasificador de margen máximo en términos de la minimización de una función de error, con un regularizador cuadrático simple, de la forma

$$\sum_{n=1}^{N} E_\infty \left(t_n y\left(x_n\right) - 1 \right) + \lambda \left\| \mathbf{w} \right\|^2 \tag{2.142}$$

donde $E_\infty\left(z\right)$ es una función que es cero si $z \geq 0$ e ∞ en caso contrario.

La Figura 44 muestra un ejemplo de la clasificación resultante de entrenar una máquina de vectores de soporte, con un conjunto de datos sintéticos simple, utilizando un Kernel Gaussiano de forma

$$k\left(\mathbf{x}, \mathbf{x}'\right) = \exp\left(\frac{-\left\| \mathbf{x} - \mathbf{x}' \right\|^2}{2\sigma^2} \right). \tag{2.143}$$

Aunque el conjunto de datos no es linealmente separable en el espacio de datos bidimensional $\mathbf{x}$, es linealmente separable en un espacio de características no lineal definido implícitamente por la función kernel no lineal. Por lo tanto, los puntos de datos de entrenamiento están perfectamente separados en el espacio de datos original.

Observación 47 *El hiperplano de margen máximo se define por la ubicación de los vectores de soporte. Otros puntos de datos se pueden mover libremente (siempre que permanezcan fuera de la región del margen) sin cambiar el límite de decisión, por lo que la solución será independiente de dichos puntos de datos.*

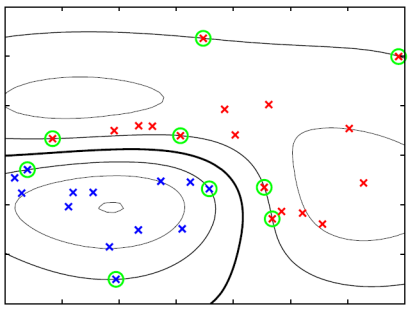


Figura 44. Datos sintéticos de dos clases (en dos dimensiones) que muestran contornos de $y(\mathbf{x})$ obtenidos de un kernel Gaussiano. También se muestra la frontera de decisión, los límites del margen y los vectores de soporte [16].

2.4.3. Distribución de clases superpuestas

Hasta ahora, se ha asumido que los puntos de datos de entrenamiento son linealmente separables en el espacio de características $\phi(\mathbf{x})$. La SVM resultante dará una separación exacta de los datos de entrenamiento en el espacio de entrada original $\mathbf{x}$, aunque la frontera de decisión correspondiente no sea lineal. Sin embargo, en la práctica las distribuciones pueden superponerse, en cuyo caso la separación exacta de los datos de entrenamiento puede conducir a una mala generalización. Por consiguiente, se necesita una forma de modificar la máquina de vectores de soporte para permitir que algunos de los puntos de entrenamiento se clasifiquen incorrectamente. De (2.142) se observa que, en el caso de clases separables, se usa implícitamente una función de error que daba un error infinito si un punto de datos estaba mal clasificado y un error cero si estaba clasificado correctamente, luego se optimizan los parámetros del modelo para maximizar el margen. Ahora, se modifica este enfoque para permitir que los puntos de datos estén en el "lado equivocado" del límite del margen, pero con una penalización que aumenta con la distancia desde ese límite. Para el problema de optimización posterior, es conveniente hacer de esta penalización una función lineal de esta distancia. Para hacer esto, se introduce una *variable de holgura*, $\xi_n \geq 0$ con $n = 1, ..., N$, para cada punto de datos de entrenamiento (Bennett, 1992; Cortes y Vapnik, 1995). Estas se definen por $\xi_n = 0$ para puntos de datos que están en o dentro del límite del margen correcto y $\xi_n = |t_n - y(x_n)|$ para otros puntos. Por lo tanto, si un punto de datos está en la frontera de decisión $y(x_n) = 0$, su variable de holgura será $\xi_n = 1$, y para los puntos en los cuales su variable de holgura es $\xi_n > 1$ serán clasificados erróneamente. Las restricciones de clasificación exacta (2.128) son reemplazadas por

$$t_n y(x_n) \geq 1 - \xi_n \qquad \text{y} \qquad \xi_n \geq 0, \qquad n = 1, ..., N. \tag{2.144}$$

Los puntos de datos para los cuales $\xi_n = 0$, están clasificados correctamente y se encuentran sobre el margen o en el lado correcto del margen. Ahora, los puntos donde $0 < \xi_n \leq 1$, se encuentran dentro del margen, pero en el lado correcto de la frontera de decisión. Por último, aquellos puntos de datos para los cuales $\xi_n > 1$, se ubican en el lado equivocado de la frontera de decisión y, por lo tanto, están mal clasificados como se puede apreciar en la Figura **??**.

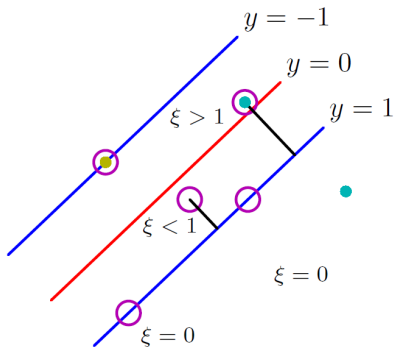


Figura **??**. Ilustración de las variables de holgura $\xi_n \geq 0$. Los puntos de datos con círculo alrededor de ellos son vectores de soporte [16].

Si bien, las variables de holgura permiten distribuciones de clases superpuestas, esto aún es sensible a valores atípicos, pues la penalización por clasificación errónea aumenta linealmente con ξ.

Ahora, el objetivo es maximizar el margen mientras se penaliza suavemente los puntos que se encuentran en el lado equivocado del límite del margen. Por lo tanto, se minimiza

$$C\sum_{n=1}^{N}\xi_n+\frac{1}{2}\left\|\mathbf{w}\right\|^2 \tag{2.145}$$

donde el parámetro $C>0$ controla el equilibrio entre la variable de holgura y el margen. Debido a que cualquier punto mal clasificado cumple que $\xi_n>1$, entonces se tiene que $\sum_n\xi_n$ es el límite superior de puntos mal clasificados. Por lo tanto, el parámetro C es análogo a (el inverso) un coeficiente de regularización, pues controla el equilibrio entre minimizar los errores de entrenamiento y la complejidad del modelo.

Ahora se minimiza (2.145) sujeto a las restricciones (2.144). El Lagrangiano correspondiente está dado por

$$L\left(\mathbf{w},b,a\right)=\frac{1}{2}\left\|\mathbf{w}\right\|^2+C\sum_{n=1}^{N}\xi_n-\sum_{n=1}^{N}a_n\left\{t_ny\left(x_n\right)-1+\xi_n\right\}-\sum_{n=1}^{N}\mu_n\xi_n, \tag{2.146}$$

donde $\{a_n\geq 0\}$ y $\{\mu_n\geq 0\}$ son multiplicadores de Lagrange. De esta manera, el conjunto de las condiciones correspondiente de $\mathcal{KKT}$ viene dado por

$$a_n\geq 0 \tag{2.147}$$
$$t_ny\left(x_n\right)-1+\xi_n\geq 0 \tag{2.148}$$
$$a_n\left\{t_ny\left(x_n\right)-1+\xi_n\right\}=0 \tag{2.149}$$
$$\mu_n\geq 0 \tag{2.150}$$
$$\xi_n\geq 0 \tag{2.151}$$
$$\mu_n\xi_n=0 \tag{2.152}$$

donde $n=1,...,N$.

Al optimizar $L\left(\mathbf{w},b,a\right)$ respecto a $\mathbf{w}$, b y $\{\xi_n\}$ da como resultado

$$\frac{\partial L}{\partial\mathbf{w}}=0\implies\mathbf{w}=\sum_{n=1}^{N}a_nt_n\phi\left(x_n\right) \tag{2.153}$$

$$\frac{\partial L}{\partial b}=0\implies\sum_{n=1}^{N}a_nt_n=0 \tag{2.154}$$

$$\frac{\partial L}{\partial\xi_n}=0\implies a_n=C-\mu_n. \tag{2.155}$$

Usando estos resultados para eliminar $\mathbf{w}$, b y $\{\xi_n\}$ del Lagrangiano, se obtiene su forma dual

$$\widetilde{L}\left(a\right)=\sum_{n=1}^{N}a_n-\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}a_na_mt_nt_mk\left(x_n,x_m\right), \tag{2.156}$$

la cual es idéntica al caso separable, excepto por las restricciones. Para ver esto, se observa que $a_n \geq 0$ al ser multiplicador de Lagrange. Además, de (2.155) y (2.150) se obtiene que $a_n \leq C$. Por lo tanto, se minimiza (2.156) con respecto a las variables duales $\{a_n\}$ sujeto a

$$0 \leq a_n \leq C \tag{2.157}$$

$$\sum_{n=1}^{N} a_n t_n = 0, \tag{2.158}$$

para $n = 1, ..., N$, donde (2.157) se conoce como *restricciones de caja*. Esto nuevamente representa un problema de programación cuadrática. Si se sustituye (2.153) en (2.124), las predicciones para nuevos puntos de datos se hacen nuevamente usando (2.136).

Ahora se puede interpretar la solución resultante. Como antes, un subconjunto de puntos de datos puede tener $a_n = 0$, en cuyo caso no contribuyen al modelo predictivo (2.136). Los puntos de datos restantes constituyen los vectores de soporte. Estos cumplen que $a_n > 0$ y, por tanto, de (2.149) deben satisfacer

$$t_n y\left(x_n\right) = 1 - \xi_n. \tag{2.159}$$

Si $a_n < C$, entonces (2.155) implica que $\mu_n > 0$, que a partir de (2.152) requiere que $\xi_n = 0$ y, por lo tanto, tales puntos se encuentran en el margen. Los puntos con $a_n = C$ pueden estar dentro del margen y pueden clasificarse correctamente si $\xi_n \leq 1$ o clasificarse incorrectamente si $\xi_n > 1$. Para determinar el parámetro b de (2.124) se observa que aquellos vectores de soporte que cumplen $0 < a_n < C$, deben tener $\xi_n = 0$, de esta manera se tiene que $t_n y(x_n) = 1$, es decir

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k\left(x, x_n\right) + b \right) = 1. \tag{2.160}$$

Nuevamente, una solución numéricamente estable se obtiene de la siguiente manera (promediando)

$$b = \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k\left(x_n, x_m\right) \right), \tag{2.161}$$

donde $\mathcal{M}$ denota el conjunto de índices de puntos de datos que tienen $0 < a_n < C$.

Aunque las predicciones para nuevas entradas se realizan utilizando solo los vectores de soporte, la fase de entrenamiento (es decir, la determinación de los parámetros a y b) hace uso de todo el conjunto de datos, por lo que es importante tener algoritmos eficientes para resolver el problema de programación cuadrática. Uno de los enfoques más populares para entrenar máquinas de vectores de soporte se llama *optimización mínima secuencial*, o *SMO* (Platt, 1999). Introduce el concepto de fragmentación y considera solo dos multiplicadores de Lagrange a la vez. En este caso, el subproblema se puede resolver analíticamente, evitando así la programación cuadrática por completo.

Las funciones del kernel corresponden a productos internos en espacios de características que pueden tener una dimensionalidad alta, incluso infinita. Al trabajar directamente en términos de la función kernel, sin introducir el espacio de características explícitamente, podría parecer que las máquinas de vectores de soporte de alguna manera logran evitar el problema de la dimensionalidad. Sin embargo, este no es el caso porque existen restricciones entre los valores de las características que restringen la dimensionalidad efectiva del espacio de características. Para ver esto, considerar el siguiente ejemplo.

Ejemplo 48 *Sea $k(x,z)$ un kernel polinomial simple de segundo orden que se puede expandir en términos de sus componentes de la siguiente forma*

$$\begin{aligned} k(x,z) &= \left(1+\mathbf{x}^T z\right)^2 \\ &= \left(1+x_1z_1+x_2z_2\right)^2 \\ &= x_1^2z_1^2+2x_1z_1x_2z_2+2x_1z_1+x_2^2z_2^2+2x_2z_2+1 \\ &= \left(1,\sqrt{2}x_1,\sqrt{2}x_2,x_1^2,\sqrt{2}x_1x_2,x_2^2\right)\left(1,\sqrt{2}z_1,\sqrt{2}z_2,z_1^2,\sqrt{2}z_1z_2,z_2^2\right)^T \\ &= \phi(x)^T\phi(z). \end{aligned} \tag{2.162}$$

Se observa que esta función kernel representa un producto interno en un espacio de características que tiene seis dimensiones, donde el mapeo del espacio de entrada al espacio de características se describe mediante la función vectorial $\phi(\mathbf{x})$. Sin embargo, los coeficientes que ponderan estas diferentes características están limitados a tener formas específicas. Por lo tanto, cualquier conjunto de puntos en el espacio bidimensional original $\mathbf{x}$ estaría limitado a estar exactamente en una variedad no lineal bidimensional incrustada en el espacio de características de seis dimensiones.

Capítulo 3

TÉCNICAS PARA BALANCEO DE DATOS

El problema de las distribuciones de datos no balanceados entre clases, ha recibido una atención considerable en diferentes disciplinas. En el contexto de problemas de clasificación, se dice que un conjunto de datos no está balanceado si una de las clases (mayoritaria) está sensiblemente más representada que el resto de clases. Esta problemática puede conducir, en términos de clasificación, a aprendizajes sesgados en perjuicio de la clase minoritaria, que usualmente, contiene los casos de mayor interés. En este capítulo se abordan diferentes técnicas de balanceo de clases para manejar este tipo de conjuntos.

3.1. Submuestreo aleatorio

El submuestreo aleatorio es una de las estrategias más simples para manejar los conjuntos datos desequilibrados. En la Figura 50, la clase mayoritaria (clase 1) está representada por los puntos de datos azules y negros. Aquí los puntos azules corresponden a las muestras eliminadas, las cuales se seleccionan al azar hasta que los datos estén equilibrados. La eliminación, obviamente, reduce la cantidad de datos en el almacenamiento y, en consecuencia, mejora el tiempo de ejecución. Sin embargo, este método puede provocar pérdida de información valiosa.

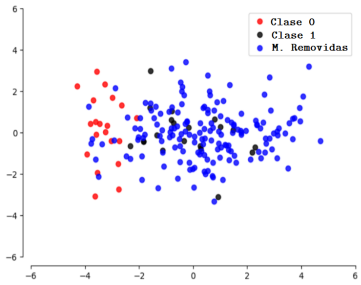


Figure 3.1. Representación gráfica de la técnica de submuestreo aleatorio.

Observación 49 *En el submuestreo aleatorio (potencialmente) se descartan grandes cantidades de datos. Esto puede ser muy problemático, ya que la eliminación de tales instancias puede hacer que el límite de decisión, de algunos modelos, no logre generalizar bien y como resultado se puede obtener un desempeño pobre al momento de clasificar.*

3.2. NearMiss

Near-miss es una técnica que ayuda a equilibrar la distribución de clases eliminando los ejemplos de las clases mayoritarias. Cuando las instancias de dos clases diferentes están muy cerca una de la otra, se eliminan los datos de la clase mayoritaria para aumentar el espacio entre las clases. El algoritmo hace este proceso teniendo en cuenta la distribución de los datos, de esta manera ayuda al proceso de clasificación. Existen tres métodos para eliminar los elementos de la clase mayoritaria:

NearMiss-1

Selecciona las muestras de la clase mayoritaria donde el promedio de la distancia, a los k-vecinos más cercanos, es la más pequeña. En la siguiente imagen se puede observar que se usan 3-vecinos más cercanos para calcular la distancia promedio en dos muestras específicas de la clase mayoritaria. Por lo tanto, en este caso se selecciona el punto vinculado por la línea de trazos verdes, ya que la distancia promedio (0,96) es menor.

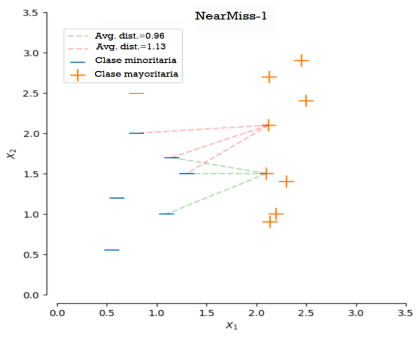


Figure **??**. Submuestreo bajo NearMiss-1.

NearMiss- 2

Selecciona muestras de la clase mayoritaria donde la distancia promedio, a los k-vecinos más lejanos, es la más pequeña. Siguiendo el mismo enfoque que en el ejemplo anterior, se selecciona la muestra vinculada a la línea punteada verde, ya que su distancia (1,66) a los 3-vecinos más lejanos es la más pequeña.

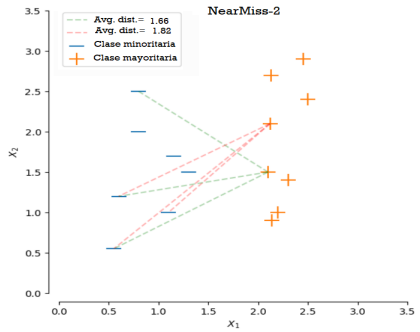


Figure **??**. Submuestreo bajo NearMiss-2.

NearMiss-3

Se puede dividir en dos pasos. Primero, se utilizan k-vecinos más cercanos para preseleccionar las muestras de la clase mayoritaria que, en este caso, corresponden a las muestras resaltadas en verde de la siguiente Figura. Luego, se selecciona la muestra con la mayor distancia promedio a los k-vecinos más cercanos.

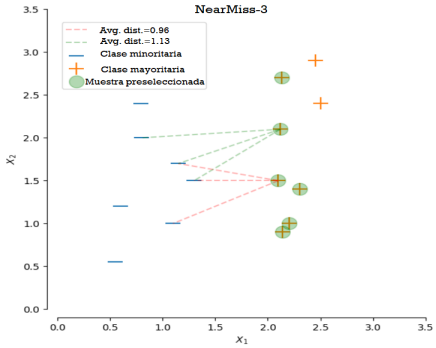


Figure **??**. Submuestreo bajo NearMiss-3.

3.3. SMOTE

La técnica de *sobremuestreo de minorías sintéticas*, SMOTE en sus siglas en inglés, se utiliza con mucha frecuencia para balancear datos desequilibrados. Esto se debe a su sencillez en el diseño del procedimiento, así como su robustez para ser aplicada en diferentes tipos de problemas. Desde que fue publicado por Nitesh V. Chawla et all. en 2002, SMOTE ha demostrado ser exitoso en una variedad de aplicaciones

de muchos dominios diferentes, inspirando distintos enfoques para contrarrestar el desequilibrio de clases. Sin embargo, una desventaja general de este enfoque es que los ejemplos sintéticos se crean sin considerar la clase mayoritaria, lo que posiblemente resulte en ejemplos ambiguos si existe una fuerte superposición. Por otra parte, al crear datos sintéticos se pueden provocar cambios en la ditribución de los datos.

