Cursos de Máster "MD008_Inteligencia Artificial para Ciencia de Datos y MD009_Herramientas avanzadas de análisis de datos" 2023

Detección de Fraude en Tarjetas de Crédito

Xiang Liu
Mabel Mires
Natalia Benitez

Abstract. *La clasificación y predicción de fraude de tarjetas de crédito es un problema de gran criticidad para las instituciones financieras y los consumidores. La detección temprana del fraude puede prevenir pérdidas económicas y proteger la integridad del sistema financiero. En este contexto, este proyecto implementa técnicas de aprendizaje supervisado para identificar patrones y comportamientos sospechosos en transacciones anonimizadas. Se intenta superar el desafío de un dataset muy desequilibrado, donde la clase de transacciones fraudulentas es significativamente menor que la clase de transacciones legítimas, y para ello, se implementan métodos de remuestreo logrando mejores resultados en la métrica que identifica los valores positivos de fraude en el conjunto de datos.*

Palabras Clave. *Detección de Fraude. Oversampling. Undersampling, Aprendizaje Supervisado.*

1. Introducción

Este proyecto surge del interés por llevar a cabo experimentos con conjuntos de datos del sector financiero. El Machine Learning (ML) y la Inteligencia Artificial (IA) en la actualidad forman parte de numerosos procesos que desarrollamos a diario. En la lucha contra el fraude bancario tienen un rol muy especial debido a que esta tecnología puede marcar la diferencia en la efectividad de la detección y la prevención de estos delitos.

Las posibilidades de construir soluciones efectivas incrementan con el desarrollo de modelos de los diferentes tipos de tipos de ML. Estos sistemas especialmente construidos para aprender a reconocer datos que puedan ser sospechosos permiten detectar con eficiencia y eficacia procesos fraudulentos. Gracias a ellos las entidades bancarias tienen la posibilidad de automatizar tareas rutinarias de seguridad mediante modelos de IA.

El aprendizaje automatizado de tipo supervisado y no supervisado junto a las redes neuronales tiene un rol fundamental en la detección automática de fraudes permitiendo construir ecosistemas bancarios cada vez más seguros.

Por lo que nuestro dominio escogido es la detección de fraude en tarjetas de crédito bancarias. Siendo un tema crítico y cada vez más importante en la industria financiera, con el aumento del comercio electrónico y la facilidad de realizar transacciones en línea, también ha aumentado el riesgo de fraude en las tarjetas de crédito.

El objetivo principal de este proyecto es diseñar, implementar y comparar modelos de machine learning capaces de clasificar y predecir fraudes en tarjetas de crédito.

2.1 Marco de Trabajo

El siguiente diagrama proporciona una descripción general de nuestro marco de trabajo (Figura 1).

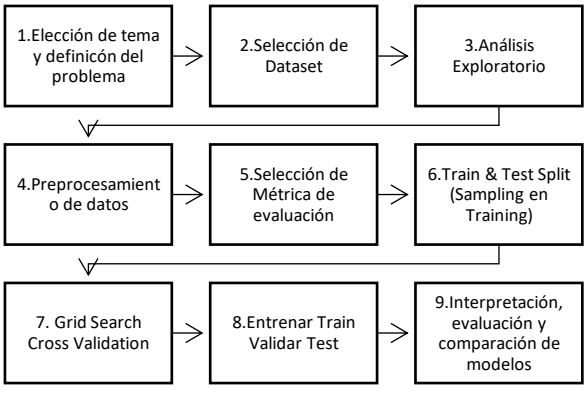

Ilustración 1. Marco de trabajo adoptado

A fin de desarrollar un modelo para clasificar y predecir fraude, se completan las siguientes etapas, de una forma iterativa para los distintos modelos:

1. Definición del problema: En esta etapa, se define el tema y el objetivo del modelo de detección de fraude.

2. Selección de Dataset: Selección del conjunto de datos de entrada necesarios para el modelo.

3. Análisis Exploratorio: Con el objetivo de tener una visión general y a la vez detallada de los datos, comprendiendo mejor la información y las relaciones entre las variables.

4. Preprocesamiento de los datos: En esta etapa, se escalan los datos. Hay algoritmos muy sensibles al escalado de los datos, ya que algunos se basan en la distancia entre los puntos de datos para determinar la similitud entre ellos. Si los datos no están escalados, las características con valores más altos dominarán las características con valores más bajos y, por lo tanto, las características con valores más bajos podrían no tener ningún impacto en la predicción del modelo.

5. Selección de Métrica de evaluación: Selección de una métrica en común para medir el desempeño de los modelos y poder compararlos.

6. Train & Test Split (Sampling en Training): Los datos se dividen en conjuntos de entrenamiento y prueba. El conjunto de entrenamiento se utiliza para entrenar el modelo y el conjunto de prueba se utiliza para evaluar el rendimiento del modelo. A su vez, probar los modelos con técnicas de Sampling para corregir desequilibrio de clase solo con el conjunto de entrenamiento.

7. Grid Search Cross Validation (o búsqueda en cuadrícula con validación cruzada) es una técnica utilizada para encontrar los mejores hiperparámetros de un modelo de aprendizaje automático. El objetivo es seleccionar los valores óptimos de los hiperparámetros que maximizan el rendimiento del modelo en los datos de prueba. Para hacer esto, se utilizan técnicas de validación cruzada, que permiten evaluar el rendimiento del modelo en diferentes divisiones de los datos de entrenamiento y prueba.

8. Entrenar Train - Validar Test: En esta fase, se entrenan los modelos de aprendizaje automático, como árboles de decisión, redes neuronales, SVM, etc. Se ajustan hiperparámetros encontrados con Grid Search. Se realiza una validación con el conjunto de test. Es importante evaluar diferentes algoritmos para determinar cuál tiene el mejor desempeño.

9. Interpretación, evaluación y comparación de modelos: Finalmente, se evalúan los modelos de aprendizaje automático de acuerdo con las métricas seleccionadas y se escoge el modelo con el mejor rendimiento en la métrica de evaluación definida.

2.2 Estructura del artículo

En esta sección se resume lo que se espera encontrar en este proyecto:

Introducción: Contexto y justificación del problema, Objetivos del trabajo.

Tecnologías utilizadas: Descripción de los métodos y modelos utilizados en el desarrollo del trabajo.

Desarrollo de la propuesta: Descripción detallada del enfoque propuesto para la detección de fraude de tarjetas de crédito utilizando los modelos de machine learning.

Experimentación / Resultados obtenidos: Explicación de los pasos y decisiones tomadas en el proceso de desarrollo de los modelos. Descripción de los experimentos realizados para evaluar el rendimiento del modelo propuesto.