Este método se hace tomando cada muestra de la clase minoritaria e creando ejemplos sintéticos a lo largo de los segmentos de línea que unen a los vecinos más cercanos de las k clases minoritarias. Dependiendo de la cantidad de sobremuestreo requerido, los vecinos de los k-vecinos más cercanos se eligen al azar. Como se verá en breve, se utiliza un paquete de Python llamado *imbalanced-learn* que ofrece una serie de técnicas de remuestreo que comúnmente son utilizadas en conjuntos de datos que muestran un fuerte desequilibrio de clases. Esta implementación utiliza actualmente 5-vecinos más cercanos. Por ejemplo, si la cantidad de sobremuestreo necesaria es del 200 %, solo dos vecinos de los cinco vecinos más cercanos se eligen y se genera una muestra en la dirección de cada uno. Las muestras sintéticas se generan tomando la diferencia entre el vector de características x_i en consideración y su vecino más cercano x_{zi}, luego se multiplica esta diferencia por un número aleatorio λ en el intervalo $[0, 1]$ y se agrega al vector de características en consideración, como se puede observar en la Figura 3.3. Esto provoca la selección de un punto aleatorio a lo largo del segmento de línea entre dos características específicas. La nueva instancia se forma con la ecuación

$$x_{new} = x_i + \lambda(x_{zi} - x_i). \tag{3.1}$$

De esta manera, los ejemplos sintéticos hacen que el clasificador cree regiones de decisión más grandes y menos específicas, como se muestra con las líneas discontinuas la Figura 3.3.

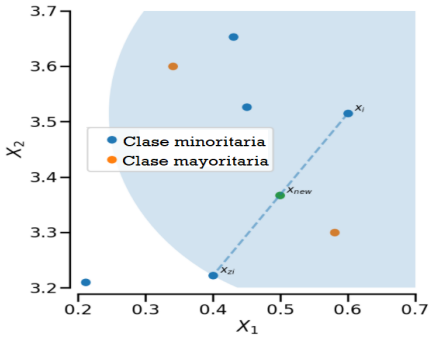


Figura 3.3. Representación gráfica de la creación de datos sintéticos mediante la técnica SMOTE.

De esta manera, el SMOTE fuerza a la región de decisión de la clase minoritaria a generalizarse y así evitar el sobreajuste.

Observación 50 *La distancia que se aplica entre los vecinos más cercanos es la distancia Euclideana. Existen extensiones que generan datos sintéticos siguiendo el enfoque de SMOTE, pero utilizando otras distancias. Por ejemplo, Lida Abdi et al.* 2015*, propusieron una extensión que se inspiraba en la distancia de Mahalanobis (MDO). Esta distancia, ayuda a preservar la estructura de la covarianza de las instancias*

de las clases minoritarias generando de manera inteligente muestras sintéticas a lo largo de los contornos de probabilidad y, en consecuencia, las nuevas instancias de clases minoritarias se modelan mejor para los algoritmos de aprendizaje. Además, MDO puede reducir el riesgo de superposición entre diferentes regiones, lo que se considera un desafío serio en problemas de varias clases. De esta manera, se puede ir trabajando SMOTE para optimizar distintos tipos de algoritmos.

Capítulo 4

DETECCIÓN DE TRANSACCIONES FRAUDULENTAS

Debido a la creciente digitalización y el predominio de aplicaciones móviles, la cantidad de transacciones con tarjetas de créditos han aumentado significativamente llevando a las personas a usar este método para realizar transacciones en línea, comprar y pagar facturas. Junto con el creciente número de usuarios, los casos de fraude han aumentado a nivel mundial causando pérdidas de miles de millones de dólares, como lo refleja un informe realizado por The Nilson Report (2016), el cual proporcionó una investigación exhaustiva sobre la situación del fraude con tarjetas de crédito en todo el mundo cuyas pérdidas financieras totales alcanzaron los $\$21,84$ mil millones en el 2015, aumentando a $\$24,71$ mil millones en el 2016 y superando los $27 mil millones en 2017. Lo que es aún peor, es que las pérdidas globales seguirán aumentando año tras año y posiblemente superen los $\$50,67$ mil millones en el año 2030.

El fraude se puede clasificar como un acto ilegal realizado por una o varias personas físicas o jurídicas y se caracteriza principalmente por la utilización del engaño para obtener algún beneficio en perjuicio de otra persona o institución. El fraude con tarjeta de crédito implica el uso no autorizado de la información de la persona con el propósito de cargar compras en la cuenta de la víctima o extraer sus fondos. Este tipo de fraude está considerado como una forma de robo de identidad y para concretar este tipo de delito se utilizan diferentes métodos, entre ellos, el robo de la tarjeta, producir tarjetas falsas, clonar el sitio original, o bien borrar o modificar la banda magnética presente en la tarjeta de crédito. La detección de fraudes mediante el uso de técnicas de aprendizaje automático es una forma eficaz de abordar este problema. En los últimos años, se ha observado un alto rendimiento en la detección de actividades fraudulentas entre un gran volumen de transacciones. A pesar de que esto plantea desafíos en la creación de modelos de clasificación debido al desequilibrio en los datos, cada vez más entidades financieras están adoptando estas técnicas para fortalecer sus medidas de seguridad.

El objetivo de este capítulo es hacer una comparación del rendimiento de diferentes modelos (redes neuronales, SVM y regresión logística) para la clasificación de transacciones fraudulentas con tarjetas de crédito. Debido a que los datos presentes son altamente desbalanceados, como es común en estos tipos de

problemas, se trabaja con técnicas de submuestreo y sobremuestreo para mejorar el desempeño de cada modelo.

4.1. Preprocesamiento y comprensión de los datos

El preprocesamiento de datos es una etapa fundamental en el proceso de extracción del conocimiento, cuyo objetivo principal es obtener un conjunto de datos final que sea de calidad y útil para la fase de extracción del conocimiento. Esta etapa se encarga de la limpieza de los datos, su integración, transformación y reducción para la siguiente fase de modelización. En el mundo real los datos frecuentemente no están limpios, faltan valores claves, existen datos inconsistentes (incluyendo discrepancias) o suelen mostrar ruido lo que provoca graves errores al momento de plantear un modelo. Es aquí donde el *preprocesamiento de datos* tiene una vital importancia.

Para desarrollar, visualizar y ejecutar esta tesis se usa el entorno de *Jupyter Notebook.* Para la fase de modelización se ocupa un conjunto de datos (*card1*) extraído de la plataforma *Kaggle,* el cual contiene transacciones realizadas con tarjetas de crédito en septiembre de 2013 por titulares de tarjetas europeo. Este conjunto presenta transacciones que ocurrieron en dos días con 492 fraudes de un total de 284,807 transacciones, las cuales incluyen solo variables de entrada numéricas que son el resultado de una transformación de PCA *(Principal Component Analysis, en sus siglas en inglés).* Desafortunadamente, debido a problemas de confidencialidad, no se proporcionan las características originales y más información de fondo sobre los datos. Las características de $V1$ a $V28$ son los componentes principales obtenidos mediante PCA y las únicas que no se han transformado son *Tiempo* y *Cantidad.* La característica Tiempo contiene los segundos transcurridos entre cada transacción, la variable Cantidad contiene el monto realizado por cada paersona y la característica Clase es la variable de respuesta que toma el valor 1 en caso de fraude y 0 en caso contrario.

Análisis exploratorio

La visualización y estadística descriptiva pueden ayudar a comprender el contenido de los datos, evaluar la calidad y descubrir conocimientos iniciales. Por esto, es conveniente realizar un análisis exploratorio para conocer de manera preliminar como está conformada la base de datos en estudio y así, tener una idea preliminar de las tendencias claves existentes.

Realizar modelos de machine learning en Python es una de las formas más conveniente para trabajar, ya que es un lenguaje de programación interpretado que busca desarrollar una sintaxis que priorice la legibilidad del código. Este lenguaje de programación es conocido como multiparadigma, ya que soporta diferentes orientaciones y desde su lanzamiento en 1991 ha tenido un crecimiento asombroso convirtiéndose en uno de los preferidos por los programadores y con el auge de la inteligencia artificial tomó el liderato al ser el lenguaje de programación más utilizado para el desarrollo de este tipo de proyectos.

A continuación, en el código [1], se procede a importar algunas librerías de gran utilidad para el desarrollo de esta tesis, las cuales harán que el proceso de aprendizaje sea bastante amigable. Entre ellas, se describen:

Pandas: es una herramienta que permite trabajar con conjuntos de datos tabulares o en columnas. Por ejemplo, se puede analizar, organizar, clasificar, filtrar, agregar, limpiar, calcular en el conjunto de datos si se necesita.

Matplotlib: es una biblioteca de visualización para gráficos $2D$ de matrices, cuya multiplataforma está construida sobre matrices *NumPy*. Uno de los mayores beneficios de la visualización es permitir el acceso visual a grandes cantidades de datos en imágenes fácilmente digeribles, por ejemplo, gráficos en barra, gráficos de dispersión, histograma, entre otros.

Seaborn: es una biblioteca para hacer gráficos estadísticos en Python. Está construido sobre *matplotlib* y está estrechamente integrado con las estructuras de datos de *pandas*.

NumPy: ofrece funciones matemáticas complejas, generadores de números aleatorios, rutinas de álgebra lineal, transformadas de Fourier y más. *NumPy* es compatible con una amplia gama de plataformas informáticas y de hardware, cuyo núcleo está formado por un código en C optimizado.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
import seaborn as sns
import time
import tensorflow as tf
import collections
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import average_precision_score
from imblearn.metrics import classification_report_imbalanced
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score, recall_score,␣
 ↪f1_score,roc_auc_score, accuracy_score, classification_report
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
from collections import Counter
from pandas.plotting import scatter_matrix
warnings.filterwarnings("ignore")
```

Se comienza cargando el conjunto de datos, *card1,* en el entorno Jupyter para empezar el análisis exploratorio. Observar que en el código [2] se utiliza *rename()* para renombrar las columnas *Time, Amount* y *Class* como *Tiempo, Cantidad* y *Clase,* respectivamente.

[2]:
```
card=pd.read_csv('C:/Users/saave/OneDrive/Escritorio/Python.Final/creditcard.
 ↪csv',delimiter=',')
card1=card.rename(columns={'Time':'Tiempo','Amount':'Cantidad','Class':'Clase'})
```

Una herramienta útil de *Pandas* es la función *describe()*, la cual entrega un resumen de la tendencia central, la dispersión y la distribución de los datos respecto a las clases. Sin embargo, el conjunto de datos

puede contener valores vacíos no computables, los cuales se deben tratar antes de trabajar con ellos, a estos valores python los llama NaN (Not a Number). Dado que la función *describe()* excluye los *NaN*, se utiliza la función *isnull()*, la cual entrega un conteo de todos los valores no computables dentro de la base de datos.

[3]:
```
print('Número de NaN:', card1.isnull().sum().max())
print('Porcentaje de transacciones no fraudulentas:', round(card1['Clase'].
 ↪value_counts()[0]/len(card1) * 100,2), '% .')
print('Porcentaje de transacciones fraudulentas:', round(card1['Clase'].
 ↪value_counts()[1]/len(card1) * 100,2), '%.')
print('Proporción de desbalance (R) =',round(card1['Clase'].value_counts()[0]/
 ↪card1['Clase'].value_counts()[1],2))
card1[["Tiempo","Cantidad","Clase"]].describe()
```

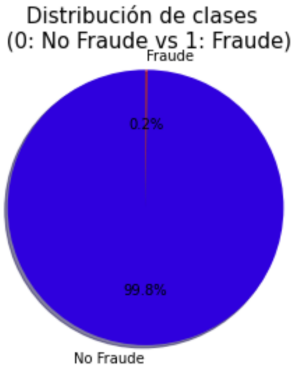
```
Número de NaN: 0
Porcentaje de transacciones no fraudulentas: 99.83 % .
Porcentaje de transacciones fraudulentas: 0.17 %.
Proporción de desbalance (R) = 577.88
```

	Tiempo	Cantidad	Clase
count	284807.000000	284807.000000	284807.000000
mean	94813.859575	88.349619	0.001727
std	47488.145955	250.120109	0.041527
min	0.000000	0.000000	0.000000
25%	54201.500000	5.600000	0.000000
50%	84692.000000	22.000000	0.000000
75%	139320.500000	77.165000	0.000000
max	172792.000000	25691.160000	1.000000

Se puede observar en la salida del código [3] que el conjunto de datos es altamente desbalanceado alcanzando un 99,83 % de transacciones genuinas y solo un 0,17 % de transacciones fraudulentas lo que representa un problema al momento de ajustar los clasificadores.

La distribución de las clases se ilustra en la salida del código [4], donde las transacciones fraudulentas se representan en rojo y las genuinas en azul.

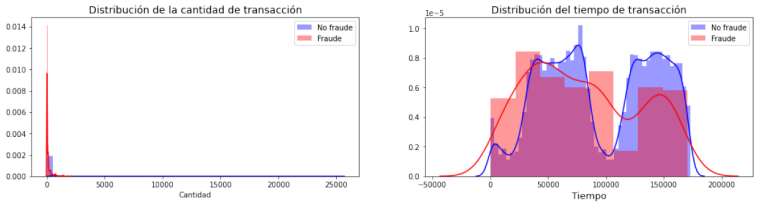
[4]:
```
color=["#0101DF", "#DF0101"]
sns.countplot(x='Clase', data=card1, palette=color)
plt.xlabel("Clase", fontsize=13)
plt.ylabel("Número de transacciones", fontsize=13)
plt.title('Distribución de clases \n (0: No Fraude vs 1: Fraude)', fontsize=15)
```



Observación 51 *Antes de realizar cualquier tarea de clasificación se debe tener en cuenta la distribución de las clases, ya que, por lo general, afecta a los algoritmos en su proceso de generalización perjudicando a las clases minoritarias. En consecuencia, si se entrenan los modelos propuestos con* 284,807 *elementos de la clase no fraudulenta y solo* 492 *de la otra clase, no se puede pretender que logre diferenciar una clase de otra. Lo más probable, es que el clasificador se limite a responder siempre "la transacción no es fraudulenta" puesto que así tuvo un acierto de aproximadamente el* 99 % *en su fase de entrenamiento. Por consiguiente, los resultados que arrojan las métricas de desempeño estarán sesgados.*

A continuación, se utiliza la función *subplots()* de *Matplotlib* y *displots()* de *Seaborn* para visualizar la distribución de las columnas *Tiempo* y *Cantidad* con el objetivo de tener una idea general de su comportamiento.

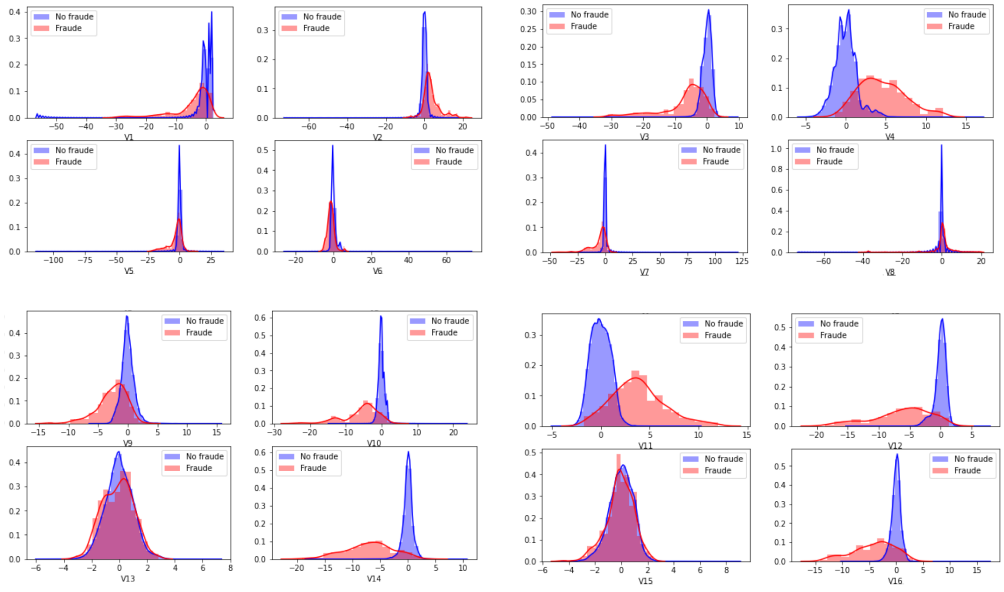
[5]:
```
fig, ax = plt.subplots(1, 2, figsize=(18,4))
sns.distplot(card1[card1['Clase'] == 0]['Tiempo'], ax=ax[1], label='No fraude',␣
 ↪color='r')
sns.distplot(card1[card1['Clase'] == 1]['Tiempo'], ax=ax[1],␣
 ↪label='Fraude',color='b')
ax[1].legend()
ax[1].set_title('Distribución del tiempo de transacción', fontsize=14)
sns.distplot(card1[card1['Clase'] == 0]['Cantidad'], ax=ax[0], label='No␣
 ↪fraude', color='r')
sns.distplot(card1[card1['Clase'] == 1]['Cantidad'], ax=ax[0], label='Fraude',␣
 ↪color='b')
ax[0].legend()
ax[0].set_title('Distribución de la cantidad de transacción', fontsize=14)␣
 ↪#título de los graficos
plt.xlabel("Tiempo", fontsize=13)
plt.show()
```

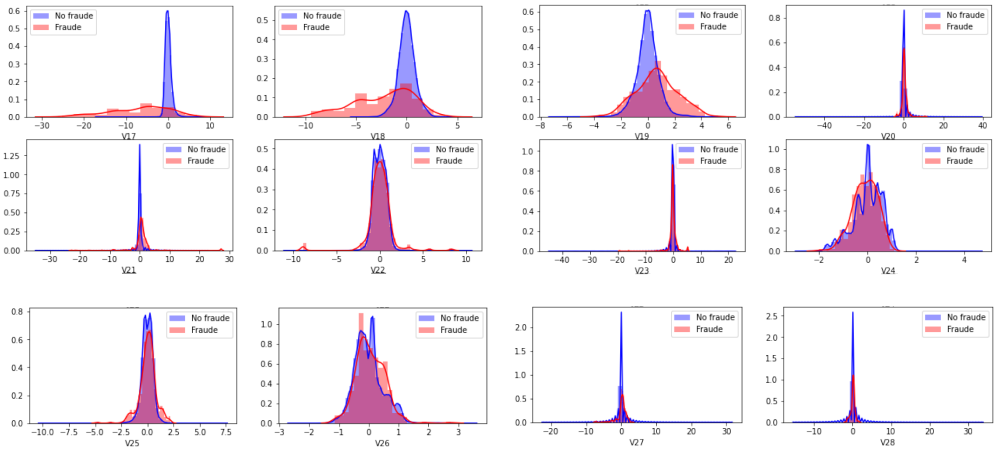


En la salida del código [5] la característica *Tiempo* (figura del lado derecho) tiene dos puntos extremos, incluso hay algunos extremos locales. Se puede pensar en estos como la hora del día en que la mayoría de la gente hace las transacciones y los mínimos se pueden interpretar como la hora de la noche cuando la mayor parte de la gente está durmiendo. Los datos contienen transacciones con tarjetas de crédito durante dos días, por lo que hay dos extremos máximos para cada día y un mínimo para la noche.

El código [6] muestra la distribución de clases de las características restantes. Esta evaluación visual puede ayudar a elegir un subconjunto de características para el problema de clasificación o ver cuál de ellas contiene valores atípicos que afecten al modelo.

```
[6]: fig1, ax1 = plt.subplots(7,4, figsize=(22,22))
     for i in range(28):
         sns.distplot(card1[card1['Clase'] == 0][f'V{i+1}'], ax=ax1[i//4,i%4],␣
      ↪label='No fraude',color='b')
         sns.distplot(card1[card1['Clase'] == 1][f'V{i+1}'], ax=ax1[i//4,i%4],␣
      ↪label='Fraude',color='r')
         ax1[i//4,i%4].legend()
     plt.show()
```





En la salida de código [6], algunas características presentan una buena separabilidad de clases. Por ejemplo, las características $V10$, $V12$ y $V14$ están parcialmente separadas incluso se puede apreciar cierta separación lineal, como se observa en la Figura 73. Por otra parte, $V1$, $V2$ y $V3$ tienen un perfil bastante distinto, mientras que las características de $V21$ a $V28$ tienen perfiles similares para las dos de clases, lo que puede estar indicando que estas características no son lo suficientemente explicativas. En general, a excepción de las características *Tiempo* y *Cantidad*, la distribución para transacciones genuinas (*Clase* 0) se centran alrededor de 0. Al mismo tiempo, las transacciones fraudulentas (Clase 1) tienden a tener una distribución asimétrica con una cola pesada lo que da indicios de que las transacciones fraudulentas tienden a tener valores extremos. Para algunas columnas, como $V15$, la distribución de fraudes es muy similar a la distribución de no fraudes.

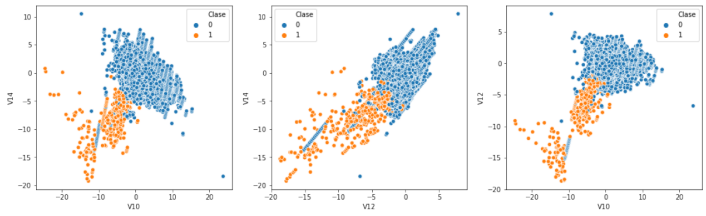
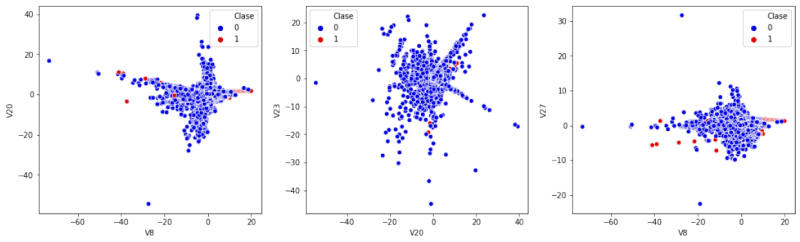


Figura 73. Distribuciones, respecto a las clases, de las características V10 vs V14, V12 vs V14 y V10 vs V12 donde se aprecia cierta separación lineal.

En la salida del código [7] se trazan las columnas $V8$ vs $V20$, $V23$ vs $V27$ y $V8$ vs $V27$ donde se observa que las clases están superpuestas por ciertos valores atípicos, lo que causa una baja en el desempeño al momento de ajustar los modelos de clasificación. Esta es una de las razones que hacen indispensable el

preprocesamiento de los conjuntos de datos.

[7]:
```
fig, ax = plt.subplots(1,3, figsize=(18,5))
color=["#0101DF", "#DF0101"]
sns.scatterplot(x='V8', y='V20', hue='Clase', data=card1, ax=ax[0],␣
 ↪palette=color)
sns.scatterplot(x='V20', y='V23', hue='Clase', data=card1,␣
 ↪ax=ax[1],palette=color)
sns.scatterplot(x='V8', y='V27', hue='Clase', data=card1, ax=ax[2],palette=color)
```

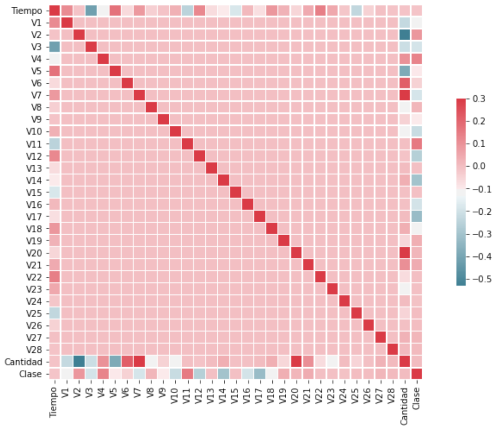


Correlación de las características

La evidencia empírica de la literatura de *selección de características* muestra que, junto con las características irrelevantes, también se debe eliminar la información redundante. Se dice que una característica es redundante si una o más de las otras características están altamente correlacionadas. Sin embargo, aunque no se sabe el significado de las características V, será útil comprender como influyen en el resultado. Si hay una correlación muy alta entre dos características mantenerlas no es una buena idea, ya que esto puede causar un ajuste excesivo.

La matriz de correlación que se ilustra en la salida del código [8] muestra que ninguna de los componentes, después de haber realizado el PCA, tiene alguna correlación entre si (como era de esperar). No obstante, se observa que la columna *Clase* está correlacionada positiva y negativamente con algunos componentes V, lo cual es bueno, ya que un buen subconjunto de características es aquel que contiene características altamente correlacionadas con la *Clase*.

[8]:
```
corr = card1.corr()
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
cmap = sns.diverging_palette(220, 10, as_cmap=True)
f, ax2 = plt.subplots(figsize=(14,8))
ax2 = sns.heatmap(corr, cmap=cmap, vmax=.3, square=True, linewidths=.5,␣
 ↪cbar_kws={"shrink": .5}, annot=False)
```



La función *corr()* de *Pandas* realiza la correlación de todas las variables, excluyendo los NaN. Esta función, tiene la opción *method,* la cual entrega la correlación de Pearson, Kendall o Spearman según se especifique. La salida [9] entrega la correlación de Pearson respecto a las primeras cuatro características V junto con la característica *Tiempo.*

```
[9]: round(corr.head(),3)
```

```
[9]:          Tiempo     V1     V2     V3     V4     V5     V6     V7     V8     V9  \
     Tiempo    1.000  0.117 -0.011 -0.42 -0.105  0.173 -0.063  0.085 -0.037 -0.009
     V1        0.117  1.000  0.000 -0.00  0.000  0.000  0.000  0.000 -0.000  0.000
     V2       -0.011  0.000  1.000  0.00 -0.000 -0.000  0.000  0.000 -0.000 -0.000
     V3       -0.420 -0.000  0.000  1.00 -0.000 -0.000  0.000  0.000  0.000 -0.000
     V4       -0.105  0.000 -0.000 -0.00  1.000 -0.000 -0.000  0.000  0.000  0.000

                ...    V21    V22    V23    V24    V25    V26    V27    V28  Cantidad  \
     Tiempo     ...  0.045  0.144  0.051 -0.016 -0.233 -0.041 -0.005 -0.009    -0.011
     V1         ... -0.000  0.000  0.000  0.000 -0.000 -0.000  0.000  0.000    -0.228
     V2         ...  0.000  0.000  0.000 -0.000 -0.000  0.000 -0.000 -0.000    -0.531
     V3         ... -0.000  0.000  0.000 -0.000  0.000  0.000  0.000  0.000    -0.211
     V4         ... -0.000  0.000  0.000  0.000  0.000 -0.000 -0.000 -0.000     0.099

              Clase
     Tiempo -0.012
     V1     -0.101
     V2      0.091
     V3     -0.193
     V4      0.133
```

Preparación de los datos

La etapa de preparación de los datos comprende todas las actividades realizadas para construir el conjunto de datos que se utilizará en la etapa de modelización. Estos incluyen la limpieza y transformación de los datos en variables más útiles, lo que mejora la precisión del modelo evitando posibles sesgos. Esta

etapa de preparación es la que lleva más tiempo, sin embargo, se puede reducir significativamente si los recursos de datos están bien administrados, bien integrados y limpios desde una perspectiva analítica, no meramente de almacenamiento.

Normalización: Algunos algoritmos de aprendizaje automático como las redes neuronales, los vecinos más cercanos, la regresión lineal o la logística funcionan mejor o convergen más rápido cuando las características están en una escala relativamente similar y cerca de la distribución normal, en consecuencia, es recomendable normalizar las variables de entrada. Normalizar significa, en este caso, comprimir o extender los valores de las variables para que estén en un rango definido. Sin embargo, una mala aplicación de la normalización, o una elección descuidada del método puede arruinar los datos, y con ello el análisis. Es por esto, que al elegir un método, si corresponde, va a depender del tipo de modelo y los valores de sus características. A continuación, se describen tres opciones de normalización que entrega *scikit-learn* para escalar los datos:

a) MinMaxScaler

Para cada valor de una característica, *MinMaxScaler* resta el valor mínimo de la característica y luego lo divide por su rango. El rango es la diferencia entre el máximo y el mínimo. *MinMaxScaler* conserva la forma de la distribución original y no cambia significativamente la información incrustada en los datos originales. El rango predeterminado para la función devuelta por *MinMaxScaler* es de 0 a 1.

$$\frac{x_i - \text{mín}}{\text{máx} - \text{mín}}. \tag{4.1}$$

Se debe tener en cuenta que *MinMaxScaler* no reduce la importancia de los valores atípicos.

b) Escalador robusto

Si los datos contienen muchos valores atípicos, es probable que el escalado utilizando la media y la varianza de los datos no funcione muy bien. En estos casos, es recomendable usar *RobustScaler* como reemplazo directo, ya que utiliza estimaciones más sólidas para el centrado y rango de sus datos. *RobustScaler* transforma el vector de características restando la mediana y luego dividiendo el rango intercuartílico, como se muestra en la siguiente ecuación

$$\frac{x_i - mediana}{p_{75} - p_{25}}, \tag{4.2}$$

donde p_{75} y p_{25} son el percentil 75 y 25, respectivamente.

c) Escalador estándar

La función *StandardScaler* estandariza los datos restando la media y luego escalando a la varianza de la unidad. *StandardScaler* da como resultado una distribución con una desviación estándar igual a 1 y media 0. Aproximadamente el 68 % de los valores estarán entre -1 y 1. De esta manera, se obtiene la siguiente ecuación

$$\frac{x_i - \mu}{\sigma}, \tag{4.3}$$

donde μ es media y σ es la desviación estándar del conjunto de datos. Se debe tener en cuenta que es posible que los datos se comporten mal si las características individuales no se parecen (más o menos) a los datos distribuidos normalmente estándar.

Dado que la mayoría de las características del conjunto de datos ya están normalizadas, se procede a escalar las columnas restantes (*Cantidad* y *Tiempo*). Como se ilustra en el código [10], se utiliza la función *MinMaxScaler*, pues se busca mantener la distribución de los datos y dado que se trabaja con un alto desbalance, se tiene que al momento de equilibrar las clases generalmente se procede a crear o eliminar instancias lo que puede modificar de cierta manera su distribución, en consecuencia, es recomendable influir lo menos posible al momento de normalizar.

[10]:
```
from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler
MinMax_scaler = MinMaxScaler()
card1['Cantidad escalada'] = MinMax_scaler.fit_transform(card1['Cantidad'].
 ↪values.reshape(-1,1))
card1['Tiempo escalado'] = MinMax_scaler.fit_transform(card1['Tiempo'].values.
 ↪reshape(-1,1))
card1.drop(['Tiempo','Cantidad'], axis=1, inplace=True)
scaled_amount = card1['Cantidad escalada']
scaled_time = card1['Tiempo escalado']
card1.drop(['Cantidad escalada', 'Tiempo escalado'], axis=1, inplace=True)
card1.insert(0, 'Cantidad escalada', scaled_amount)
card1.insert(1, 'Tiempo escalado', scaled_time)
card1.head()
```

[10]:

	Cantidad escalada	Tiempo escalado	V1	V2	V3	V4	\
0	0.005824	0.000000	-1.359807	-0.072781	2.536347	1.378155	
1	0.000105	0.000000	1.191857	0.266151	0.166480	0.448154	
2	0.014739	0.000006	-1.358354	-1.340163	1.773209	0.379780	
3	0.004807	0.000006	-0.966272	-0.185226	1.792993	-0.863291	
4	0.002724	0.000012	-1.158233	0.877737	1.548718	0.403034	

	V5	V6	V7	V8	...	V20	V21	V22	\
0	-0.338321	0.462388	0.239599	0.098698	...	0.251412	-0.018307	0.277838	
1	0.060018	-0.082361	-0.078803	0.085102	...	-0.069083	-0.225775	-0.638672	
2	-0.503198	1.800499	0.791461	0.247676	...	0.524980	0.247998	0.771679	
3	-0.010309	1.247203	0.237609	0.377436	...	-0.208038	-0.108300	0.005274	
4	-0.407193	0.095921	0.592941	-0.270533	...	0.408542	-0.009431	0.798278	

	V23	V24	V25	V26	V27	V28	Clase
0	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	0
1	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	0
2	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	0
3	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	0
4	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	0

Observación 52 *MinMaxScaler es sensible a valores atípicos, que pueden afectar significativamente el rendimiento de un modelo de aprendizaje automático. Por lo tanto, es importante detectar y evaluar su influencia en el modelo. Esto se debe a que los valores atípicos pueden debilitar el poder estadístico, ya que pueden distorsionar nuestra comprensión de los datos y reducir el rendimiento del aprendizaje. Sin embargo, en algunos casos, los valores atípicos pueden ser objeto de interés, como en la detección de anomalías (técnica utilizada para identificar patrones inusuales que no se ajustan al comportamiento esperado).*

4.2. Ajuste y desempeño de los clasificadores

Analizar el rendimiento de los algoritmos de aprendizaje en presencia de un desequilibrio de clases es una tarea difícil, ya que para algunas medidas de desempeño (como la *efectividad*) se ven afectadas, pues la prevalencia de la clase mayoritaria puede enmascarar un rendimiento de clasificación deficiente en clases poco frecuentes.

En esta sección se presentan algunas medidas de desempeño que se utilizan frecuentemente para evaluar el rendimiento de los modelos ajustados, por ejemplo, la regresión logística, la SVM o las redes neuronales. En esta etapa se determina qué modelo es el más apropiada para tratar el problema y que técnicas aplicar de forma consistente atendiendo a los datos, recursos y necesidades que se tienen. Por lo general, se puede volver a la fase anterior (preprocesamiento) para tener una entrada acorde a las necesidades.

4.2.1. Métricas de desempeño

La calidad de los modelos generalmente se evalúa analizando qué tan bien se desempeñan en los datos de prueba. Para ver la fiabilidad de los ajustes hechos y poder comparar los diferentes modelos se utilizarán las métricas de *exactitud, precisión, sensibilidad o recall, especificidad, la matriz de confusión y la curva ROC.* Estas medidas, son un buen indicador para cuantificar el desempeño y comparar los resultados obtenidos.

Matriz confusión

Una forma conveniente de resumir el rendimiento de los clasificadores es hacer una tabulación cruzada entre las clases reales y las predichas. La tabulación cruzada resultante es una matriz, denominada matriz de confusión. Las columnas de la matriz de confusión representan los recuentos de instancias en las clases predichas, mientras que las filas representan los recuentos de instancias en las clases reales (o viceversa), es decir, en términos prácticos permite ver qué tipos de aciertos y errores está teniendo el modelo a la hora de pasar por el proceso de aprendizaje. En la Figura 66, se muestra un ejemplo de una matriz de confusión para un problema de clases binarias. Las siguientes cuatro opciones son las que conforman la matriz de confusión:

- **Verdaderos positivos (VP o TP):** Es la cantidad de casos positivos que fueron clasificados correctamente por el modelo. Por ejemplo, visto como un problema de fraude crediticio serían las transacciones fraudulentas que el modelo las clasificó correctamente como fraude.
- **Falsos positivos (FP):** Es la cantidad de casos negativos que fueron clasificados incorrectamente como positivos. Por ejemplo, visto como un problema de fraude crediticio serían las transacciones genuinas que el modelo las clasificó incorrectamente como fraude.
- **Verdadero Negativo (VN o TN):** Es la cantidad de casos negativos que fueron clasificados correctamente por el modelo. Por ejemplo, visto como un problema de fraude crediticio serían las transacciones no fraudulentas que el modelo las clasificó correctamente como genuinas.
- **Falso Negativo (FN):** Es la cantidad de casos positivos que fueron clasificados incorrectamente como negativos. Por ejemplo, visto como un problema de fraude crediticio serían las transacciones fraudulentas que el modelo las clasificó como no fraudulentas.

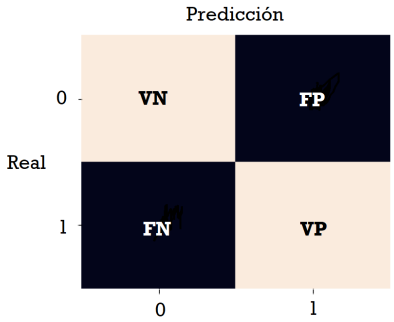


Figura 66. Representación gráfica de la matriz de confusión para el caso binario.

Mediante los resultados de la matriz de confusión se pueden derivar diferentes métricas de desempeño. Estas medidas corresponden a diferentes visiones de lo que constituye un buen clasificador y dependerá de las necesidades de cada problema. De esta manera, se evalúan las fortalezas y debilidades de cada modelo.

Definición 53 *La precisión es la capacidad del clasificador para no etiquetar como positiva una muestra negativa, es decir, entrega la certeza del modelo y cuantifica que tan seguro es. La precisión se calcula como*

$$precisión = \frac{VP}{VP + FP}. \tag{4.4}$$

Definición 54 *Recall o sensibilidad mide la capacidad del modelo para detectar muestras positivas. Cuanto mayor sea el recall, más muestras positivas detectadas Este se calcula como*

$$recall = \frac{VP}{VP + FN}. \tag{4.5}$$

Ejemplo 55 *Visto como un problema de clasificación de fraude con tarjetas de crédito, la precisión sería la cantidad de casos fraudulentos que el modelo es capaz de detectar del total de las transacciones fraudulentas.*

Observación 56 *El Recall es un buen indicador para seleccionar un buen modelo cuando hay un alto costo asociado con los falsos negativos. Por ejemplo, en la detección de fraudes con tarjetas de créditos si el clasificador predice una transacción fraudulenta (positivo real) como genuina (predicción negativa) las consecuencias para el ente financiero pueden ser considerables.*

Definición 57 *La especificidad es el número de ejemplos negativos que el algoritmo ha clasificado correctamente, es decir, cuando la clase es negativa, que porcentaje se logra clasificar correctamente. La especificidad se calcula como*

$$especificidad = \frac{VN}{VN + FP}. \tag{4.6}$$

Observación 58 *La especificidad mide lo opuesto a Recall.*

Definición 59 *Una métrica de rendimiento común y ampliamente utilizada para evaluar el rendimiento de la clasificación es la exactitud. La exactitud (4.7) es el porcentaje de instancias correctamente clasificadas. Corresponde a la suma de los elementos diagonales en la matriz de confusión (que en el caso binario es $VP + VN$) dividida por el número total de instancias.*

$$exactitud = \frac{VP + VN}{VP + VN + FP + FN} \tag{4.7}$$

Observación 60 *La exactitud es la medida más directa de la calidad de los clasificadores. Su valor se encuentra entre* 0 *y* 1 *y cuanto más alto, mejor. Sin embargo, aunque se utiliza ampliamente y es fácil de calcular e interpretar presenta algunos inconvenientes en conjuntos de datos desbalanceados. En primer lugar, es fácil obtener una alta exactitud (o una tasa de error baja) en problemas muy desequilibrados. Por ejemplo, en un conjunto donde la clase mayoritaria contiene* 1000 *datos y la minoritaria solo* 10*, si un clasificador asigna sistemáticamente a la clase mayoritaria las nuevas instancias, se logrará un* 99 % *de exactitud (tasa de error del* 1 %*). En segundo lugar, asume que los errores tienen el mismo costo. Sin embargo, en conjuntos desbalanceados clasificar erróneamente instancias de la clase minoritaria es generalmente mucho más costoso que clasificar erróneamente instancias de la clase mayoritaria.*

Entre las diferentes métricas para la evaluación del rendimiento existen herramientas gráficas que también ayudan a entender la eficacia del modelo, por ejemplo, la curva *ROC*. Este método toma en consideración un umbral de decisión para evaluar si una muestra se clasifica como positiva o negativa. Este valor se fija entre 0 y 1 y se aplica sobre el resultado de la probabilidad de pertenecer a una u otra clase en la salida del modelo.

Definición 61 *La curva ROC (Receiver Operating Characteristics) es una técnica para visualizar, organizar y seleccionar clasificadores basado en su rendimiento. El gráfico de una curva ROC es bidimensional, donde la sensibilidad se gráfica en el eje y y* 1*-especificidad en el eje* **x**.

La curva ROC (vista como un problema de clasificación de fraudes) es una métrica que constituye un método estadístico que ayuda determinar la capacidad del modelo para diferenciar entre transacciones genuinas versus transacciones fraudulentas describiendo la compensación entre los beneficios (VP) y los costos (FP). Para esto, se mide el área bajo la curva (AUC en sus siglas en ingles) ROC y se interpreta *como la probabilidad de que ante una transacción* (fraudulenta o genuina) *el modelo la clasifique correctamente*, es decir, que tan bien se están distinguiendo los dos grupos. El punto $(0, 1)$ representa la clasificación perfecta, por consiguiente, un buen modelo se encuentra en la región superior a la diagonal

$y = x$ y con una tendencia al punto $(0, 1)$, ver Figura 68.

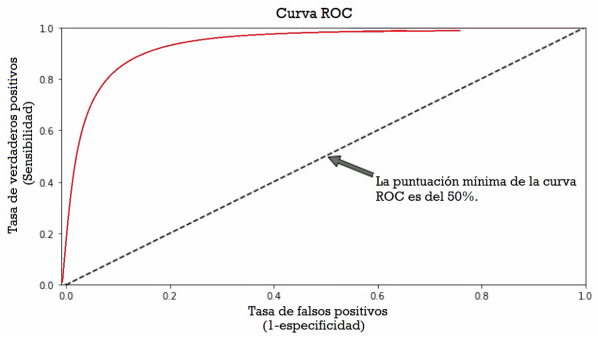


Figura 68. Ilustración de la curva ROC, donde el eje x esta dado por 1-especificidad y el eje y representa la sensibilidad.

4.2.2. Ajuste y comparación de los clasificadores

Después de haber finalizado la etapa del preprocesamiento de datos se procede con la fase de modelización. Los modelos a ajustar y, posteriormente comparar (respecto a las medidas de desempeño antes mencionadas) serán los modelos de regresión logística, SVM y redes neuronales.

Submuestreo aleatorio

Se comienza con la técnica de submuestreo aleatorio que, como se mencionó anteriormente, consiste básicamente en eliminar datos de forma aleatoria y, de esta manera, lograr el equilibrio de clases. Para conseguir esto, se siguen los siguientes pasos:

- Lo primero es determinar cuán desequilibrada están las clases. Como se vio en el análisis exploratorio, existen 492 transacciones fraudulentas versus 284,315 transacciones genuinas.
- Antes de realizar el submuestreo aleatorio, se mezclan los datos para mantener la aleatoriedad y ver si el modelo tiene cierta precisión cada vez que se ejecuta.
- Por último, se procede a equilibrar las clases con la misma proporción, es decir, 492 casos de fraude y 492 casos de transacciones no fraudulentas, lo que genera un nuevo conjunto de datos.

Siguiendo los pasos anteriores, se utiliza la función *sample()* (de Pandas) para generar una columna aleatoria a partir del conjunto de datos *card1*. Posteriormente, se escogen aleatoriamente 492 transacciones no fraudulentas utilizando la función *loc()* y así, finalmente ocupar la función *concat()* para concatenar las dos clases formando el nuevo conjunto de datos, *nuevo_ card1,* como se muestra en el código [12].

[12]:
```
card1 = card1.sample(frac=1)
fraude_card1 = card1.loc[card1['Clase'] == 1]
no_fraude_card1 = card1.loc[card1['Clase'] == 0][:492]
distr_card1 = pd.concat([fraude_card1, no_fraude_card1])
nuevo_card1 = distr_card1.sample(frac=1, random_state=42)
nuevo_card1.head()
```

[12]:

	Cantidad escalada	Tiempo escalado	V1	V2	V3	\
144957	2.989355e-04	0.500700	2.144891	-0.029341	-2.600130	
154684	3.892389e-07	0.593615	-28.709229	22.057729	-27.855811	
247547	9.130378e-03	0.889005	-0.912660	-0.305367	-0.571259	
154454	5.755676e-03	0.587973	0.913116	1.145381	-4.602878	
6882	4.281628e-05	0.050975	-4.617217	1.695694	-3.114372	

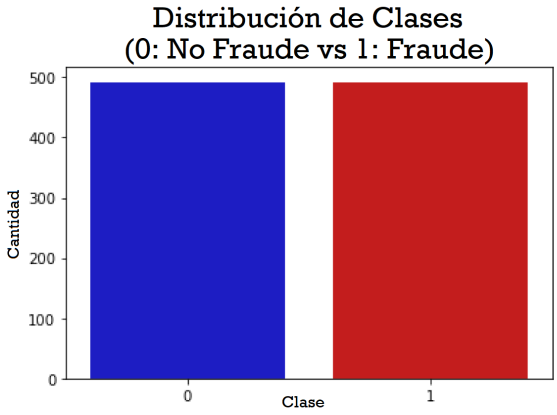
	V4	V5	V6	V7	V8	...	V20	\
24662	1.224589	1.031176	0.135486	0.016486	0.070978	...	0.155229	
251881	4.173516	1.239751	-0.746186	0.572731	-0.131235	...	-0.172216	
268280	-0.867155	1.032594	-1.243194	0.230685	0.342547	...	-0.666174	
9509	10.313349	-4.351341	-3.322689	-10.788373	5.060381	...	1.434240	
142557	0.389760	-0.281214	-0.055123	1.326232	0.195700	...	0.634184	

	V21	V22	V23	V24	V25	V26	V27	\
24662	0.229343	0.712940	-0.378579	-0.791025	-0.158758	-0.046666	0.089260	
251881	-0.301001	-0.818972	0.206812	-0.263683	-0.114958	-0.240672	-0.006629	
268280	0.359632	0.834083	-0.194228	-0.416408	0.270192	-0.228947	-0.487089	
9509	1.990545	0.223785	0.554408	-1.204042	-0.450685	0.641836	1.605958	
142557	0.325575	0.014002	0.844946	0.114963	0.156365	-0.619437	-0.120351	

	V28	Clase
24662	0.173367	0
251881	0.017258	1
268280	0.298282	0
9509	0.721644	1
142557	0.035594	1

En la salida del código [13] se ilustra las clases ya equilibradas donde la barra azul representa la clase no fraudulenta y en rojo la clase con las transacciones fraudulentas.

[13]:
```
print('Grafico de la distribución de las clases submuestreadas \n')
plt.ylabel("Número de transacciones")
plt.title('Distribución de clases \n (0: No Fraude vs 1: Fraude)', fontsize=19)
sns.countplot('Clase', data= nuevo_card1, palette=color)
```



Ahora se comienza con la fase de entrenamiento. Para esto, se deja un 80 % del conjunto de datos para el entrenamiento y un 20 % para el test. Una forma de hacerlo es utilizando la función *StratifiedKFold()* de *Scikit-learn*, la cual hace este proceso de forma estratificada lo que ayuda a conservar las proporciones de las clases. El código [14] muestra cómo se realiza esta separación.

[14]:
```
X =nuevo_card1.drop('Clase', axis=1)
y = nuevo_card1['Clase']