Conclusiones: Discusión de los resultados obtenidos y de las limitaciones del modelo propuesto. Resumen de los resultados y conclusiones del trabajo.

Líneas abiertas – Trabajo Futuro: Análisis de posibles mejoras o ampliaciones futuras del modelo. Propuestas para líneas futuras de investigación.

3. Tecnologías usadas

El mayor énfasis en el proceso de detección de fraudes en estados financieros ha crecido en esta era digital en la utilización de datos técnicas de minería, demostrando ser bastante efectivas no solo para detectar fraudes en estados financieros sino también para descubrir otros delitos financieros que pueden involucrar fraudes relacionados con cheques falsos o sin fondos, fraudes con tarjetas de crédito, fraudes de préstamos y valores, fraudes corporativos, fraudes bancarios y de seguros, etcétera. Es factible detectar patrones subyacentes desconocidos a partir de los datos y predecir tendencias y comportamientos futuros mediante el uso de herramientas y técnicas de minería de datos. Se pueden usar diferentes tipos de técnicas de minería de datos para descubrir fraudes financieros. **[1]** De acuerdo con lo estudiado en el Máster y en el estudio de la literatura, las

técnicas de minería de datos aplicadas para este trabajo fueron:

3.1 Técnicas para Resolver Clase Desequilibrada

El desequilibrio de clases es un problema común en el aprendizaje automático, especialmente en los problemas de clasificación. Los datos de desequilibrio pueden dificultar mucho la precisión de un modelo ya que se obtiene una precisión bastante alta simplemente prediciendo la clase mayoritaria, pero no se logra capturar la clase minoritaria, que suele ser el objetivo de la creación del modelo en primer lugar.

En nuestro dataset nos encontramos con la variable Class con un 0.17% de casos de fraude versus un 99.83% de no fraude.

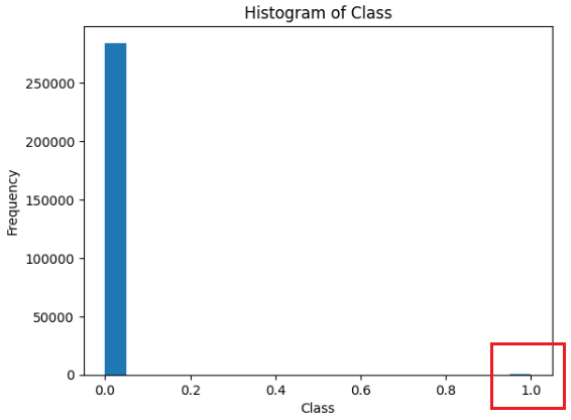

Ilustración 2. Variable Clase totalmente desequilibrada

Una de las técnicas de desequilibrio de clases ampliamente adoptadas para manejar conjuntos de datos altamente desequilibrados se denomina remuestreo. Consiste en quitar muestras de la clase mayoritaria (under-sampling) y/o añadir más muestras de la clase minoritaria (over-sampling). (Figura 3) [2]

En nuestro caso, estas técnicas fueron aplicadas con la librería de Pyhton: imblearn. Imbalanced-learn (importado como imblearn) es una biblioteca de código abierto con licencia del MIT que se basa en scikit-learn (importado como sklearn) y proporciona herramientas cuando se trata de la clasificación con clases desequilibradas. [3]

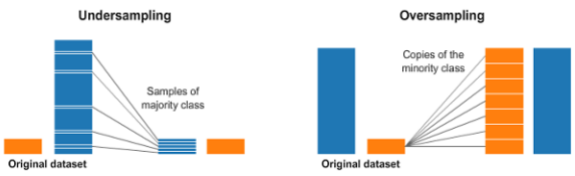

Ilustración 3. Técnicas de Remuestreo: Undersampling y Oversampling

3.1.1 Random Under-Sampling con librería Imblearn

Aplicamos RandomUnderSampler para equilibrar nuestros datos. Este submuestreo aleatorio implica seleccionar aleatoriamente ejemplos de la clase mayoritaria y eliminarlos del conjunto de datos de entrenamiento. La desventaja que nos encontramos es que puede resultar en la pérdida de información invaluable para un modelo.

```
# Se aplica undersampling solo al dataset de training
undersampler = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled = undersampler.fit_resample(X_train, y_train)
```

Ilustración 4. Función Random Undersampling aplicada

3.1.2 Random Over-Sampling con librería Imblearn

Otra forma de combatir los datos desequilibrados es generando nuevas muestras en la clase minoritaria. La estrategia que usamos es RandomOversampling para duplicar algunas de las muestras originales de la clase minoritaria mediante muestreo aleatorio con el reemplazo de las actualmente disponibles.

```
# Se aplica oversampling solo al dataset de training
oversampler = RandomOverSampler(random_state=42)
X_train_resampled, y_train_resampled = oversampler.fit_resample(X_train, y_train)
```

Ilustración 5. Función Random Oversampling aplicada

3.2 Algoritmos implementados

Al ser un problema de clasificación y predicción binaria, se optó por empezar por algoritmos simples y robustos como KNN, SVM y Regresión Logística. Al notar buenos resultados en cuanto a recall pero no en cuanto a la métrica de precisión, se experimenta con modelos de Redes Neuronales. Si bien, se obtuvo una mejora en el valor de la precisión y recall, aún había margen para mejora de dichas métricas. Por lo que se implementan modelos de árboles de decisión como Random Forest y XGBoost.

3.2.1 KNN

En la clasificación k-NN, un objeto se clasifica por mayoría de voto de sus vecinos, asignándose el objeto a la clase más común entre sus k más cercanas vecinos (k es un número entero positivo, típicamente pequeño). Si k = 1, entonces el objeto simplemente se asigna a la clase de ese único vecino más cercano. A continuación, se explica gráficamente este algoritmo. [4]

Algoritmo kNN

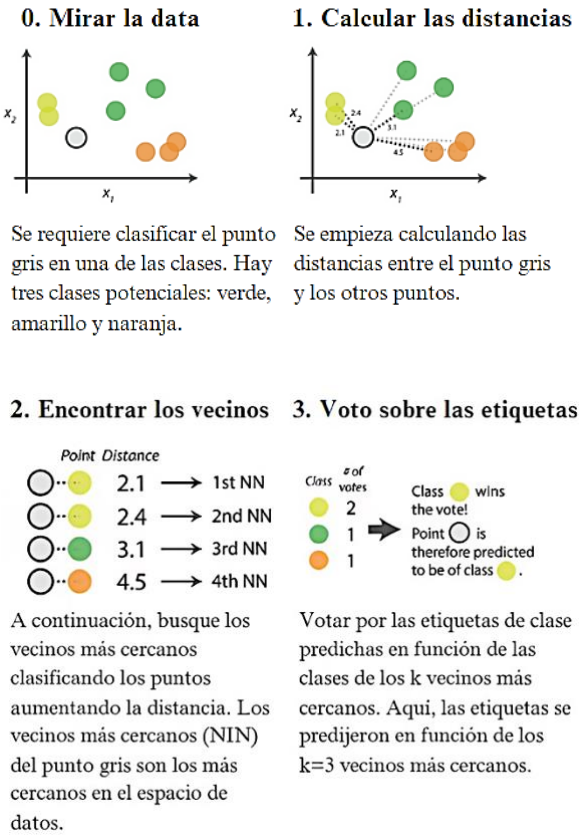

Ilustración 6. Funcionamiento de kNN

Se puede ver que la selección de k es bastante importante, al igual que la selección de sus datos de entrenamiento, porque esto es todo en lo que se basará el modelo predictivo. Con respecto a k, generalmente en casos binarios es mejor elegir un valor impar de K para evitar empates entre vecinos. Los valores de k ligeramente más altos también pueden actuar para reducir el ruido en los conjuntos de datos. Sin embargo, es mejor experimentar con diferentes valores de k y usar técnicas de validación cruzada para encontrar el mejor valor para cada caso específico. **[4]**

3.2.2 SVM

Una máquina de vectores de soporte, o SVM (Support Vector Machine), es un modelo de aprendizaje supervisado no paramétrico. El término "no paramétrico" se refiere a que SVM no hace suposiciones explícitas sobre la forma funcional de la relación entre los datos de entrada y salida. En cambio, se basa en la identificación de patrones complejos en los datos de entrenamiento para generar una función de decisión que pueda separar los datos en distintas clases. Las SVM construyen un hiperplano o un conjunto de hiperplanos de separación en un espacio dimensional alto o infinito, que puede usarse para clasificación, regresión u otras tareas. [5]

En otras palabras, dados los datos de entrenamiento etiquetados (aprendizaje supervisado), el algoritmo genera un hiperplano óptimo que categoriza nuevos ejemplos. En el espacio bidimensional, este hiperplano es una línea que divide un plano en dos partes donde, en cada clase, se encuentra a cada lado. [6]

Algoritmo SVM

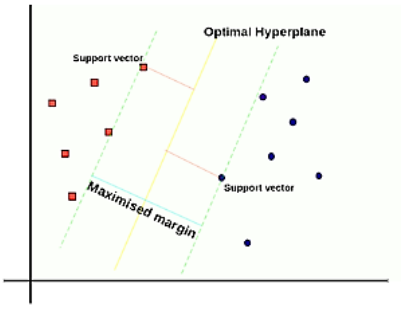

Ilustración 7. Algoritmo SVM

De acuerdo con el algoritmo SVM, encontramos los puntos más cercanos a la línea de ambas clases. Estos puntos se denominan vectores de soporte. Ahora, calculamos la distancia entre la línea y los vectores de soporte. Esta distancia se llama margen. El objetivo es maximizar el margen. El hiperplano para el cual el margen es máximo es el hiperplano óptimo. (Figura 7)

3.2.3 Regresión Logística

La regresión logística es un enfoque analítico predictivo que se utiliza a menudo en la detección de fraudes con tarjetas de crédito por diversas razones clave.

En primer lugar, la detección de fraude de tarjetas de crédito se categoriza como un problema de clasificación binaria: una transacción es o no es fraudulenta. La regresión logística es una técnica de clasificación binaria que ofrece la capacidad de proporcionar la probabilidad de que una transacción sea fraudulenta.

Además, este método es de naturaleza probabilística. Esto significa que, además de proporcionar una clasificación binaria (fraude/no fraude), también puede calcular la probabilidad de la ocurrencia de un evento. En el contexto de la detección de fraude, esto proporciona una medida de la probabilidad de que una transacción sea fraudulenta, un atributo útil para priorizar las alertas de fraude.

Otra ventaja es su interpretabilidad. A diferencia de algunos métodos de aprendizaje automático más avanzados, los coeficientes de un modelo de regresión logística pueden interpretarse en términos de las probabilidades de los resultados. Esto puede ser beneficioso para comprender qué características están más fuertemente asociadas con el fraude.

En términos de eficiencia, ofrece ventajas sobre algoritmos más complejos. Esto puede ser particularmente valioso cuando se procesan grandes volúmenes de datos de transacciones.

Finalmente, la regresión logística es robusta a las correlaciones entre las características. Siendo un aspecto común en los datos de transacciones de tarjetas de crédito.

3.2.3.1 Funcionamiento del modelo de regresión logística

En la primera etapa, denominada función de puntuación, la regresión logística calcula una puntuación lineal. Esta puntuación, a menudo referida como el logit o log-odds, se basa en las características de entrada (X) y los coeficientes del modelo (β).

$$z = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$ [1]

En la segunda etapa, la puntuación calculada se pasa a través de la función logística, también conocida como función sigmoide. Como resultado, se obtiene un valor de probabilidad (p) que oscila entre 0 y 1.

$$p = \frac{1}{1 + e^{-z}}$$ [2]

Finalmente, en la tercera etapa, se selecciona un umbral de decisión para producir una clasificación final. Si la probabilidad p supera este umbral, la observación se clasifica como 1 (la clase positiva); de lo contrario, se clasifica como 0 (la clase negativa). Comúnmente, el umbral se establece en 0.5.

Los coeficientes del modelo se determinan por medio de un proceso de optimización que busca maximizar la "verosimilitud" de los datos de entrenamiento dados los coeficientes. Este proceso se realiza a través de un algoritmo conocido como descenso de gradiente, o algún tipo de optimización de Newton-Raphson. [7]

3.2.4 Redes Neuronales (ANN)

Las Redes Neuronales Artificiales (ANN) representan una opción prominente en la detección de fraude en tarjetas de crédito, debido a diversas ventajas que ofrecen en este ámbito. A continuación, se enumeran algunas de las razones que respaldan la elección de ANN para tal propósito:

1. **Capacidad para modelar relaciones no lineales**: El conjunto de datos que usamos presenta características derivadas de la transformación PCA, las cuales a menudo encierran relaciones no lineales complejas. Las ANN son particularmente aptas para modelar estas relaciones, ya que pueden aprender de interacciones no lineales entre variables.

2. **Manejo de alta dimensionalidad**: El conjunto de datos cuenta con 30 características, una dimensión que las ANN son capaces de manejar eficientemente. No se requiere un proceso de selección de características previo, ya que las ANN pueden aprender de todas las características y determinar su importancia relativa.