ssss = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)
for train_ind, test_ind in ssss.split(X,y):
    X_train, X_test = X.iloc[train_ind], X.iloc[test_ind]
    y_train, y_test = y.iloc[train_ind], y.iloc[test_ind]

X_train=  X_train.values
X_test = X_test.values
y_train= y_train.values
y_test = y_test.values
```

Separado el conjunto de datos *nuevo_card*1, en la salida del código [15] se puede observar el desempeño en el entrenamiento al ajustar los modelos de regresión logística y SVM. Se utiliza la función *cross_val_score()* y *score()* que devuelven la eficiencia de cada modelo (entendida como el porcentaje de valores clasificados correctamente con respecto al total de elementos). La función *cross_val_score()* evalúa el desempeño mediante *validación cruzada,* es decir, el conjunto de entrenamiento se divide en k conjuntos más pequeños (existen otros enfoques, pero generalmente siguen los mismos principios) garantizando que los resultados sean independientes de la partición entre datos de entrenamiento y prueba. La validación cruzada lleva a cabo el siguiente procedimiento para cada uno de los k-pliegues:

- Se entrena el modelo utilizando $k-1$ pliegues, luego se valida en el pliegue restante como si fuese un conjunto de prueba y, de esta manera, se calculan las medidas de rendimiento.
- La medida de rendimiento informada por la validación cruzada es el promedio de los valores calculados en el ciclo.

Observación 62 *Este enfoque puede ser costoso desde el punto de vista computacional, pero no desperdicia demasiados datos (como es el caso cuando se arregla un conjunto de validación arbitrario), lo cual es una gran ventaja cuando hay un número pequeño de muestras. Por otra parte, como la función score() no usa validación cruzada, puede ser un poco impreciso, ya que depende de la aleatoriedad en que se escoja el pliegue.*

[15]:
```
classifiers = { "Regresión logística": LogisticRegression(),"Maquina de vectores de soporte": SVC()}
for key, classifier in classifiers.items():
    classifier.fit(X_train, y_train)
    training_score = cross_val_score(classifier, X_train, y_train, cv=5)
    print("Clasificadores: ", classifier.__class__.__name__, "tiene puntuación mediante validación cruzada de",
          round(training_score.mean(), 2) * 100, "%.")
rl=LogisticRegression().fit(X_train,y_train)
print('Puntuación de la Regresión logística en el entrenamiento:',round(rl.score(X_train,y_train),2)*100,'%')
clf=SVC().fit(X_train,y_train)
print('Puntuación de la SVM en el entrenamiento:', round(clf.score(X_train,y_train), 2)*100,'%')
```

```
Clasificadores:  LogisticRegression tiene puntuación mediante validación cruzada
de 94.0 %.
Clasificadores:  SVC tiene puntuación mediante validación cruzada de 94.0 %.
Puntuación de la Regresión logística en el entrenamiento: 95.0 %
Puntuación de la SVM en el entrenamiento: 94.0 %
```

Como se puede apreciar en la salida del código [15], el desempeño mediante validación cruzada para la regresión logística resulta ser menor que la puntuación obtenida por la función *score()*. Generalmente esto ocurre, ya que la medida de rendimiento informada por la validación cruzada es el promedio de cada pliegue lo que conduce a resultados más realistas.

Otra función interesante de *sklearn* es *GridSearchCV()*. Esta permite seleccionar los valores de los hiperparámetros para un modelo haciendo una búsqueda exhaustiva de manera que evalúa todas las combinaciones. Esta estrategia tiene el inconveniente de que el gasto computacional es bastante alto, lo que lleva a invertir mucho tiempo en regiones de poco interés antes de evaluar otras combinaciones. Una alternativa, pero un poco menos precisa es hacer una búsqueda aleatoria, de esta forma, se consigue explorar el espacio de búsqueda de una forma más distribuida. *RandomizedSearchCV()* permite este tipo de estrategia, únicamente requiere que se le indique el espacio de búsqueda de cada hiperparámetro (lista de opciones o una distribución) y el número de combinaciones aleatorias a evaluar. A continuación, se utiliza *GridSearchCV()*, ya que el costo computacional en este caso es bajo.

In [16]:
```
reg_log_param = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.7 ,0.1,0.2, 1, 10, 100, 1000]}
grid_reg_log1 = GridSearchCV(LogisticRegression(), reg_log_param)
grid_reg_log1.fit(X_train, y_train)
reg_log1 = grid_reg_log1.best_estimator_
svc_params = {'C': [0.5, 0.7, 0.9, 1, 1.5], 'kernel': ['rbf', 'poly', 'sigmoid', 'linear']}
grid_svc1 = GridSearchCV(SVC(), svc_params)
grid_svc1.fit(X_train, y_train)
svc1= grid_svc1.best_estimator_
print(reg_log1)
print(svc1)
```

```
LogisticRegression(C=0.2)
SVC(C=0.9, kernel='linear')
```

[16]:
```
reg_log_param = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.7 ,0.1,0.2, 1,␣
↪10, 100, 1000]}
grid_reg_log1 = GridSearchCV(LogisticRegression(), reg_log_param)
grid_reg_log1.fit(X_train, y_train)
reg_log1 = grid_reg_log1.best_estimator_
svc_params = {'C': [0.5, 0.7, 0.9, 1, 1.5], 'kernel': ['rbf', 'poly', 'sigmoid',␣
↪'linear']}
grid_svc1 = GridSearchCV(SVC(), svc_params)
grid_svc1.fit(X_train, y_train)
svc1= grid_svc1.best_estimator_
print(reg_log1)
print(svc1)
```

```
LogisticRegression(C=0.2)
SVC(C=1.5)
```

Al ajustar los hiperparámetros de los modelos de regresión logística en 0,2 y la SVM en 1,5 se obtiene una leve mejoría en la puntuación de la validación cruzada, sin embargo, en la puntuación mediante la función *score()* la SVM mejora un 2 %, mientras que la regresión logística mejora un 1 %, como se puede observar en la salida del código [17].

[17]:
```
reg_log_score = cross_val_score(reg_log1, X_train, y_train, cv=5)
print('Puntuación con validación cruzada de la regresión logística: ',
 round(reg_log_score.mean() * 100, 2).astype(str) + '%')
svc_score = cross_val_score(svc1, X_train, y_train, cv=5)
print('Puntuación Configuración local/ validación cruzada de la SVM',
      round(svc_score.mean() * 100,2).astype(str) + '%')
print('=='*35)
rl2=LogisticRegression(C=1).fit(X_train,y_train)
print('Puntuación de la Regresión logística en el entrenamiento:',
      round(rl2.score(X_train,y_train),2)*100,'%')
clf2=SVC(C=0.5, kernel='linear').fit(X_train,y_train)
print('Puntuación de la SVM en el entrenamiento:', round(clf2.
↪score(X_train,y_train), 2)*100,'%')
```

```
Puntuación con validación cruzada de la regresión logística:  94.29%
Puntuación Configuración local/ validación cruzada de la SVM 94.29%
======================================================================
Puntuación de la Regresión logística en el entrenamiento: 95.0 %
Puntuación de la SVM en el entrenamiento: 96.0 %
```

Característica operativa del receptor

Ambos clasificadores alcanzaron un muy buen desempeño con la métrica de características operativas del receptor, ROC. En la salida del código [21] se observa que la SVM alcanzó un 98,14 % versus un 97,69 % de la regresión logística, siendo un resultado bastante parecido. Por consiguiente, ambos modelos

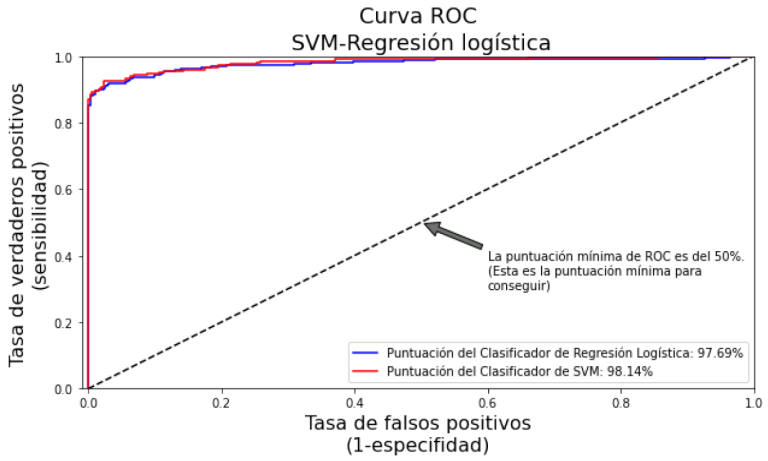
logran distinguir bastante bien ambas clases.

[21]:
```
log_reg_pred = cross_val_predict(reg_log1, X_train, y_train, cv=5,␣
 ↪method="decision_function")
svc_pred = cross_val_predict(svc1, X_train, y_train,␣
 ↪cv=5,method="decision_function")
print('La capacidad de distinguir VP y VN de la Regresión logística es: ',
      round(roc_auc_score(y_train, log_reg_pred)*100,2),'%')
print('La capacidad de distinguir VP y VN de la SVM: ',
      round(roc_auc_score(y_train, svc_pred)*100,2),'%')
```

```
La capacidad de distinguir VP y VN de la Regresión logística es:  97.69 %
La capacidad de distinguir VP y VN de la SVM:  98.14 %
```

La representación gráfica de la curva ROC se ilustra en la salida del código [22], en la cual se puede observar que la curva está bastante cerca del punto $(0, 1)$ lo que indica el bien desempeño de ambos modelos, bajo esta métrica, en el entrenamiento.

[22]:
```
log_fpr, log_tpr, log_thresold = roc_curve(y_train, log_reg_pred)
svc_fpr, svc_tpr, svc_threshold = roc_curve(y_train, svc_pred)
def graph_roc_curve_multiple(log_fpr, log_tpr, svc_fpr, svc_tpr):
    plt.figure(figsize=(10,5))
    plt.title('Curva ROC\n SVM-Regresión logística', fontsize=18)
    plt.plot(log_fpr, log_tpr,'b-' ,label=
             'Puntuación del Clasificador de Regresión Logística: {:.2%}'
             .format(roc_auc_score(y_train, log_reg_pred)))
    plt.plot(svc_fpr, svc_tpr,'r-',
             label='Puntuación del Clasificador de SVM: {:.2%}'.
 ↪format(roc_auc_score(y_train, svc_pred)))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([-0.01, 1, 0, 1])
    plt.xlabel('Tasa de falsos positivos\n(1-especifidad)', fontsize=16)
    plt.ylabel('Tasa de verdaderos positivos\n(sensibilidad)', fontsize=16)
    plt.annotate('La puntuación mínima de ROC es del 50%. \n(Esta es la␣
 ↪puntuación mínima para\nconseguir)',
                 xy=(0.5, 0.5), xytext=(0.6, 0.3),␣
 ↪arrowprops=dict(facecolor='#6E726D', shrink=0.05),)
    plt.legend()
graph_roc_curve_multiple(log_fpr, log_tpr, svc_fpr, svc_tpr)

plt.show()
```



Conjunto de prueba

Para ver el desempeño de cada clasificador, *Sklearn,* contiene la función *classification_ report,* la cual crea un informe de texto que muestra las principales métricas de clasificación, entregando el rendimiento de cada clase. La salida del código [23] muestra el desempeño de la regresión logística y la SVM tomando las métricas de precisión, exactitud y f1-score del conjunto de prueba.

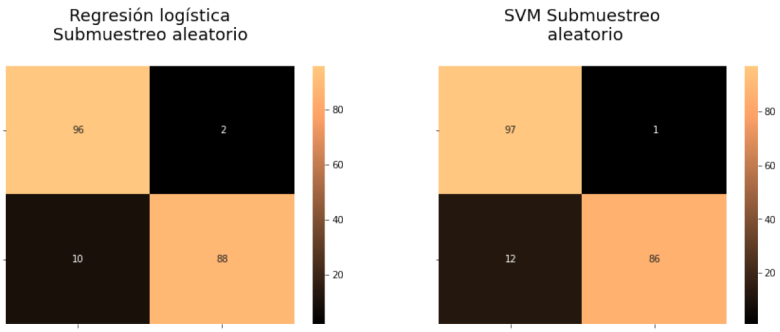
[23]:
```
print('='*55)
print(' '*5,'Regresión logística (Submuestreo aleatorio)')
print('='*55)
labels = ['No Fraude', 'Fraude']
sub_prediction = reg_log1.predict(X_test)
print(classification_report(y_test, sub_prediction, target_names=labels))
print('='*55)
print(' '*12,'SVM (Submuestreo aleatorio)')
print('='*55)
sub_pred = svc1.predict(X_test)
print(classification_report(y_test, sub_pred, target_names=labels))
```

```
=======================================================
      Regresión logística (Submuestreo aleatorio)
=======================================================
              precision    recall  f1-score   support

   No Fraude       0.91      0.98      0.94        98
      Fraude       0.98      0.90      0.94        98

    accuracy                           0.94       196
   macro avg       0.94      0.94      0.94       196
weighted avg       0.94      0.94      0.94       196
```

```
=====================================================
            SVM (Submuestreo aleatorio)
=====================================================
              precision    recall  f1-score   support

   No Fraude       0.89      0.99      0.94        98
      Fraude       0.99      0.88      0.93        98

    accuracy                           0.93       196
   macro avg       0.94      0.93      0.93       196
weighted avg       0.94      0.93      0.93       196
```

Ambos modelos presentan resultados similares, pero la regresión logística tiene un recall levemente mayor comparado al de la SVM, lo que implica que está clasificando de mejor manera las transacciones fraudulentas, sin embargo, la diferencia no es significativa. Esto se puede verificar en la salida del código [24] mediante la matriz confusión, que muestra un desempeño prácticamente igual entre los dos modelos.

[24]:
```
y_pred_log_reg = reg_log1.predict(X_test)
y_pred_svc = svc1.predict(X_test)
log_reg_cf = confusion_matrix(y_test, y_pred_log_reg)
svc_cf = confusion_matrix(y_test, y_pred_svc)
fig, ax = plt.subplots(1, 2,figsize=(15,5))
sns.heatmap(log_reg_cf, ax=ax[0], annot=True, cmap=plt.cm.copper, fmt='2')
ax[0].set_title("\nRegresión logística\nSubmuestreo aleatorio\n", fontsize=18)
ax[0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0].set_yticklabels(['', ''], fontsize=14, rotation=360)
sns.heatmap(svc_cf, ax=ax[1], annot=True, cmap=plt.cm.copper, fmt='2')
ax[1].set_title("\nSVM Submuestreo\n aleatorio\n", fontsize=18)
ax[1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1].set_yticklabels(['', ''], fontsize=14, rotation=360)

plt.show()
```



Submuestreo mediante Near Miss

Ahora se sigue el mismo procedimiento que se utilizó con la técnica de submuestreo aleatorio, pero la eliminación de los datos de la clase mayoritaria se hace mediante near miss. A continuación, en el código [25], se separa el conjunto de datos de forma estratificada con 788 datos para el entrenamiento (80 %) y 196 para el conjunto de prueba. Por otra parte, se utiliza la biblioteca de código abierto *Imbalanced-learn* que se basa en *Scikit-learn* (importado como sklearn), la cual proporciona herramientas para tratar clases desbalanceadas. De esta manera, para equilibrar las clases se utiliza la función *NearMiss()* que entrega esta biblioteca.

[25]:
```
X_al = card1.drop('Clase', axis=1)
y_al = card1['Clase']

X_nearmiss, y_nearmiss = NearMiss(sampling_strategy='majority').
 ↪fit_sample(X_al,y_al)

ss = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)
for train_in, test_in in ss.split(X_nearmiss,y_nearmiss):
    X_train_al, X_test_al = X_nearmiss.iloc[train_in], X_nearmiss.iloc[test_in]
    y_train_al, y_test_al = y_nearmiss.iloc[train_in], y_nearmiss.iloc[test_in]

X_train_al=  X_train_al.values
X_test_al = X_test_al.values
y_train_al= y_train_al.values
y_test_al= y_test_al.values
print('Datos para el entrenamiento (80%):',X_train_al.shape)
print('Datos para el test (20%):',X_test_al.shape)
```

```
Datos para el entrenamiento (80%): (788, 30)
Datos para el test (20%): (196, 30)
```

Al ajustar los clasificadores, la puntación mediante validación cruzada de la regresión logística es mayor que la puntuación de la SVM en un 3 % como se puede observar en la salida del código [26].

[26]:
```
classifiers = { "Regresión logística": LogisticRegression(),"Maquina de vectores␣
 ↪de soporte": SVC()}
for key, classifier in classifiers.items():
    classifier.fit(X_train_al,y_train_al)
    training_score = cross_val_score(classifier, X_train_al, y_train_al, cv=5)
    print("Clasificadores: ", classifier.__class__.__name__, "tiene puntuación␣
 ↪mediante validación cruzada de",
          round(training_score.mean(), 2) * 100, "%.")
```

```
Clasificadores:  LogisticRegression tiene puntuación mediante validación cruzada
de 95.0 %.
Clasificadores:  SVC tiene puntuación mediante validación cruzada de 92.0 %.
```

El hiperpárametro óptimo encontrado mediante la función *GridShearchCV()*, para la regresión logística es 10 y para la SVM es 1,5 con un kernel lineal.

[27]:
```
reg_log_param = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.7 ,0.1,0.2, 1,␣
 ↪10, 100, 1000]}
grid_reg_log2 = GridSearchCV(LogisticRegression(), reg_log_param)
grid_reg_log2.fit(X_train_al, y_train_al)
reg_log2 = grid_reg_log2.best_estimator_
svc_params = {'C': [0.5, 0.7, 0.9, 1, 1.5], 'kernel': ['rbf', 'poly', 'sigmoid',␣
 ↪'linear']}
grid_svc2 = GridSearchCV(SVC(), svc_params)
grid_svc2.fit(X_train_al, y_train_al)
svc2= grid_svc2.best_estimator_
print(reg_log2)
print(svc2)
```

```
LogisticRegression(C=10)
SVC(C=1.5, kernel='linear')
```

Se puede observar en la salida del código [28] que ambos modelos presentaron una mejoría en su desempeño, especialmente la SVM, que subió su rendimiento un 3,18 % y comparado con el submuestreo aleatorio ambos clasificadores lo superan levemente en el conjunto de entrenamiento.

[28]:
```
reg_log_score_a = cross_val_score(reg_log2, X_train_al, y_train_al, cv=5)
print('Puntuación despues de utilizar GridSearchCV (regresión logística):',
      round(reg_log_score_a.mean() * 100, 2).astype(str) + '%')
svc_score_a = cross_val_score(svc2, X_train_al, y_train_al, cv=5)
print('Puntuación despues de utilizar GridSearchCV (SVM):',
      round(svc_score_a.mean() * 100,2).astype(str) + '%')
```

```
Puntuación despues de utilizar GridSearchCV (regresión logística): 95.05%
Puntuación despues de utilizar GridSearchCV (SVM): 95.18%
```

La curva ROC alcanzo un 96,59 % en la regresión logística y en la SVM un 97,03 %. Esto es un resultado bastante bueno, pero inferior al alcanzado con la técnica de submuestreo aleatorio.

[29]:
```
log_reg_pred_al = cross_val_predict(reg_log2, X_train_al, y_train_al, cv=5,␣
 ↪method="decision_function")
svc_pred_al = cross_val_predict(svc2, X_train_al, y_train_al,␣
 ↪cv=5,method="decision_function")
print('ROC:')
print('=='*50)
print('La capacidad de distinguir VP y VN de la Regresión logística es: ',
      round(roc_auc_score(y_train_al,log_reg_pred_al)*100,2),'%')
print('--'*50)
print('La capacidad de distinguir VP y VN de la SVM: ',
      round(roc_auc_score(y_train_al, svc_pred_al)*100,2),'%')
print('=='*50)
```

```
ROC:
================================================================================
La capacidad de distinguir VP y VN de la Regresión logística es:  96.59 %
--------------------------------------------------------------------------------
La capacidad de distinguir VP y VN de la SVM:  97.03 %
================================================================================
```

En el entrenamiento, el recall de la regresión logística es levemente mayor al recall de la SVM lo que indica que este clasificador está teniendo menos FN, es decir, las transacciones que son fraudulentas y que el modelo las detecto como genuinas son menores que las detectadas por la SVM, siendo esta métrica una de las que más importantes en estos casos debido al costo asociado, sin embargo, la precisión de la SVM es levemente más alta.

[31]:
```
y_pred_al = reg_log2.predict(X_train_al)
print('=' * 55)
print(' '*10,'Regresión logística (Near Miss)')
print('=' * 55)
print('Desempeño en el entrenamiento: \n')
print('Puntuación recall: {:.2%}'.format(recall_score(y_train_al, y_pred_al)))
print('Puntuación de precisión: {:.2%}'.format(precision_score(y_train_al,␣
 ↪y_pred_al)))
print('Puntuación F1: {:.2%}'.format(f1_score(y_train_al, y_pred_al)))
print('Puntuación de exactitud: {:.2%}'.format(accuracy_score(y_train_al,␣
 ↪y_pred_al)))
print('=' * 55)
y_pred_al_svm = svc2.predict(X_train_al)
print(' '*18,'SVM (Near Miss)')
print('=' * 55)
print('Desempeño en el entrenamiento: \n')
print('Puntuación recall: {:.2%}'.format(recall_score(y_train_al,␣
 ↪y_pred_al_svm)))
print('Puntuación de precisión: {:.2%}'.
 ↪format(precision_score(y_train_al,y_pred_al_svm)))
print('Puntuación F1: {:.2%}'.format(f1_score(y_train_al, y_pred_al_svm)))
print('Puntuación de exactitud: {:.2%}'.format(accuracy_score(y_train_al,␣
 ↪y_pred_al_svm)))
```

```
==========================================================
           Regresión logística (Near Miss)
==========================================================
Desempeño en el entrenamiento:

Puntuación recall: 94.42%
Puntuación de precisión: 98.67%
Puntuación F1: 96.50%
Puntuación de exactitud: 96.57%
==========================================================
                   SVM (Near Miss)
==========================================================
Desempeño en el entrenamiento:

Puntuación recall: 93.15%
Puntuación de precisión: 99.19%
Puntuación F1: 96.07%
Puntuación de exactitud: 96.19%
```

Conjunto de prueba

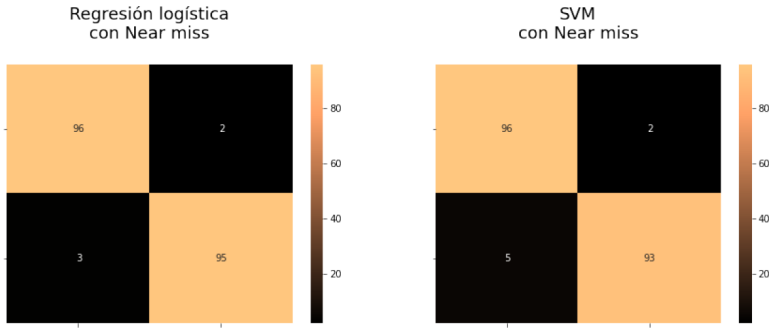
Como se puede ver en la salida del código [32], el desempeño de la regresión logística en el conjunto de prueba es levemente mayor que la SVM (en cada una de las métricas), lo que indica que está detectando de mejor manera las transacciones fraudulentas y las genuinas, aunque esta diferencia no es significativa. Por otra parte, el submuestreo mediante la técnica near miss es superior al submuestreo aleatorio efectuado anteriormente bajo las métricas de precisión, recall, f1-score y exactidud.

[32]:
```
print('='*55)
print(' '*11,'Regresión logística (Near Miss)')
print('='*55)
labels = ['No Fraude', 'Fraude']
prediction_al_lr = reg_log2.predict(X_test_al)
print(classification_report(y_test_al, prediction_al_lr, target_names=labels))
print('='*55)
print(' '*19,'SVM (Near Miss)')
print('='*55)
prediction_al_svm = svc2.predict(X_test_al)
print(classification_report(y_test_al, prediction_al_svm, target_names=labels))
```

```
=====================================================
           Regresión logística (Near Miss)
=====================================================
              precision    recall  f1-score   support

   No Fraude       0.97      0.98      0.97        98
      Fraude       0.98      0.97      0.97        98

    accuracy                           0.97       196
   macro avg       0.97      0.97      0.97       196
weighted avg       0.97      0.97      0.97       196


=====================================================
                  SVM (Near Miss)
=====================================================
              precision    recall  f1-score   support

   No Fraude       0.95      0.98      0.96        98
      Fraude       0.98      0.95      0.96        98

    accuracy                           0.96       196
   macro avg       0.96      0.96      0.96       196
weighted avg       0.96      0.96      0.96       196
```

Para contabilizar las predicciones hechas por estos modelos, se recurre a la matriz confusión. Se puede observar en la salida del código [33] que la regresión logística tuvo 5 instancias mal clasificadas versus 7 de la SVM. Esto indica que la regresión logística, mediante near miss, tuvo un mejor desempeño detectando falsos positivos.

[33]:
```
y_pred_log_al = reg_log2.predict(X_test_al)
y_pred_svc_al = svc2.predict(X_test_al)
log_reg_cf_al = confusion_matrix(y_test_al, y_pred_log_al)
svc_cf_al = confusion_matrix(y_test_al, y_pred_svc_al)
fig, ax = plt.subplots(1, 2,figsize=(15,5))
sns.heatmap(log_reg_cf_al, ax=ax[0], annot=True, cmap=plt.cm.copper, fmt='2')
ax[0].set_title("\nRegresión logística\ncon Near miss\n", fontsize=18)
ax[0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0].set_yticklabels(['', ''], fontsize=14, rotation=360)
sns.heatmap(svc_cf_al, ax=ax[1], annot=True, cmap=plt.cm.copper, fmt='2')
ax[1].set_title("\nSVM\ncon Near miss\n", fontsize=18)
ax[1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1].set_yticklabels(['', ''], fontsize=14, rotation=360)

plt.show()
```



Sobremuestreo mediante SMOTE

Cuando se desea equilibrar las clases mediante un sobremuestreo con la técnica SMOTE el gasto computacional que, comparado con las técnicas de submuestreo, es bastante mayor lo que puede originar a un tiempo de entrenamiento considerable. Sin embargo, la gran ventaja de estos métodos es la conservación de la información, pues no se eliminan instancias como en el caso del submuestreo aleatorio y near miss.

En el código [34] se utiliza la función *SMOTE()*, de *Imbalanced-learn,* para realizar el sobremuestreo de la clase minoritaria (opción *sampling_ strategy="minority"*) alcanzando un total de 554,077 transacciones y, al igual que antes, se separa el conjunto en entrenamiento y prueba con proporción de 80 % y 20 % respectivamente.

[34]:
```
submuestra_Xs = card1.drop('Clase', axis=1)
submuestra_ys = card1['Clase']

sm =SMOTE(sampling_strategy='minority', random_state=42)

Xs_train, ys_train = sm.fit_sample(submuestra_Xs, submuestra_ys)
s= StratifiedKFold(n_splits=5, random_state=None, shuffle=False)
for train_s, test_s in s.split(Xs_train, ys_train):
    print("Entrenamiento:", train_s, "Prueba:", test_s)
    submuestra_Xtrain, submuestra_Xtest = Xs_train.iloc[train_s], Xs_train.
 ↪iloc[test_s]
    submuestra_ytrain, submuestra_ytest = ys_train.iloc[train_s], ys_train.
 ↪iloc[test_s]

submuestra_Xtrain = submuestra_Xtrain.values
submuestra_Xtest = submuestra_Xtest.values
submuestra_ytrain = submuestra_ytrain.values
submuestra_ytest = submuestra_ytest.values
```

```
Entrenamiento: [ 55497  55498  55499 ... 554075 554076 554077] Prueba: [     0
1      2 ... 332444 332445 332446]
Entrenamiento: [     0      1      2 ... 554075 554076 554077] Prueba: [ 55497
55498  55499 ... 387852 387853 387854]
Entrenamiento: [     0      1      2 ... 554075 554076 554077] Prueba: [111002
111003 111004 ... 443260 443261 443262]
Entrenamiento: [     0      1      2 ... 554075 554076 554077] Prueba: [166508
166509 166510 ... 498667 498668 498669]
Entrenamiento: [     0      1      2 ... 498667 498668 498669] Prueba: [222016
222017 222018 ... 554075 554076 554077]
```

Al ajustar ambos modelos se puede observar, en la salida del código [35], que el desempeño de la SVM, bajo validación cruzada, es un 3 % mayor que el desempeño de la regresión logística alcanzando el mayor rendimiento (en la fase de entrenamiento) en lo que va de este capítulo, logrando un 98 %. Sin embargo, el tiempo de ejecución es bastante mayor respecto a las técnicas de submuestreo.

[35]:
```
t0=time.time()
classifiers = { "Regresión logística": LogisticRegression(),"Maquina de vectores␣
↪de soporte": SVC()}
for key, classifier in classifiers.items():
    classifier.fit(submuestra_Xtrain,submuestra_ytrain)
    training_score = cross_val_score(classifier, submuestra_Xtrain,␣
↪submuestra_ytrain, cv=5,n_jobs=6)
    print("Clasificadores: ", classifier.__class__.__name__, "tiene puntuación␣
↪mediante validación cruzada de",
          round(training_score.mean(), 2) * 100, "%.")
print('==' * 50)
t1=time.time()
print('Tiempo de ejecución:',round((t1-t0)/60,2),'min.')
print('==' * 50)
```