3. **Aprendizaje profundo**: Las ANN, y en particular las Redes Neuronales Profundas (DNN), tienen la capacidad de aprender representaciones abstractas y de alto nivel a partir de los datos de entrada. Esto es especialmente útil en este conjunto de datos, ya que las características son transformaciones PCA y su interpretación no es intuitiva.

4. **Robustez frente al ruido y los datos faltantes**: Aunque este conjunto de datos está relativamente limpio, las ANN tienen la ventaja de ser robustas frente al ruido y pueden lidiar con datos faltantes, lo que las hace útiles en contextos más generales de detección de fraude. [8]

A pesar de estas ventajas, es importante considerar que las ANN pueden ser más difíciles de interpretar en comparación con otros modelos, como la regresión logística, y pueden requerir una mayor cantidad de datos y tiempo de procesamiento para su entrenamiento. Además, debido a su flexibilidad, las ANN pueden ser susceptibles al sobreajuste si no se configuran y entrenan de manera adecuada.

3.2.4.1 Metodología de ANN

Las ANN se inspiran en la estructura y funcionalidad del cerebro humano para el procesamiento de información. Estos modelos consisten en una serie de unidades de procesamiento, denominadas "neuronas", organizadas en capas múltiples y conectadas entre sí.

Una ANN típica, como se muestra abajo, incluye tres tipos de capas: capa de entrada, capas ocultas y capa de salida. La capa de entrada recibe los datos de entrada, las capas ocultas realizan la mayoría de los cálculos y la capa de salida produce el resultado final. [9]

[1] Aquí, X1 a Xn representan las características de entrada, mientras que β0 a βn son los coeficientes del modelo que se aprenden a partir de los datos de entrenamiento.

[2] En esta ecuación, e es la base del logaritmo natural (aproximadamente 2.71828), y z es la puntuación obtenida en la primera etapa.

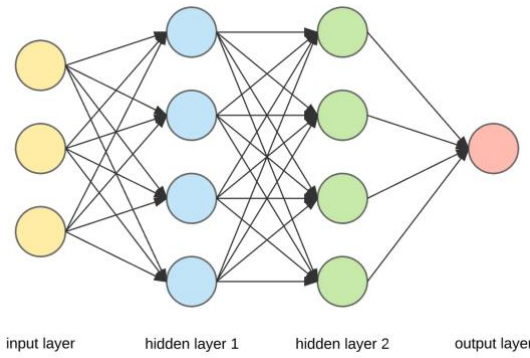

Ilustración 8 Diagrama básico de una red neuronal

1. Propagación hacia adelante

La fase de propagación hacia adelante de una ANN implica la transmisión de datos desde la capa de entrada a través de las capas ocultas y finalmente a la capa de salida. Cada neurona en una capa recibe una serie de entradas, cada una de las cuales se multiplica por un peso correspondiente. Estos productos se suman, se añade un término de sesgo y luego se aplica una función de activación al resultado.

$$z = w \cdot x + b^3$$

La función de activación f se aplica entonces a z para producir la salida a de la neurona:

$$a = f(z)$$

La función de activación puede tomar diversas formas, como la función sigmoidea, la función tangente hiperbólica, la función de la Unidad Lineal Rectificada (ReLU), entre otras.

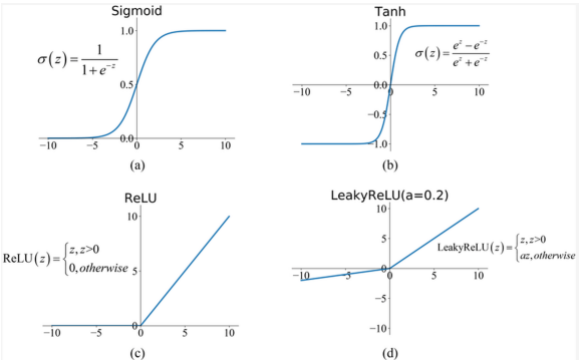

Ilustración 9 Funciones de activación

2. Función de coste y retropropagación

Tras la propagación hacia adelante, se calcula una función de coste C que mide la discrepancia entre la salida de la red y el valor objetivo. El objetivo es minimizar esta función de coste.

Para lograr esto, se utiliza un proceso llamado "retropropagación", que calcula el gradiente de la función de coste con respecto a los pesos y sesgos de la red, y luego ajusta estos parámetros para minimizar la función de coste:

$$w = w - \eta \nabla C$$

$$b = b - \eta \nabla C$$

3. Optimización

El proceso de propagación hacia adelante, cálculo de la función de coste y retropropagación se repite en múltiples iteraciones, en un proceso conocido como "entrenamiento" de la red. El objetivo es encontrar los pesos y sesgos que minimizan la función de coste, a menudo utilizando un algoritmo de optimización como el descenso de gradiente estocástico (SGD) o variantes más sofisticadas como Adam o RMSprop.

3.2.5 Random Forest

Random Forest es un algoritmo de la familia de los árboles de decisión.

Ilustración 10. Familia de árboles de decisión

Existen dos limitaciones principales de los árboles de decisión: son propensos a sobreajustarse y que tienden a no ser robustos, lo que significa que un pequeño cambio en los datos de entrenamiento da como resultado un árbol muy diferente. El modelo Random Forest supera estas dos deficiencias de los árboles de decisión generando muchos árboles de decisión y luego agregando las predicciones de cada árbol individual a una sola predicción del modelo.

Crear y luego combinar los resultados de un montón de árboles de decisiones parece bastante básico, sin embargo, simplemente crear múltiples árboles a partir de los mismos datos de entrenamiento no sería productivo, daría como resultado una serie de árboles fuertemente correlacionados. Todos estos árboles ordenarían los datos de la misma manera, por lo que este método no tendría ninguna ventaja sobre un solo árbol de decisión. Aquí es donde Random Forest comienza a entrar en juego. Para descorrelacionar los árboles que componen un bosque aleatorio, se lleva a cabo un proceso llamado agregación de arranque (también conocido como embolsado, *bagging*). El *bagging* genera nuevos conjuntos de datos de entrenamiento a partir de un conjunto de datos

[3] Donde w es el vector de pesos, x es el vector de entrada, b es el sesgo, y z es la suma ponderada de las entradas.

original mediante el muestreo de los datos de entrenamiento originales con reemplazo (*bootstrapping*, cualquier prueba o métrica que utilice muestreo aleatorio con reemplazo). Esto se repite para tantos árboles de decisión que conformarán el bosque aleatorio. Cada conjunto de datos de arranque individual se usa luego para construir un árbol. Este proceso reduce efectivamente la varianza (error introducido por el ruido aleatorio en los datos de entrenamiento, es decir, sobreajuste) del modelo sin aumentar el sesgo (falta de ajuste). Por sí solo, empaquetar los datos de entrenamiento para generar varios árboles crea lo que se conoce como un modelo de árboles empaquetados.