```
Clasificadores:  LogisticRegression tiene puntuación mediante validación cruzada
de 95.0 %.
Clasificadores:  SVC tiene puntuación mediante validación cruzada de 98.0 %.
================================================================================
Tiempo de ejecución: 56.14 min.
```

Para escoger los hiperparámetros óptimos en lugar de ocupar la función *GridShearchCV(),* como se ha hecho hasta ahora, se procede a utilizar la función *RandomizedSearchCV(),* la cual tiene un gasto computacional significativamente menor. Al utilizar *GridShearchCV()* el tiempo para encontrar los hiperparámetros óptimos es aproximadamente un 92 % más que *RandomizedSearchCV()* (70 horas versus 5,44 horas en el caso de SMOTE). En consecuencia, cuando se trabaja con conjuntos que contienen una gran cantidad datos es conveniente utilizar esta función. En la salida del código [36] se observa que los hiperparámetros

óptimos para la regresión logística y la SVM son 0,7 y 1,5, respectivamente.

[36]:
```
t0=time.time()
reg_log_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.7 ,0.1,0.2, 1,␣
 ↪10, 100,1000]}
grid_reg_log = RandomizedSearchCV(LogisticRegression(),␣
 ↪reg_log_params,n_iter=5,n_jobs=6)
grid_reg_log.fit(submuestra_Xtrain, submuestra_ytrain)
reg_log = grid_reg_log.best_estimator_
svc_params = {'C': [0.5, 0.7, 0.9, 1, 1.5], 'kernel': ['rbf', 'poly', 'sigmoid',␣
 ↪'linear']}
grid_svc = RandomizedSearchCV(SVC(), svc_params ,n_iter=4, n_jobs=6)
grid_svc.fit(submuestra_Xtrain, submuestra_ytrain)
svc= grid_svc.best_estimator_
print('='*45)
print(reg_log)
print('='*45)
print(svc)
print('='*45)
t1=time.time()
print('==' * 50)
print('Tiempo de ejecución:',round((t1-t0)/3600 ,2),'h.')
```

```
=============================================
LogisticRegression(C=0.7)
=============================================
SVC(C=1.5)
=============================================
Tiempo de ejecución: 5.44 h.
```

Observación 63 *Si bien, es posible que RandomizedSearchCV() no encuentre un resultado tan preciso como GridSearchCV(), sorprendentemente elige el mejor resultado la mayoría de las veces y en una fracción del tiempo que tomaría GridSearchCV. Con los mismos recursos, la búsqueda aleatoria puede incluso superar a la búsqueda en cuadrícula (GridSearchCV()).*

Al ajustar los hiperparámetros (código [37]) el resultado no cambia significativamente. Esto se puede deber a que la SVM y la regresión logística tienen por default el hiperparámetro $C = 1$, lo que no está

muy alejado de los valores óptimos entregados por *RandomizedSearchCV()*.

[37]:
```
t0=time.time()
reg_log_score_a = cross_val_score(reg_log, submuestra_Xtrain, submuestra_ytrain,␣
 ↪cv=5,n_jobs=6)

print('Puntuación con el mejor optimizador (regresión logística):',
 round(reg_log_score_a.mean() * 100, 2).astype(str) + '%')

svc_score_a = cross_val_score(svc, submuestra_Xtrain, submuestra_ytrain, cv=5,␣
 ↪n_jobs=6)

print('Puntuación con el mejor optimizador (SVM):', round(svc_score_a.mean() *␣
 ↪100,2).astype(str) + '%')
print('==' * 50)
t1=time.time()
print('==' * 50)
print('Tiempo de ejecución:',round(t1-t0)/60,2),'min.')
```

```
Puntuación con el mejor optimizador (regresión logística): 94.98%
Puntuación con el mejor optimizador (SVM): 98.03%
================================================================================

Tiempo de ejecución: 2 7 min.
```

En la salida del código [38] se observa la puntuación de la curva ROC obteniendo el mejor desempeño hasta el momento, bajo esta métrica, en la fase de entrenamiento.

[38]:
```
t0=time.time()
log_reg_pred_sm = cross_val_predict(reg_log, submuestra_Xtrain,␣
 ↪submuestra_ytrain, cv=5, method="decision_function",n_jobs=6)
svc_pred_sm = cross_val_predict(svc, submuestra_Xtrain, submuestra_ytrain,␣
 ↪cv=5,method="decision_function", n_jobs=6)
print('ROC:')
print('=='*50)
print('La capacidad de distinguir VP y VN de la Regresión logística es:␣
 ↪',round(roc_auc_score(submuestra_ytrain,
                                                                          ␣
 ↪              log_reg_pred_sm)*100,2),'%')
print('--'*50)
print('La capacidad de distinguir VP y VN de la SVM: ',␣
 ↪round(roc_auc_score(submuestra_ytrain, svc_pred_sm)*100,2),'%')
print('=='*50)
t1=time.time()
print('==' * 50)
print('Tiempo de ejecución:',round(t1-t0)/60,2),'min.')
```

```
ROC:
================================================================================
La capacidad de distinguir VP y VN de la Regresión logística es:  99.02 %
--------------------------------------------------------------------------------
La capacidad de distinguir VP y VN de la SVM:  99.8 %
================================================================================

Tiempo de ejecución: 25.4 min.
```

Conjunto de prueba

La SVM se comporta bastante bien con SMOTE, aunque se tarda mucho más en ser ajustada que la regresión logística, pero tiene muy buen desempeño en la precisión, recall, f1-score y exactitud alcanzando un 98 % en todas estas. Por consiguiente, la SVM con SMOTE es el mejor modelo hasta el momento, esto se puede deber a la generación de muestras sintéticas, las cuales, de alguna manera, ayudan al hiperplano de separación a generalizar ambas clases. Sin embargo, el gasto computacional es bastante mayor.

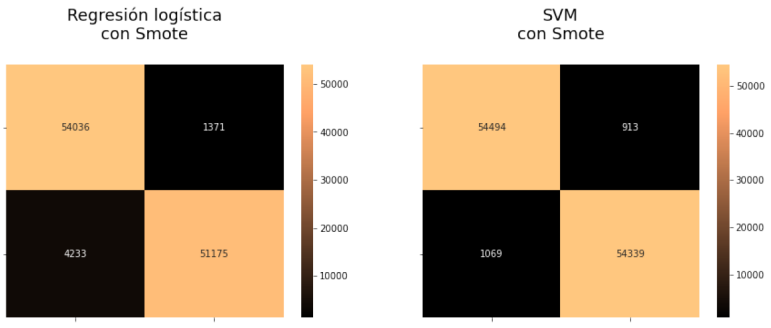
[41]:
```
t0=time.time()
print('='*55)
print(' '*17,'Regresión logística')
print('='*55)
labels = ['No Fraude', 'Fraude']
prediction_sm_lr = reg_log.predict(submuestra_Xtest)
print(classification_report(submuestra_ytest, prediction_sm_lr,␣
 ↪target_names=labels))
print('='*55)
print(' '*23,'SVM')
print('='*55)
prediction_sm_svm = svc.predict(submuestra_Xtest)
print(classification_report(submuestra_ytest, prediction_sm_svm,␣
 ↪target_names=labels))
print('=' * 55)
t1=time.time()
print('==' * 50)
print('Tiempo de ejecución:',round((t1-t0)/60,2),'min.')
```

```
=======================================================
                  Regresión logística
=======================================================
              precision    recall  f1-score   support

   No Fraude       0.93      0.98      0.95     55407
      Fraude       0.97      0.92      0.95     55408

    accuracy                           0.95    110815
   macro avg       0.95      0.95      0.95    110815
weighted avg       0.95      0.95      0.95    110815
```

```
=====================================================
                        SVM
=====================================================
              precision    recall  f1-score   support

   No Fraude       0.98      0.98      0.98     55407
      Fraude       0.98      0.98      0.98     55408

    accuracy                           0.98    110815
   macro avg       0.98      0.98      0.98    110815
weighted avg       0.98      0.98      0.98    110815

=====================================================
Tiempo de ejecución: 2.07 min.
```

La salida del código [42] muestra el total de las instancias acertadas junto a los errores cometidos por ambas clases. Se puede observar que la SVM supera ampliamente a la regresión logística alcanzando un total de 1,069 FN y 913 FP versus los 4,233 FN y 1,371 FP alcanzados por la regresión logística. Sin embargo, cuando se está presente en problemas de clasificación de este tipo alcanzar 1069 instancias clasificadas como falsos negativos pueden causar graves pérdidas a una institución financiera, pues el algoritmo está detectando operaciones genuinas siendo que estas son fraudulentas. En consecuencia, surge la necesidad de disminuir lo más posible estos tipos de errores, ya sea buscando un nuevo modelo o volviendo a la etapa de preprocesamiento de datos.

[42]:
```
y_pred_log_sm = reg_log.predict(submuestra_Xtest)
y_pred_svc_sm = svc.predict(submuestra_Xtest)
log_reg_cf_sm = confusion_matrix(submuestra_ytest, y_pred_log_sm)
svc_cf_sm = confusion_matrix(submuestra_ytest, y_pred_svc_sm)
fig, ax = plt.subplots(1, 2,figsize=(15,5))
sns.heatmap(log_reg_cf_sm, ax=ax[0], annot=True, cmap=plt.cm.copper, fmt='2')
ax[0].set_title("\nRegresión logística\ncon Smote\n", fontsize=18)
ax[0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0].set_yticklabels(['', ''], fontsize=14, rotation=360)
sns.heatmap(svc_cf_sm, ax=ax[1], annot=True, cmap=plt.cm.copper, fmt='2')
ax[1].set_title("\nSVM\ncon Smote\n", fontsize=18)
ax[1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1].set_yticklabels(['', ''], fontsize=14, rotation=360)
print('=='*50)

plt.show()
```



Redes neuronales

Ahora se implementa una red neuronal con las tres técnicas de remuestreo antes mencionadas. Se comienza importando la librería *Keras* y algunas herramientas que serán de gran utilidad para llevar a cabo este modelo de forma sencilla. La librería *Keras,* es una *API* de aprendizaje profundo escrita en Python que se ejecuta sobre la plataforma de aprendizaje automático *TensorFlow*, la cual se desarrolló con un enfoque que permita pasar de la idea al resultado lo más rápido posible.

[43]:
```
import keras
from keras import backend as K
from keras.models import Sequential
from keras.layers import Activation
from keras.layers.core import Dense
from keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
from tensorflow.keras import layers
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
```

Submuestreo aleatorio

Para hacer el ajuste del modelo se comenzará con el submuestreo aleatorio. Para esto, se ocupa el conjunto de datos submuestreado anteriormente en el código [12] y separado bajo la misma proporción del código [14], donde X_train (80 %) es para la fase de entrenamiento y X_test (20 %) para etapa de prueba. Para crear la estructura de la red neuronal se utiliza la función *Sequential(),* la cual establece la cantidad de capas con sus respectivas funciones de activación que, en este caso, corresponden a la función *Relu* descrita en la Sección 2.1 y para la capa de salida la función *Softmax* descrita en la Sección 2.3.2.

[44]:
```
n_inputs = X_train.shape[1]
undersample_model = Sequential([ Dense(n_inputs, input_shape=(n_inputs, ),␣
↪activation='relu'),
   Dense(32, activation='relu'),
   Dense(2, activation='softmax')])
```

La librería *Keras* proporciona una herramienta útil para resumir la estructura del modelo, esto lo hace mediante la función *summary(),* la cual entrega información de como está estructurada la red neuronal devolviendo el número de capas, el número de entradas, el número de salidas y el total de parámetros (pesos).

[45]:
```
undersample_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 30)                930
_________________________________________________________________
dense_1 (Dense)              (None, 32)                992
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 66
=================================================================
Total params: 1,988
Trainable params: 1,988
Non-trainable params: 0
_________________________________________________________________
```

La función *compile(),* utilizada en el código [46], es una herramienta de *Keras* que configura el modelo para su posterior entrenamiento y es aquí donde se escoge la función de pérdida que mejor se ajuste. En este caso, la función de pérdida que mejor se ajustó a los datos es la entropía cruzada. Por lo tanto, se utiliza la opción $loss = "sparse_categorical_crossentropy"$de la función *compile()* (se utiliza sparse y categorical por la estructura de la salida de la red). Por otra parte, para optimizar los parámetros el método del descenso de gradiente estocástico, basado en la *estimación adaptativa de momentos*, es una de las técnicas que actualmente más se ocupan para optimizar los parámetros, ya que su hiperparámetro flotante acelera el descenso del gradiente y amortigua las oscilaciones. Esta técnica recibe el nombre de *adam* y, según Kingma et al., 2014, el método es computacionalmente eficiente, tiene pocos requisitos de memoria, invariante al reajuste diagonal de gradientes y es muy adecuado para problemas que son grandes en términos de datos y parámetros. Para llevar a cabo esto, se ocupa la opción $Adam(learning\_rate = 0{,}001)$ con métrica de desempeño (en el entrenamiento) *metrics=accuracy.*

[46]:
```
undersample_model.compile(Adam(learning_rate=0.001),␣
↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Para ajustar el modelo se utiliza la función *fit()* y es en esta etapa donde se entrena el modelo para un número fijo de *epoch,* las cuales se deben especificar. A continuación, se consideran las siguientes opciones para llevar a cabo este ajuste:

Validation_split: Es la fracción de los datos del entrenamiento que se utilizarán como validación. En este caso se elige un 20 % para mantener la misma proporción realizada con la SVM y la regresión logística.

Batch_size: Es el número de muestras por cada actualización del gradiente. Si no se especifica, el valor predeterminado será 32. Si el número de bacht es pequeño, el gasto computacional puede ser significativo (dependiendo el número de datos) lo que aumenta el tiempo de entrenamiento y esto no necesariamente se traduce en un mejor rendimiento. Como se está trabajando con el submuestreo aleatorio se elige el batch_size=10.

Callbacks: Esta función puede realizar acciones en varias etapas del entrenamiento, por ejemplo, al comienzo o al final de una *epoch*, o bien antes o después de un solo batch. Esta opción permite aplicar las siguientes técnicas:

Early Stopping: Recordar que esta técnica detiene el entrenamiento cuando una métrica monitoreada haya dejado de mejorar. Un ciclo *model.fit()* de entrenamiento verifica al final de cada *epoch* si la pérdida ya no está disminuyendo, para esto se considera *min_ delta* (cambio mínimo en la cantidad monitoreada para calificar como una mejora) y *patience* (número de *epoch* sin mejora después de las cuales se detendrá el entrenamiento) si corresponde. Una vez que ya no se ve alguna disminución, *model.stop_ training* se marca como verdadero y el entrenamiento termina.

ReducirLROnPlateau: Reduce la tasa de aprendizaje cuando una métrica ha dejado de mejorar. Los modelos, a menudo, se benefician de la reducción de esta tasa una vez que el aprendizaje se estanca. Esta función monitorea el entrenamiento y si no se ve ninguna mejora en un número de *epoch* de *patience* (5 en este caso), la tasa de aprendizaje se reduce.

En el código [47] se ajustan los datos con las descripciones recién mencionadas.

[47]:
```
history=undersample_model.fit(X_train, y_train, validation_split=0.2,␣
↪batch_size=10, epochs=20, shuffle=True, verbose=2
                              ,callbacks=[ReduceLROnPlateau(patience=3,␣
↪verbose=1, min_lr=1e-6),
                                EarlyStopping(patience=5, verbose=1)])
```

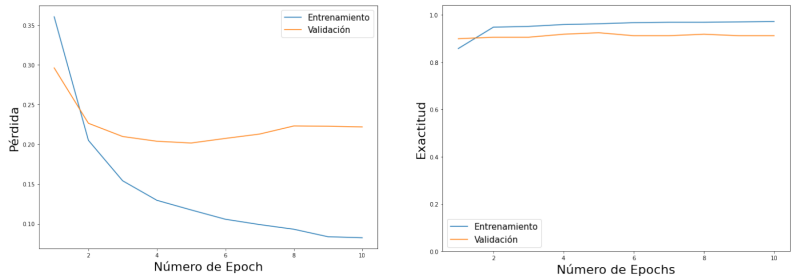
```
Epoch 1/20
63/63 - 0s - loss: 0.3606 - accuracy: 0.8571 - val_loss: 0.2963 - val_accuracy:
0.8987
Epoch 2/20
63/63 - 0s - loss: 0.2050 - accuracy: 0.9476 - val_loss: 0.2265 - val_accuracy:
0.9051
Epoch 3/20
63/63 - 0s - loss: 0.1541 - accuracy: 0.9508 - val_loss: 0.2099 - val_accuracy:
0.9051
Epoch 4/20
63/63 - 0s - loss: 0.1295 - accuracy: 0.9587 - val_loss: 0.2039 - val_accuracy:
0.9177
Epoch 5/20
63/63 - 0s - loss: 0.1174 - accuracy: 0.9619 - val_loss: 0.2017 - val_accuracy:
0.9241
Epoch 6/20
63/63 - 0s - loss: 0.1057 - accuracy: 0.9667 - val_loss: 0.2075 - val_accuracy:
0.9114
Epoch 7/20
63/63 - 0s - loss: 0.0990 - accuracy: 0.9683 - val_loss: 0.2130 - val_accuracy:
0.9114
Epoch 8/20
```

```
Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
63/63 - 0s - loss: 0.0931 - accuracy: 0.9683 - val_loss: 0.2231 - val_accuracy:
0.9177
Epoch 9/20
63/63 - 0s - loss: 0.0837 - accuracy: 0.9698 - val_loss: 0.2228 - val_accuracy:
0.9114
Epoch 10/20
63/63 - 0s - loss: 0.0824 - accuracy: 0.9714 - val_loss: 0.2219 - val_accuracy:
0.9114
Epoch 00010: early stopping
```

Se puede observar que en el octavo *epoch* se activa *ReducirLROnPlateau()* reduciendo la tasa de aprendizaje y en el décimo *epoch,* cuando ya no se visualizó alguna mejora en la métrica de monitoreo, se activa *EarlyStopping()*.

Para explorar visualmente el entrenamiento se procede a graficar, en el código [48], la pérdida versus la exactitud de cada *epochs*. Esto permite ver más claramente si es posible entrenar el modelo por más tiempo (si la pérdida en la validación aún está disminuyendo), o si hubo un ajuste excesivo del modelo en los datos de entrenamiento. Recordar que el sobreajuste es cuando el modelo se desempeña muy bien en un conjunto de entrenamiento, pero terriblemente en los conjuntos de prueba.

[48]:
```
num_epoch = len(history.history["loss"])
fig, axarr = plt.subplots(1, 2, figsize=(24, 8))
axarr[0].set_xlabel("Número de Epoch",fontsize=22)
axarr[0].set_ylabel("Pérdida",fontsize=22)
ax=sns.lineplot(x=range(1, num_epoch+1), y=history.history["loss"],␣
 ↪label="Entrenamiento", ax=axarr[0])
sns.lineplot(x=range(1, num_epoch+1), y=history.history["val_loss"],␣
 ↪label="Validación", ax=axarr[0])
ax.legend(fontsize='15', loc='best')
axarr[1].set_xlabel("Número de Epochs",fontsize=22)
axarr[1].set_ylabel("Exactitud",fontsize=22)
axarr[1].set_ylim(0, 1.04)
sns.lineplot(x=range(1, num_epoch+1), y=history.history["accuracy"],␣
 ↪label="Entrenamiento", ax=axarr[1])
sns.lineplot(x=range(1, num_epoch+1), y=history.history["val_accuracy"],␣
 ↪label="Validación", ax=axarr[1])
plt.legend(fontsize='15')
```



Se puede visualizar que a la izquierda de la salida del código [48] la pérdida por validación empieza a subir desde el quinto *epoch,* mientras que la pérdida en el entrenamiento continúa bajando. Por otra parte, la

exactitud mediante validación comienza a subir levemente alcanzando su máximo en el quinto *epoch* para después mantenerse alrededor de 91,5 %, sin embargo, la exactitud en el entrenamiento continúa subiendo hasta llegar al 97 % en el décimo *epoch.* Es aquí donde nace la necesidad de realizar técnicas, como Early Stopping, que ayuden al modelo a prevenir el sobreajuste de los datos.