También se implementa un proceso similar llamado método de subespacio aleatorio (llamado empaquetado de atributos o empaquetado de características, *attribute bagging* o *feature bagging*) para crear un modelo de bosque aleatorio. Para cada árbol, se muestrea un subconjunto de las posibles variables predictoras, lo que da como resultado un conjunto más pequeño de variables predictoras para seleccionar para cada árbol. Esto descorrelaciona aún más los árboles al evitar que las variables predictoras dominantes sean las primeras o las únicas variables seleccionadas para crear divisiones en cada uno de los árboles de decisión individuales. Sin implementar el método de subespacio aleatorio, existe el riesgo de que una o dos variables predictoras dominantes se seleccionen consistentemente como la primera variable de división para cada árbol de decisión, y los árboles resultantes estarían altamente correlacionados. La combinación de embolsado y el método de subespacio aleatorio da como resultado un modelo de bosque aleatorio. [10]

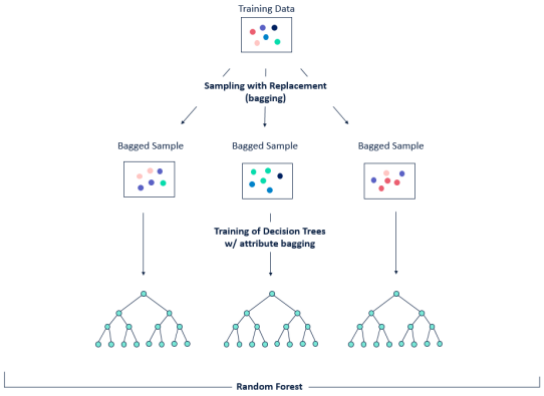

Ilustración 11. Funcionamiento y lógica de Random Forest

3.2.6 XGBoost, Extreme Gradient Boosting

Basado también en arboles de decisión, es considerado como el algoritmo más evolucionado en el estado del arte. Se trata de un algoritmo paralelizable, es decir permite usar de manera óptima toda la potencia de procesado de la que se dispone. Se pueden utilizar múltiples núcleos a la vez. Pero también permite ejecutarlo en GPUs en paralelo, o incluso en clústers de servidores. Por lo que tiene una rapidez de entrenamiento superior, entrenar el modelo en datasets de grandes volúmenes de datos no resulta en tanto problema como puede serlo para otro algoritmo. [11]

XGBoost utiliza el principio de boosting. La idea detrás del boosting es generar múltiples modelos de predicción "débiles" secuencialmente, y que cada uno de estos tome los resultados del modelo anterior, para generar un modelo más "fuerte", con mejor poder predictivo y mayor estabilidad en sus resultados.

Para conseguir un modelo más fuerte a partir de estos modelos débiles, se emplea un algoritmo de optimización, este caso Gradient Descent (descenso de gradiente). Durante el entrenamiento, los parámetros de cada modelo débil son ajustados iterativamente tratando de encontrar el mínimo de una función objetivo, que puede ser la proporción de error en la clasificación, el área bajo la curva (AUC), la raíz del error cuadrático medio (RMSE) o alguna otra.

Cada modelo es comparado con el anterior. Si un nuevo modelo tiene mejores resultados, entonces se toma este como base para realizar modificaciones. Si, por el contrario, tiene peores resultados, se regresa al mejor modelo anterior y se modifica ese de una manera diferente. Qué tan grandes son los ajustes de un modelo a otro es uno de los hiper parámetros que se debe definir.

Este proceso se repite hasta llegar a un punto en el que la diferencia entre modelos consecutivos es insignificante, lo cual nos indica que hemos encontrado el mejor modelo posible, o cuando se llega al número de iteraciones máximas definido por el usuario. [12]

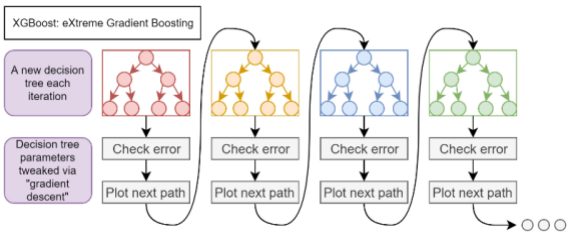

Ilustración 12. Funcionamiento de XGBoost

4. Desarrollo de la Propuesta

Para poder encontrar una solución en la detección de fraude, se aborda el proyecto como un problema de clasificación, ya que el objetivo es clasificar cada transacción como fraudulenta o no. En segundo lugar, determinar si una transacción puede llegar a ser fraude o no en función de las características y/o patrones encontrados de las transacciones pasadas, lo cual sería una predicción.

Se seleccionó un dataset, de los disponibles en la página Kaggle: creditcardfraud. [13], el cual cuenta con información para trabajar con el objetivo de entrenar un modelo de aprendizaje capaz de detectar casos de fraudes en las transacciones. Es un dataset con transacciones anonimizadas de tarjetas de crédito de septiembre de 2013 realizadas por titulares de tarjetas europeos. Este conjunto de datos presenta transacciones que ocurrieron en dos días, donde tenemos 492 fraudes de 284,807 transacciones. Si bien el conjunto de datos no presenta valores nulos, está muy desequilibrado, la clase positiva (fraudes) representa el 0,172 % de todas las transacciones. Contiene solo variables de entrada numéricas que son el resultado de una transformación PCA. Las características V1, V2, … V28 son los principales componentes obtenidos con PCA, las únicas características que no han sido transformadas con PCA son 'Tiempo' y 'Cantidad'. La característica 'Tiempo' contiene los segundos transcurridos entre cada transacción y la primera transacción en el conjunto de datos. La función 'Cantidad' es la cantidad de la transacción. La variable 'Class' es la variable de respuesta y toma valor 1 en caso de fraude y 0 en caso contrario. [13]

Una vez definido el dataset, se procede con el análisis exploratorio de los datos. Se realizaron diversas técnicas de análisis de datos: estadística descriptiva, gráficos y visualizaciones: histogramas, boxplots, matriz de correlación y análisis PCA; para identificar patrones, tendencias, examinar las distribuciones de las variables, detectar valores atípicos y evaluar la calidad de los datos.

Finalmente, se toma la decisión de usar técnicas de aprendizaje supervisado, ya que se dispone de un dataset etiquetado, es decir, se conoce si la transacción es fraude o no fraude. Los modelos implementados de aprendizaje supervisado fueron presentados en el punto 3.2.

5. Experimentación, Resultados

Uno de los desafíos en la experimentación con un dataset muy desbalanceado radica en cómo medir la performance del modelo. Si la mayoría de los datos pertenecen a una clase específica, un modelo puede lograr un alto grado de precisión simplemente clasificando todos los ejemplos en esa clase, sin importar cuántos se equivoque al clasificar los ejemplos de las otras clases. Por lo tanto, encontrar una medida de evaluación adecuada es importante para asegurarse de que el modelo está haciendo predicciones precisas y útiles para ambas clases, y no simplemente para la clase dominante. Si no, en tales casos, obtiene una precisión bastante alta simplemente prediciendo la clase mayoritaria, pero no logra capturar la clase minoritaria, que suele ser el objetivo de crear el modelo en primer lugar. Por lo que, en nuestro caso, seleccionamos métricas descartando accuracy, métricas especialmente buenas identificando una de las clases (transacciones de fraude) como Recall, Matriz de Confusión y Precisión.

El recall es una métrica que mide la eficacia de qué tan bien clasificó el modelo los casos positivos. Y la precisión nos indica qué tan bien el modelo hizo su tarea. Mientras que la matriz de confusión nos ayuda a evaluar la frecuencia con la que un modelo clasifica correcta o incorrectamente las diferentes clases a las que pertenecen los datos.

Teniendo presente estas métricas, se detalla a continuación la experimentación con cada algoritmo.

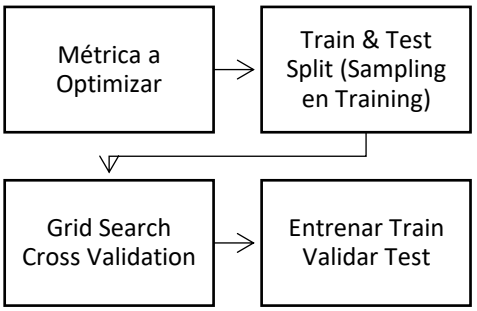

Ilustración 13. Pasos en Experimentación de Modelos

5.1 KNN

5.1.1 Determinación de valor de k

Para encontrar el mejor valor de k en K-NN, se utiliza la técnica Grid Search Cross Validation. Se define una lista de posibles valores de "k" que se quieren probar y se especifican estos valores en el parámetro "param_grid" de la instancia de GridSearchCV. Luego, se crea una instancia del clasificador K-NN y se utiliza GridSearchCV para ajustar el modelo a los datos de entrenamiento y buscar el valor óptimo de "k". El parámetro "cv" se utiliza para especificar el número de pliegues en la validación cruzada y el parámetro "scoring" se utiliza para especificar la métrica de evaluación que se quiere optimizar. En este caso, se ha especificado 'recall' como la métrica de evaluación para el modelo.

En nuestro caso, luego de ejecutar GridSearchCV, el k óptimo es 1.

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

# Definir una lista de posibles valores de k
k_list = list(range(1, 10))

# Definir los parámetros de la cuadrícula para GridSearchCV
param_grid = {'n_neighbors': k_list}

# Crear una instancia de GridSearchCV y ajustar el modelo a los datos
knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='recall')
grid_search.fit(X_train, y_train)

# Imprimir los resultados de la búsqueda en cuadrícula
print("Mejor valor de k:", grid_search.best_params_['n_neighbors'])
print("Sensibilidad media en validación cruzada:", grid_search.best_score_)
```

```
Mejor valor de k: 1
Sensibilidad media en validación cruzada: 0.7731256085686466
```

Ilustración 14. Código GridSearch CV para determinar k

Es posible que el valor óptimo de k encontrado por GridSearchCV sea 1 porque el conjunto de datos podría estar altamente sesgado hacia una de las clases. Dado que la clase positiva es la minoría en el conjunto de datos, un valor de k más pequeño (en este caso, 1) podría tener un mejor rendimiento en términos de sensibilidad (recall) para la clase positiva al predecir la clase minor.

5.1.2 kNN Resultados

Resultados Dataset Desbalanceado. Recall bueno a pesar del desbalance de datos

```
[[56847    15]
 [   16    84]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56862
           1       0.85      0.84      0.84       100

    accuracy                           1.00     56962
   macro avg       0.92      0.92      0.92     56962
weighted avg       1.00      1.00      1.00     56962
```

Ilustración 15. Resultados Dataset Desbalanceado

Resultados Dataset Balanceado con Random-Undersampling. Recall bueno, pero precisión baja para clase 1.

```
[[53354  3508]
 [    8    92]]
              precision    recall  f1-score   support

           0       1.00      0.94      0.97     56862
           1       0.03      0.92      0.05       100

    accuracy                           0.94     56962
   macro avg       0.51      0.93      0.51     56962
weighted avg       1.00      0.94      0.97     56962
```

Ilustración 16. Resultados Dataset Balanceado con Random-Undersampling

Resultados Dataset Balanceado con Random-Oversampling. Recall bueno, pero precisión baja para clase 1

```
[[52970  3894]
 [    7    91]]
              precision    recall  f1-score   support

           0       1.00      0.93      0.96     56864
           1       0.02      0.93      0.04        98