Conjunto de prueba

La salida del código [49] muestra la exactitud alcanzada en el conjunto de prueba, la cual llega a un 93,37 %, siendo menor que el alcanzado por la SVM con la técnica SMOTE.

[49]:
```
scores = undersample_model.evaluate(X_test, y_test)
print('==='*41)
print('Pérdida:',scores[0],'\nExactitud:{:.2%}'.format(scores[1]))
print('==='*41)
```

```
7/7 [==============================] - 0s 620us/step - loss: 0.1521 - accuracy:
0.9337
================================================================================
Pérdida: 0.152128204703331
Exactitud:93.37%
```

Ahora se utiliza la función *classification_report()* para obtener las métricas de desempeño anteriormente utilizadas. Se puede apreciar en la salida del código [50] que el rendimiento de la red neuronal utilizando el submuestreo aleatorio no supera el desempeño de los modelos antes ajustados, ya que esta alcanza una menor puntuación en cada una de sus métricas.

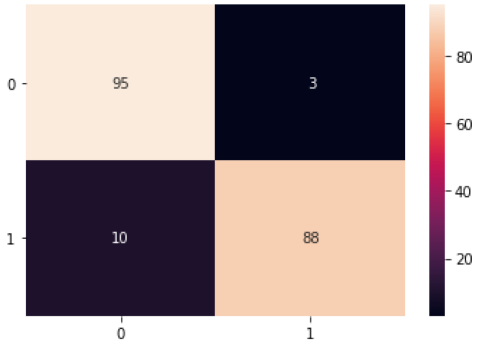
[50]:
```
print('=' * 64)
labels = ['No Fraude', 'Fraude']
print( ' '*7, 'Report con Submuestreo Aleatorio (Redes Neuronales)')
print('=' * 64)
print(classification_report(y_test, undersample_fraud_predictions,␣
 ↪target_names=labels))
```

```
================================================================
        Report con Submuestreo Aleatorio (Redes Neuronales)
================================================================
              precision    recall  f1-score   support

   No Fraude       0.90      0.97      0.94        98
      Fraude       0.97      0.90      0.93        98

    accuracy                           0.93       196
   macro avg       0.94      0.93      0.93       196
weighted avg       0.94      0.93      0.93       196
```

Al realizar la matriz de confusión en el código [51], se observa que la red neuronal obtuvo 13 instancias mal clasificadas de un total de 196, de las cuales 10 corresponden a falsos negativos y 3 a falsos positivos

lo que resulta alto si se compara con la SVM aplicando SMOTE.

[51]:
```
confusion = pd.DataFrame(confusion_matrix(y_test, undersample_fraud_predictions))
confusion.index = ["0", "1"]
confusion.columns = ["0", "1"]
sns.heatmap(confusion, annot=True, fmt='2')
plt.yticks(rotation=0)
```



Conjunto de prueba con near miss

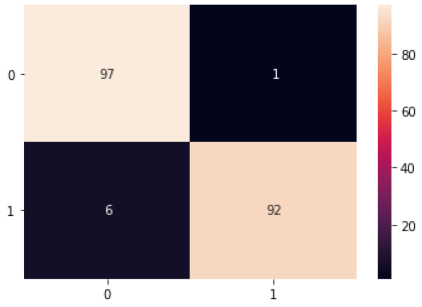
Al igual que en el submuestreo aleatorio, near miss con la red neuronal obtuvo un desempeño inferior a la SVM mediante SMOTE, alcanzando un 96 % de recall, 97 % de precisión y una exactitud del 96 % versus el 98 % alcanzado por la SVM bajo estas mismas métricas

[52]:
```
print('=' * 58)
labels = ['No Fraude', 'Fraude']
print( ' '*14, 'Near Miss (Redes Neuronales)')
print('=' * 58)
print(classification_report(y_test_al, Nearsample_fraud, target_names=labels))
```

```
==========================================================
               Near Miss (Redes Neuronales)
==========================================================
              precision    recall  f1-score   support

   No Fraude       0.94      0.99      0.97        98
      Fraude       0.99      0.94      0.96        98

    accuracy                           0.96       196
   macro avg       0.97      0.96      0.96       196
weighted avg       0.97      0.96      0.96       196
```

Near miss con redes neuronales supera levemente al muestreo aleatorio realizado por la regresión logística y la SVM. Más aún, incurrió en un FN lo que es favorable para este tipo de problemas.

[53]:
```
confusion = pd.DataFrame(confusion_matrix(y_test_al, Nearsample_fraud))
confusion.index = ["0", "1"]
confusion.columns = ["0", "1"]
sns.heatmap(confusion, annot=True, fmt='2')
plt.yticks(rotation=0)
```



Conjunto de prueba con SMOTE

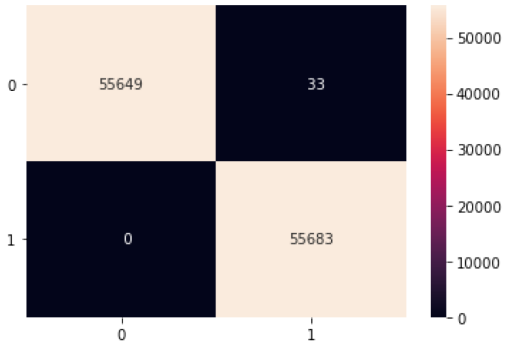
Por último, como se puede observar en la salida del código [54], la red neuronal aplicando un balanceo de clases mediante la técnica SMOTE alcanzo el mejor rendimiento comparando todas las métricas de desempeño antes evaluadas. Esto significa que este clasificador está detectando las transacciones fraudulentas y genuinas de manera muy efectiva superando los modelos de regresión logística y SVM bajo las técnicas de near miss como submuestreo aleatorio.

[54]:
```
print('=' * 55)
print(' '*15,'Smote (Redes Neuronales)')
print('=' * 55)
print(classification_report(submuestra_ytest, oversample_fraud_predictions,␣
 ↪target_names=labels))
print('=' * 55)
```

```
=====================================================
                Smote (Redes Neuronales)
=====================================================
              precision    recall  f1-score   support

   No Fraude       1.00      1.00      1.00     55682
      Fraude       1.00      1.00      1.00     55683

    accuracy                           1.00    111365
   macro avg       1.00      1.00      1.00    111365
weighted avg       1.00      1.00      1.00    111365
```

Por último, en la salida del código [55] mediante la matriz de confusión se observa que el algoritmo obtuvo cero falsos negativos, es decir, de un total de 55,683 transacciones fraudulentas el modelo clasificó todas correctamente, lo que es excelente para una entidad financiera, ya que se disminuye las pérdidas mediante fraude. Por otra parte, los falsos positivos solo fueron 33 de 55,682, lo que representa una alta capacidad de detectar las transacciones genuinas. Por lo tanto, el mejor clasificador es la red neuronal con la técnica SMOTE.

[55]:
```
confusion_smt= pd.DataFrame(confusion_matrix(submuestra_ytest,␣
 ↪oversample_fraud_predictions))
confusion.index = ["Real Negativa", "Real Positiva"]
confusion.columns = ["Predicción Negativa", "Predicción Positiva"]
sns.heatmap(confusion_smt, annot=True, fmt='4')
plt.yticks(rotation=0)
```



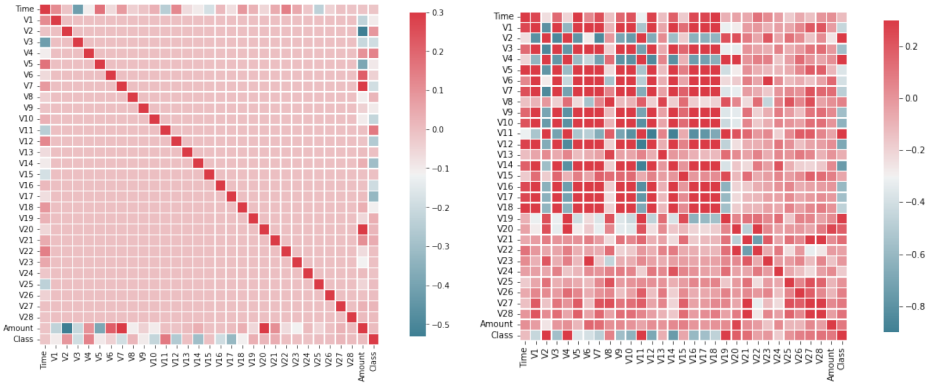
Observación 64 *Para ajustar la red mediante SMOTE, se utilizó Batch_ size*= 300*, ya que la cantidad de datos aumentó de* 278,413 *transacciones a* 556,826*, es decir, se crearon* 278,413 *datos sintéticos. Es por esta razón, que se recomienda aumentar el tamaño de los batch y así disminuir el tiempo de entrenamiento.*

Capítulo 5

Valores extremos para la Clasificación de Operaciones Bancarias Fraudulentas

En la Sección 4.2.2 primero se realiza un balanceo de clases y, posteriormente se separa el conjunto de datos en entrenamiento-test En esta sección, se muestra el problema de hacer esto y se propone un método de submuestreo que toma en cuenta los valores extremos a través de la distancia de Mahalanobis. Los datos, llamados *credit card*, han sido previamente procesados a través de una transformación de PCA y sus características tienen una correlación muy cercana a cero (ver Figura 120a). Sin embargo, al realizar el balanceo de clases se genera un cierto grado de correlación entre las características. La Figura 120 muestra la correlación generada después de aplicar un submuestreo aleatorio, pero este patrón se repite

en los demas casos de muestreo. Como resultado, el rendimiento de los modelos tiende a ser sobrestimado.



(a) (b)

Figura 120. Correlaciónentre las caracteristicas del conjunto de datos credit - card antes y después de un submuestreo aleatorio. (a) Datos en bruto; (b) Después de un balanceo de clases.

La metodología utilizada en esta tesis se presenta a continuación y se divide en dos casos. En el primer caso (conjunto de prueba contaminado), se divide el conjunto de datos en conjuntos de entrenamiento y test después de aplicar técnicas de muestreo, lo que produce una intervención directa en el conjunto de prueba, provocando la pérdida de su desbalance y generando una correlación entre los conjuntos de entrenamiento y test. En el segundo caso (conjunto de prueba no contaminado), se divide el conjunto de datos en conjuntos de entrenamiento y test antes de aplicar las técnicas de muestreo descritas en este trabajo, por consiguiente, no se contamina. En ambos casos, se presentan tres enfoques:

CASO: Conjunto de Test correlacionado con el conjunto de entrenamiento.

Enfoque 1. Se aplican las técnicas de muestreo (submuestreo y sobremuestreo) NearMiss, Submuestreo Aleatorio y SMOTE. Este enfoque describe la métodologia realizada en la Sección 4.2.2 y se ilustra en la Figura 121.

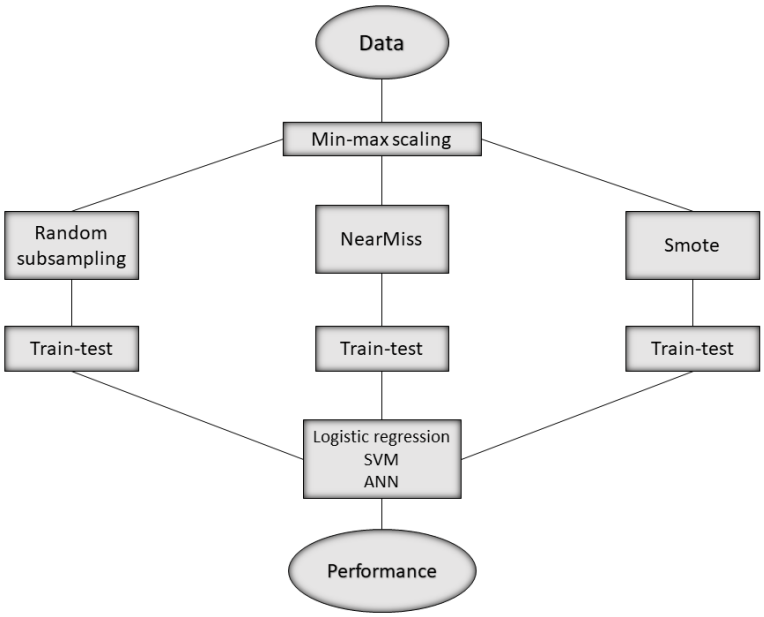


Figura 121. Diagrama de implementación del **enfoque 1** bajo el caso del conjunto de test correlacionado con el conjunto de entrenamiento.

Para llevar a cabo los siguientes enfoques se toma en cuenta la distancia de Mahalanobis. La distancia de Mahalanobis de un vector multivariante $x = (x_1, x_2, \ldots, x_d)^T$ de un grupo de valores con media $\mu = (\mu_1, \mu_2, \ldots, \mu_d)$ y matriz de varianza covarianza S se define como

$$d_M(x) = \sqrt{(x-\mu)^T S^{-1}(x-\mu)}. \tag{5.1}$$

Si la matriz de covarianza es la matriz identidad, la distancia de Mahalanobis se reduce a la distancia euclidiana. Ahora, si $x \sim \mathcal{N}_d(\mu, S)$ con $S > 0$. Entonces,

$$(x-\mu)^T S^{-1}(x-\mu) \sim \chi^2_d \tag{5.2}$$

y

$$\left\{x \in \mathbb{R}^d : (x-\mu)^T S^{-1}(x-\mu) \leq c^2 = \chi^2_{d,\alpha}\right\} \tag{5.3}$$

es un elipsoide de $(1-\alpha)100\,\%$ de confianza, donde $\chi^2_{d,\alpha}$ es el cuantil $1-\alpha$ de la distribución χ^2_d, cuyos ejes están dados por $\pm c\sqrt{\lambda_i e_i}$, $Se_i = \lambda_i$ para $i = 1, \ldots, d$ y el par (λ, e) son los valores y vectores propios de S. La distancia entre dos puntos de datos $x = (x_1, x_2, \ldots, x_d)^T$ e $y = (y_1, y_2, \ldots, y_d)^T$ en un espacio d-dimensional $\mathbb{R}^d$ se define como

$$d_M(x, y) = \sqrt{(x-y)^T S^{-1}(x-y)}. \tag{5.4}$$

Enfoque 2. Consiste en un submuestreo de la clase mayoritaria con el fin de equilibrar las clases respecto a la distancia de Mahalanobis, teniendo en cuenta los valores extremos. Para ello, se calculó la distancia de Mahalanobis de cada uno de los vectores d-dimensionales x_n (con $n = 1, 2, ..., 284{,}807$) respecto al vector de medias μ y a la matriz de varianzas y covarianzas S del conjunto de datos credit card. Luego, se seleccionaron las 492 instancias x_n correspondientes a las distancias de Mahalanobis c_n^2 dada por

$$(x_n - \mu)^T S^{-1}(x_n - \mu) = c_n^2 \tag{5.5}$$

más altas, obteniendo así un conjunto de datos balanceado. La Figura 122 muestra el diagrama de esta metodología.

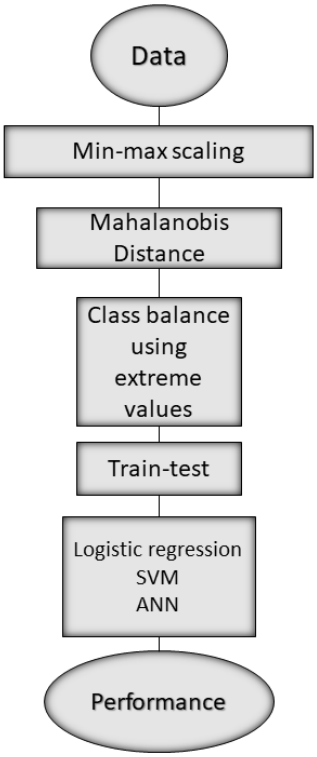


Figura 122. Diagrama de implementación del **enfoque 2** bajo el caso del conjunto de test correlacionado con el conjunto de entrenamiento.

Enfoque 3. Este enfoque consiste en un submuestreo del conjunto de datos completo, en el cual se seleccionan los valores más alejados de ambas clases mediante la distancia de Mahalanobis (ver Figura 123). Esta distancia tiene la ventaja de tener en cuenta la variabilidad multivariada presente en los datos. Al preservar la estructura de covarianza de las instancias de ambas clases y seleccionar los valores extremos a lo largo de los contornos de probabilidad, se logra un modelado más preciso para los algoritmos de aprendizaje. En este enfoque se tomaron 50,000 datos, de los cuales 456 eran fraudulentos.

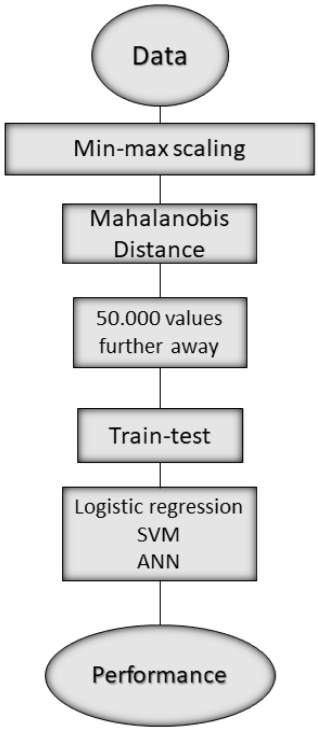


Figura 123. Diagrama de implementación del **enfoque 3** bajo el caso del conjunto de test correlacionado con el conjunto de entrenamiento.

Los resultados del desempeño de los tres enfoques bajo el caso de test correlacionado con el conjunto de entrenamiento se presentan en la Cuadro 4.1. Se observa que la ANN con la técnica SMOTE resultó ser la mejor opción, seguida por la SVM. Es importante considerar que, en este caso, el conjunto de test fue manipulado para ser balanceado a excepción del Enfoque 3, el cual realiza la clasificación sin modificar el desbalance (por lo que obtiene una efectividad del 100 %), sin embargo, también se manipula el conjunto de test, ya que se consideran solo los valores extremos.

Cuadro 5.1: Desempeño del caso de estudio: Primero se balancean las clases o se toman los 50.000 valores más extremos, luego se separan en train-test.

Random Subsampling (Approach 1)				
Models	**Precision** %	**Recall** %	**F1-score** %	**Accuracy** %
Logistic regression	94, 1	93,5	93,5	94, 6
SVM	94, 3	93, 5	93,5	93, 2
ANN	94, 2	93,5	93,5	93, 3
NearMiss (Approach 1)				
Logistic regression	97,5	97,5	97	97, 8
SVM	96,5	96,5	96, 5	96, 8
ANN	97, 3	96,5	96,5	96, 0
Smote (Approach 1)				
Logistic regression	95, 2	95, 2	95, 6	95, 4
Svm	98, 4	98,1	98, 2,	98, 6
ANN	100	100	100	100
Balancing with extreme values (Approach 2)				
Logistic regression	91, 3	91, 4	91,4	91, 7
SVM	97, 5	97,0	97,3	97, 8
ANN	96, 2	96, 4	96, 4	96, 8
50,000 **M;ost extreme (Approach 3)**				
Logistic regression	94, 2	83, 4	88, 4	100
SVM	96, 8	91, 5	93, 3	100
ANN	94, 5	94, 4	94, 4	100

CASO: No correlacionado con el conjunto de entrenamiento.