    accuracy                           0.93     56962
   macro avg       0.51      0.93      0.50     56962
weighted avg       1.00      0.93      0.96     56962
```

Ilustración 17. Resultados Dataset Balanceado con Random-Oversampling

Con el modelo KNN, podemos ver que existe una mejor predicción con el dataset con técnica Over-Sampling si consideramos el Recall para la clase 1 de un 93%, prediciendo 91 transacciones fraude y 7 que deberían haber sido clasificadas como fraude, sin embargo, su precisión es muy baja, es decir que el modelo no realiza su tarea de una forma óptima.

Los modelos tienen dificultades para detectar transacciones fraudulentas, lo que sugiere que se necesitan técnicas de modelado más avanzadas o más datos para mejorar el rendimiento del modelo. También se observa una clara asimetría en la precisión y recall entre las clases, por lo que es necesario prestar atención al balance de clases en el conjunto de datos y tomar medidas para abordar este problema.

5.2 SVM

5.2.1 SVM Hiperparámetros

Se ejecuta SVM con clasificadores SVC o LinearSVC, pero el tiempo de ejecución superó las 10 horas y continuaba corriendo. Intentando buscar una explicación y sobre todo optimizar la performance de este algoritmo, consultando la teoría explicada en punto 4.2.2 se llega a la conclusión que SVM intenta encontrar el hiperplano que maximiza el margen entre diferentes clases, lo que implica resolver un problema de optimización cuadrática con una gran cantidad de restricciones. A medida que aumenta la cantidad de puntos de datos, también aumenta la cantidad de restricciones en el problema de optimización, lo que hace que su solución sea más costosa desde el punto de vista computacional. Además, el truco del kernel, que se usa comúnmente con SVM para transformar los datos en un espacio de mayor dimensión, puede aumentar aún más el tiempo de entrenamiento y los requisitos de memoria.

Luego de investigar, para entrenar un modelo SVM en un gran conjunto de datos, se pueden usar varias

técnicas de optimización y aproximaciones para acelerar el proceso. Una de esas técnicas es el descenso de gradiente estocástico (SGD), que puede optimizar la función objetivo de SVM utilizando solo un subconjunto de los datos en cada iteración.

SGDClassifier implementa modelos lineales regularizados con aprendizaje de descenso de gradiente estocástico (SGD): el gradiente de la pérdida se estima cada muestra a la vez y el modelo se actualiza a lo largo del camino con un programa de fuerza decreciente (también conocido como tasa de aprendizaje). SGD permite el aprendizaje de minilotes (en línea/fuera del núcleo) a través del método de ajuste parcial. Para obtener los mejores resultados con el programa de tasa de aprendizaje predeterminado, los datos deben tener una media cero y una varianza unitaria. Esta implementación funciona con datos representados como matrices densas o dispersas de valores de punto flotante para las entidades. El modelo al que se ajusta se puede controlar con el parámetro de pérdida; por defecto, se ajusta a una máquina de vectores de soporte lineal (SVM). [7]

Se experimenta el algoritmo con los parámetros que son los valores predeterminados en el SGDClassifier. Pero se realiza un GridSeacrh CV para determinar los valores óptimos de parámetros a usar en el modelo. Los parámetros para evaluar son:

alpha	Controla la fuerza de regularización del modelo SVM. Constante que multiplica el término de regularización. Cuanto mayor sea el valor, más fuerte será la regularización. También se utiliza para calcular la tasa de aprendizaje cuando learning_rate se establece en "óptimo". Los valores deben estar en el rango [0.0, inf).
penalty	Determina el tipo de regularización aplicada en el modelo SVM. Establece la sanción que se utilizará. El valor predeterminado es 'l2', que es el regularizador estándar para los modelos SVM lineales. 'l1' y 'elasticnet' pueden aportar escasez al modelo (selección de funciones) que no se puede lograr con 'l2'. No se agrega penalización cuando se establece en Ninguno.
loss	Determina la función de pérdida utilizada para el entrenamiento del modelo SVM. Se proporcionan dos opciones para la búsqueda de cuadrícula: 'hinge' para pérdida de bisagra y 'log' para pérdida logística.
max_iter	Controla el número máximo de iteraciones permitidas para la convergencia del modelo SVM. El número máximo de pases sobre los datos de entrenamiento (también conocidos como épocas). Solo afecta el comportamiento en el método de ajuste, y no en el método de ajuste parcial. Los valores deben estar en el rango [1, inf).

Tabla 1. Tabla de parámetros usados para SDGClassifier - SVM

GridSearchCV, en teoría, ayudó a encontrar los hiperparámetros óptimos para el modelo SVM y, por lo tanto, mejorar la precisión y el rendimiento general del modelo. En sección 6.2.1 se detallan los resultados.

5.2.2 SVM Resultados

Resultados Dataset Desbalanceado. Recall muy bajo para clase 1. Modelo deficiente.

```
[[56854    10]
 [   53    45]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.82      0.46      0.59        98

    accuracy                           1.00     56962
   macro avg       0.91      0.73      0.79     56962
weighted avg       1.00      1.00      1.00     56962
```

Ilustración 18. Resultados Dataset Desbalanceado

Resultados Dataset Balanceado con Random-Undersampling. Recall mejorado para clase 1 pero precisión muy baja

```
[[53835  3029]
 [    7    91]]
              precision    recall  f1-score   support

           0       1.00      0.95      0.97     56864
           1       0.03      0.93      0.06        98

    accuracy                           0.95     56962
   macro avg       0.51      0.94      0.51     56962
weighted avg       1.00      0.95      0.97     56962
```

Ilustración 19. Resultados Dataset Balanceado con Random-Undersampling

Resultados Dataset Balanceado con Random-Oversampling. Recall regular para clase 1, aún con precisión baja.

```
[[55474  1388]
 [    9    91]]
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56862
           1       0.06      0.91      0.12       100

    accuracy                           0.98     56962
   macro avg       0.53      0.94      0.55     56962
weighted avg       1.00      0.98      0.99     56962
```

Ilustración 20. Resultados Dataset Balanceado con Random-Oversampling

En general, los modelos parecen tener una alta precisión para la clase 0, pero una precisión variable para la clase 1. El recall y el F1-score también varían considerablemente según el modelo y la clase. Podríamos concluir que el modelo con undersampling tiene un mejor recall para la clase 1, pero su precisión no es muy buena.

5.3 Regresión Logística

Para crear el modelo de regresión logística, hemos comprobado que usaremos todas las variables independientes tras el análisis exploratorio, y no hemos detectado valores atípicos.

Utilizamos la búsqueda en cuadrícula (grid search) para encontrar los mejores hiperparámetros para el modelo. Estos son:

C	Controla la fuerza de la regularización en el modelo de regresión logística. Un valor más alto de C indica una menor regularización, lo que puede mejorar el rendimiento del modelo, pero también aumentar el riesgo de sobreajuste. Un valor más bajo de C indica una mayor regularización, lo que puede prevenir el sobreajuste, pero también reducir el rendimiento del modelo. En este ejemplo, se prueban los valores de 0.01, 0.1, 1 y 10 para C.
solver	Indica el tipo de algoritmo de solución a utilizar para el modelo de regresión logística. Hay varios tipos de solucionadores disponibles en scikit-learn, incluyendo 'lbfgs', 'liblinear', 'sag' y 'saga'. Cada solucionador tiene sus propias ventajas y desventajas, y puede ser más adecuado para diferentes tipos de datos. En este caso, se están probando todos los solucionadores disponibles.

Tabla 2. Tabla de parámetros usados para Regresión Logística

Una vez se encuentran los mejores hiperparámetros, entrenamos el modelo y hacemos predicciones con los datos de prueba. También utilizamos validación cruzada con 5 pliegues (folds) y la puntuación F1 macro para evaluar la precisión y exhaustividad del modelo. La puntuación F1 macro se calcula como la media ponderada de la precisión y la exhaustividad de todas las clases, y es una métrica útil para conjuntos de datos desequilibrados donde una clase puede ser mucho más grande que las demás.