La metodología seguida en este caso sigue los mismos enfoques que en el caso anterior, pero en lugar de intervenir el conjunto de test, se divide primero el conjunto de datos en entrenamiento y test y luego solo se aplican las técnicas de preprocesamiento al conjunto de entrenamiento sin modificar el conjunto de test. La implementación de esta metodología se ilustra en las siguientes figuras.

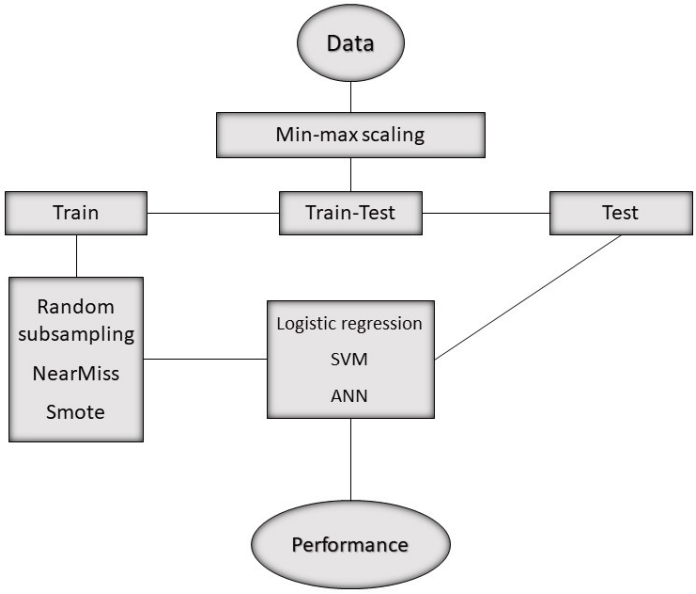


Figura 124. Diagrama de implementación del **enfoque 1** bajo el caso del conjunto de test no correlacionado con el conjunto de entrenamiento.

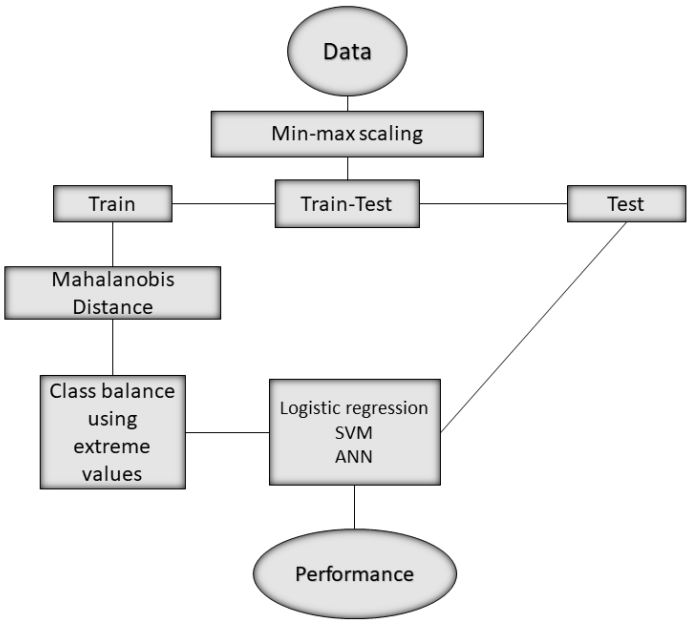


Figura 125. Diagrama de implementación del **enfoque 2** bajo el caso del conjunto de test no contaminado.

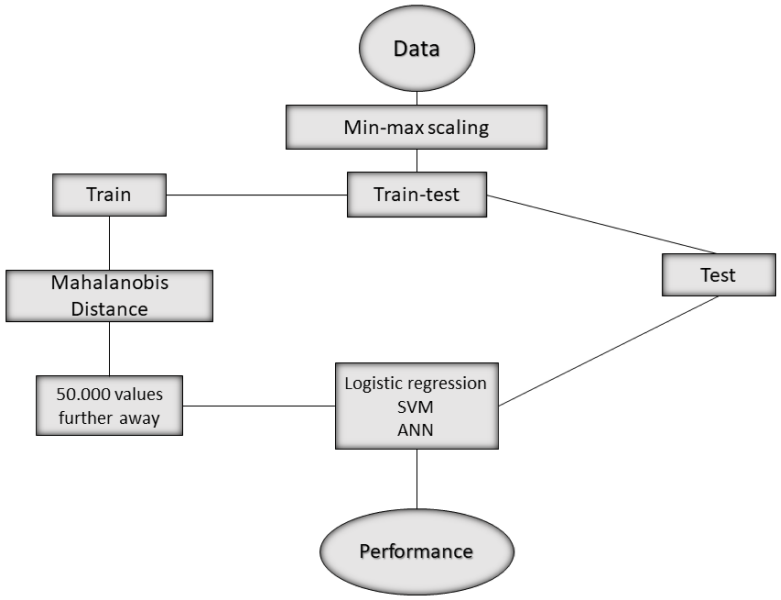


Figura 126. Diagrama de implementación del **enfoque 3** bajo el caso del conjunto de test no correlacionado con el conjunto de entrenamiento.

Los resultados del desempeño de los enfoques bajo el caso de test no contaminado se presentan en el Cuadro 4.2. Al no intervenir el conjunto de test, se obtienen resultados menos sesgados en comparación con el caso anterior debido a que el conjunto de test mantiene su distribución original y, por lo tanto, es más representativo del conjunto de datos completo. Además, se evita la manipulación del conjunto de test, lo que puede llevar a un sobreajuste del modelo y resultados poco confiables. En general, se observa que el Enfoque 3, utilizando los modelos de ANN y SVM, tuvo un desempeño significativamente mejor que las otras técnicas de muestreo, destacando en términos de precisión y f1-score. Sin embargo, tuvo un recall inferior al obtenido mediante la técnica SMOTE y Submuestreo aleatorio. Aun así, en términos generales se obtienen resultados comparables.

Cuadro 5.2: Desempeño del caso de estudio: Primero se separa por train-test, luego se balancean las clases o se alejan los valores de los Primeros 50.000.

Random Subsampling (Approach 1)				
Models	**Precision** %	**Recall** %	**F1-score** %	**Accuracy** %
Logistic regression	50, 5	80, 3	41, 5	68, 0
SVM	50, 4	76, 3	39, 9	62, 6
ANN	52, 5	94, 2	54, 2	97, 1
NearMiss (Approach 1)				
Logistic regression	50, 5	80, 5	41, 8	68, 3
SVM	50, 5	76, 1	39, 0	62, 0
ANN	50, 5	77, 2	40, 2	66,1
Smote (Approach 1)				
Logistic regression	53, 2	96,1	55, 7	98, 7
SVM	55, 3	96, 2	59, 3	98, 1
ANN	76, 3	94, 5	82, 5	100
Balancing with extreme values (Approach 2)				
Logistic regression	50, 5	55, 2	13, 4	15, 0
Svm	50, 0	50, 0	1,2	1, 0
ANN	50, 5	56, 5	14, 5	17, 4
50, 000 **most extreme values (Approach 3)**				
Logistic regression	94, 1	79, 2	85, 2	100
SVM	97, 3	88, 1	92,1	100
ANN	96, 3	88, 2	92, 4	100

Capítulo 6

CONCLUSIONES Y TRABAJOS FUTUROS

En el presente trabajo se propuso un modelo de clasificación para transacciones fraudulentas mediante valores extremos y se comparó los resultados obtenidos entre los modelos de Regresión logística, SVM y ANN. A continuación, se describirán las conclusiones del presente trabajo y algunos de los posibles trabajos futuros que pueden continuar desarrollándose como resultado de la investigación.

6.1. Conclusiones

En esta tesis, se presenta un marco de comparación del efecto que produce aplicar técnicas de submuestreo y sobremuestreo de clases en un conjunto de datos altamente desbalanceado antes y después de realizar un equilibrio de clases. Se puso en manifiesto el cambio en la correlación que esto provoca y la influencia que tiene en el desempeño de los modelos generando una sobreestimación de estos. Nuestros resultados indican que cuando se separa el conjunto de datos en entrenamiento-test y después se aplican las técnicas de submuestreo aleatorio, NearMiss, SMOTE y balanceo de clases con valores extremos (en el conjunto de entrenamiento) el desempeño de los modelos en la precisión y el f1-score disminuye y en algunos casos significativamente. Sin embargo, la técnica de submuestreo propuesto bajo el *Enfoque 3* alcanzó buenos resultados llegando a un 97 % de precisión, un 92 % en el f1-score y un 88 % de recall lo que estaría indicando una buena generalización de la clase minoritaria. Por lo tanto, esta técnica de submuestreo es una opción viable para ser aplicada en problemas de clasificación con características similares.

6.2. Trabajos futuros

Durante el proceso de elaboración de esta tesis, se han identificado algunas líneas de investigación futuras que se han dejado pendientes y que se espera abordar en el futuro. Algunas de estas líneas están directamente relacionadas con el tema de esta tesis y han surgido durante su elaboración. Otras son más generales y, aunque no forman parte de este trabajo de tesis, pueden ser consideradas para futuros

proyectos de investigación o para retomarlas en el futuro. A continuación, se presentan algunos de los posibles trabajos futuros que se podrían realizar.

- En esta tesis, el número de instancias para llevar a cabo el entrenamiento, bajo el *segundo enfoque,* fue determinado mediante ensayo y error, no se plantea un método para escoger el número de instancias óptimas. Sería interesante encontrar un método que determine esta cantidad.
- Cuando se realiza un balanceo de clases con las técnicas expuestas la distribución del conjunto de datos se ve afectada. Se podría construir un método que genere instancias sintéticas de la clase minoritaría sin perturbar su distribución. Análogamente, si se hace un submuestreo de la clase mayoritaria.
- Probar otros modelos de machine learning como redes generativas adversarias, árboles de decisión, XGboost, entre otros.
- El conjunto de datos utilizado para nuestros experimentos tienen atributos numéricos, es decir, reales. En un trabajo futuro se podría estudiar diferentes posibilidades de adaptar nuestra propuesta para manejar también atributos nominales y de tipo mixto. (numérico/nominal).
- Extender la técnica propuesta a problemas multiclases.

Bibliografía

[1] Nilson Report: https://nilsonreport.com/.

[2] Bhattacharyya, S., Jha, S., Tharakunnel, K., Westland, J.C.: Data mining for credit card fraud: A comparative study. Decision support systems 50(3), 602-613(2011).

[3] Brause, R., Langsdorf, T., Hepp, M.: Neural data mining for credit card fraud detection. In: Proceedings 11th International Conference on Tools with Artificial Intelligence, pp. 103-106 (1999). IEEE.

[4] Gómez, J.A., Arévalo, J., Paredes, R., Nin, J.: End-to-end neural network architecture for fraud scoring in card payments. Pattern Recognition Letters 105, 175-181 (2018).

[5] Save, P., Tiwarekar, P., Jain, K.N., Mahyavanshi, N.: A novel idea for credit card fraud detection using decision tree. International Journal of Computer Applications 161(13) (2017).

[6] Sahin, Y., Duman, E.: Detecting credit card fraud by ANN and logistic regression. In: 2011 International Symposium on Innovations in Intelligent Systems and Applications, pp. 315-319 (2011). IEEE.

[7] Dal Pozzolo, A., Boracchi, G., Caelen, O., Alippi, C., Bontempi, G.: Credit card fraud detection: a realistic modeling and a novel learning strategy. IEEE transactions on neural networks and learning systems 29(8), 3784–3797 (2017).

[8] He, H., Bai, Y., Garcia, E.A., Li, S.: Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In: 2008 IEEE International Joint Conference on Neural Networks (IEEEWorld Congress on Computational Intelligence), pp. 1322–1328 (2008). IEEE

[9] Pazzani, M., Merz, C., Murphy, P., Ali, K., Hume, T., Brunk, C.: Reducing misclassification costs, 217–225 (1994).

[10] Sahin, Y., Bulkan, S., Duman, E.: A cost-sensitive decision tree approach for fraud detection. Expert Systems with Applications 40(15), 5916–5923 (2013).

[11] Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE: synthetic minority oversampling technique. Journal of artificial intelligence research 16, 321–357 (2002).

[12] Nguyen, H.M., Cooper, E.W., Kamei, K.: Borderline over-sampling for imbalanced data classification. International Journal of Knowledge Engineering and Soft Data Paradigms 3(1), 4–21 (2011).

[13] Abdi, L., Hashemi, S.: To combat multi-class imbalanced problems by means of over-sampling techniques. IEEE transactions on Knowledge and Data Engineering 28(1), 238–251 (2015).

[14] Kubat, M., Matwin, S., et al.: Addressing the curse of imbalanced training sets: one-sided selection. In: Icml, vol. 97, p. 179 (1997). Nashville, USA.

[15] Mani, I., Zhang, I.: knn approach to unbalanced data distributions: a casestudy involving information extraction. In: Proceedings of Workshop on Learning from Imbalanced Datasets, vol. 126, pp. 1–7 (2003). ICML.

[16] Bishop, C. M., & Nasrabadi, N. M. (2006). Pattern recognition and machine learning (Vol. 4, No. 4, p. 738). New York: springer.

[17] Yi, X., Xu, Y., Hu, Q., Krishnamoorthy, S., Li, W., & Tang, Z. (2022). ASN-SMOTE: a synthetic minority oversampling method with adaptive qualified synthesizer selection. Complex & Intelligent Systems, 1-26.

[18] Tao, Y., Zhang, Y., & Jiang, B. (2020). DBCSMOTE: A clustering-based oversampling technique for data-imbalanced warfarin dose prediction. BMC medical genomics, 13(10), 1-13.

[19] Sanguanmak, Y., & Hanskunatai, A. (2016, July). DBSM: The combination of DBSCAN and SMOTE for imbalanced data classification. In 2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE) (pp. 1-5). IEEE.so IEEE , 9 , 74763-74777.

[20] Pradipta, G. A., Wardoyo, R., Musdholifah, A., & Sanjaya, I. N. H. (2021). Radius-SMOTE: a new oversampling technique of minority samples based on radius distance for learning from imbalanced data. IEEE Access, 9, 74763-74777.

[21] Laura, I., & Santi, S. (2017). Introduction to Data Science: A Python Approach to Concepts, Techniques and Applications.

[22] Boser, B.E., Guyon, I.M. & Vapnik, V.N. A training algorithm for optimal margin classifiers. in 5th ANNual ACM Workshop on COLT (ed. Haussler, D.) 144–152 (ACM Press, Pittsburgh, PA, 1992).

[23] Noble, W.S. Support vector machine applications in computational biology. in Kernel Methods in Computational Biology (eds. Schoelkopf, B., Tsuda, K. & Vert, J.-P.) 71–92 (MIT Press, Cambridge, MA, 2004).

[24] Vapnik, V. & Lerner, A. Pattern recognition using generalized portrait method. Autom. Remote Control 24, 774–780, 1963.

[25] Sundar, R., & Punniyamoorthy, M. (2019). Performance enhanced Boosted SVM for Imbalanced datasets. Applied Soft Computing, 83, 105601.

[26] Salas, R., Moreno, S., Allende, H., & Moraga, C. (2007). A robust and flexible model of hierarchical self-organizing maps for non-stationary environments. Neurocomputing, 70(16-18), 2744-2757.

[27] Viloria, A., Lezama, O. B. P., & Mercado-Caruzo, N. (2020). Unbalanced data processing using oversampling: Machine learning. Procedia Computer Science, 175, 108-113.

[28] Sain, H., & Purnami, S. W. (2015). Combine sampling support vector machine for imbalanced data classification. Procedia Computer Science, 72, 59-66.

[29] Varmedja, D., Karanovic, M., Sladojevic, S., Arsenovic, M., & Anderla, A. (2019, March). Credit card fraud detection-machine learning methods. In 2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH) (pp. 1-5). IEEE.

[30] Sahayasakila, V., Aishwaryasikhakolli, D., & Yasaswi, V. (2019). Credit card fraud detection system using SMOTE technique and whale optimization algorithm. Int. J. Eng. Adv. Technol.(IJEAT), 8(5), 190-192.

[31] Nguyen, H. M., Cooper, E. W., & Kamei, K. (2011). Borderline over-sampling for imbalanced data classification. International Journal of Knowledge Engineering and Soft Data Paradigms, 3(1), 4-21.

[32] Mansourifar, H., & Shi, W. (2020). Deep synthetic minority over-sampling technique. arXiv preprint arXiv:2003.09788.

[33] Batista, G. E., Prati, R. C., & Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD explorations newsletter, 6(1), 20-29.

[34] Awoyemi, J. O., Adetunmbi, A. O., & Oluwadare, S. A. (2017, October). Credit card fraud detection using machine learning techniques: A comparative analysis. In 2017 international conference on computing networking and informatics (ICCNI) (pp. 1-9). IEEE.

[35] Li, Z., Liu, G., & Jiang, C. (2020). Deep representation learning with full center loss for credit card fraud detection. IEEE Transactions on Computational Social Systems, 7(2), 569-579.

[36] Kumar, P., & Iqbal, F. (2019, April). Credit card fraud identification using machine learning approaches. In 2019 1st International conference on innovations in information and communication technology (ICIICT) (pp. 1-4). IEEE.

[37] Jain, Y., Tiwari, N., Dubey, S., & Jain, S. (2019). A comparative analysis of various credit card fraud detection techniques. Int J Recent Technol Eng, 7(5S2), 402-407.

[38] Riffi, J., Mahraz, M. A., El Yahyaouy, A., & Tairi, H. (2020, June). Credit card fraud detection based on multilayer perceptron and extreme learning machine architectures. In 2020 International Conference on Intelligent Systems and Computer Vision (ISCV) (pp. 1-5). IEEE.

[39] Zhang, X., Han, Y., Xu, W., & Wang, Q. (2021). HOBA: A novel feature engineering methodology for credit card fraud detection with a deep learning architecture. Information Sciences, 557, 302-316.

[40] Chen, Y. J., Liou, W. C., Chen, Y. M., & Wu, J. H. (2019). Fraud detection for financial statements of business groups. International Journal of Accounting Information Systems, 32, 1-23.

[41] Salas, R., Allende, H., Moreno, S., & Saavedra, C. (2005, November). Flexible architecture of self organizing maps for changing environments. In Iberoamerican Congress on Pattern Recognition (pp. 642-653). Springer, Berlin, Heidelberg.

[42] Salas, R. (2008). Lógica Difusa. Revista de Información, Tecnología y Sociedad, 122.

[43] Salas, R., Saavedra, C., Allende, H., & Moraga, C. (2011). Machine fusion to enhance the topology preservation of vector quantization artificial neural networks. Pattern recognition letters, 32(7), 962-972.

[44] Salas, R. (2004). Redes Neuronales Artificiales [en línea]. Valparaíso (Chile), 4.

[45] Carlos Sarraute (2007). Aplicación de las Redes Neuronales al Reconocimiento de Sistemas Operativos.

[46] Hertz, J., Krogh, A., & Palmer, R. G. (2018). Introduction to the theory of neural computation. CRC Press.

[47] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.

[48] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536.

[49] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems, 2(4), 303-3