```
Confusion Matrix:
 [[56855     9]
 [   41    57]]

 Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.86      0.58      0.70        98

    accuracy                           1.00     56962
   macro avg       0.93      0.79      0.85     56962
weighted avg       1.00      1.00      1.00     56962
```

Ilustración 21. Resultados Dataset Desbalanceado

```
Confusion Matrix:
 [[55371  1493]
 [    7    91]]

 Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     56864
           1       0.06      0.93      0.11        98

    accuracy                           0.97     56962
   macro avg       0.53      0.95      0.55     56962
weighted avg       1.00      0.97      0.99     56962
```

Ilustración 22. Resultados Dataset Balanceado con Undersampling

```
Confusion Matrix:
 [[55458  1406]
 [    7    91]]

 Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56864
           1       0.06      0.93      0.11        98

    accuracy                           0.98     56962
   macro avg       0.53      0.95      0.55     56962
weighted avg       1.00      0.98      0.99     56962
```

Ilustración 23 Resultados Dataset Balanceado con Random-Oversampling

Al comparar los tres modelos, se pudo observar que todos tienen una precisión global similar de alrededor del 97-98%. Sin embargo, al analizar los resultados en mayor profundidad, se encontró que el modelo con undersampling y el modelo con oversampling tienen un recall mucho mejor para la clase fraudulenta que el modelo sin balanceo.

Para tratar un problema de detección de fraudes, el principal objetivo es maximizar la detección de fraudes. En otras palabras, es esencial identificar la mayor cantidad posible de transacciones fraudulentas. Al tener un recall más alto para la clase de transacciones fraudulentas, los modelos con técnicas de balanceo están logrando detectar más transacciones fraudulentas en comparación con el modelo sin balanceo.

En conclusión, podemos afirmar que el modelo con técnicas de balanceo ya sea undersampling u oversampling, es el mejor modelo para este problema de detección de fraudes en transacciones con tarjetas de crédito. Aunque todos los modelos tienen una precisión global similar, el modelo con técnicas de balanceo logra una mejor detección de transacciones fraudulentas, lo que es el objetivo principal en este tipo de análisis.

5.4 Redes Neuronales (ANN)

Para seguir probando algoritmos robustos para grandes volúmenes de conjuntos de datos, intentamos hacer un modelo de ANN para predecir

fraudes. Una vez que separamos el conjunto de datos en tren y prueba, procedimos a diseñar la arquitectura de la red y se definió de la siguiente manera:

1. model = Sequential(): Se crea un modelo secuencial, lo que significa que las capas de la red se ordenan secuencialmente, y cada capa tiene exactamente una capa de entrada y una capa de salida.

2. model.add(Dense(32, input_dim=X_train.shape[1], activation='relu')): Se añade la primera capa oculta a la red. Esta es una capa densa o completamente conectada, lo que significa que cada neurona en esta capa está conectada a todas las neuronas en la capa anterior. La capa contiene 32 neuronas, y el número de neuronas de entrada (es decir, la dimensión de los datos de entrada) es igual al número de características en el conjunto de entrenamiento (X_train.shape[1]). La función de activación es la Unidad Lineal Rectificada (ReLU), que es una función comúnmente utilizada que devuelve la entrada si es positiva y 0 en caso contrario.

3. model.add(Dense(16, activation='relu')): Se añade una segunda capa oculta a la red, que también es una capa densa. Esta capa contiene 16 neuronas y también utiliza ReLU como su función de activación.

4. model.add(Dropout(0.5)): Se añade una capa de "*dropout*" a la red. Durante el entrenamiento, esta capa "desactiva" aleatoriamente un porcentaje (en este caso, el 50%) de las neuronas en la capa anterior. Esto ayuda a prevenir el sobreajuste.

5. model.add(Dense(1, activation=*'sigmoid'*)): Finalmente, se añade la capa de salida a la red. Esta es una capa densa con una sola neurona, ya que estamos realizando una tarea de clasificación binaria para detectar fraudes. La función de activación es la función sigmoide, que mapea su entrada a un valor entre 0 y 1, que puede interpretarse como la probabilidad de que la entrada pertenezca a la clase positiva.

Después de definir la arquitectura de la red, el modelo se compila con una función de pérdida de entropía cruzada binaria (adecuada para la clasificación binaria), un optimizador Adam (un algoritmo de optimización popular para el aprendizaje profundo) y métricas de precisión y recall.

Finalmente, el modelo se entrena en el conjunto de entrenamiento (X_train, y_train) durante 20 épocas (es decir, el modelo ve todo el conjunto de entrenamiento 20 veces), utilizando un tamaño de lote de 32 (es decir, el modelo actualiza sus pesos después de ver 32 ejemplos) y un conjunto de validación del 20% del tamaño del conjunto de entrenamiento (utilizado para monitorear el rendimiento del modelo durante el entrenamiento y ajustar los hiperparámetros).

Al entrenar el modelo, es importante evaluar su rendimiento. El método ***evaluate*** se utiliza para calcular la pérdida y las métricas de precisión y recall en el conjunto de prueba. Luego, el método ***predict*** genera las probabilidades de pertenencia a la clase positiva para cada ejemplo en el conjunto de prueba. Finalmente, estas probabilidades se convierten en etiquetas de clase binarias, estableciendo un umbral de 0.5. Estas etiquetas pueden usarse posteriormente para calcular métricas de rendimiento y comparar las predicciones del modelo con las verdaderas etiquetas de clase.

```
Confusion Matrix:
 [[56850    14]
 [   24    74]]

 Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.84      0.76      0.80        98

    accuracy                           1.00     56962
   macro avg       0.92      0.88      0.90     56962
weighted avg       1.00      1.00      1.00     56962
```

Ilustración 24. Resultados Dataset Desbalanceado

```
Confusion Matrix:
 [[56037   827]
 [   10    88]]

 Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.99      0.99     56864
           1       0.10      0.90      0.17        98

    accuracy                           0.99     56962
   macro avg       0.55      0.94      0.58     56962
weighted avg       1.00      0.99      0.99     56962
```

Ilustración 25. Resultados Dataset Balanceado con Undersampling

```
Confusion Matrix:
 [[56821    43]
 [   14    84]]

 Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.66      0.86      0.75        98

    accuracy                           1.00     56962
   macro avg       0.83      0.93      0.87     56962
weighted avg       1.00      1.00      1.00     56962
```

Ilustración 26. Resultados Dataset Balanceado con Oversampling

A la hora de determinar cuál es el mejor modelo de Redes Neuronales Artificiales (ANN) en la detección de fraudes entre el dataset original, undersampling y oversampling, debemos analizar las métricas de evaluación proporcionadas en cada caso: precisión, recall, f1-score y accuracy.

1. Dataset original:
    - Precisión clase 1: 0.84
    - Recall clase 1: 0.76
    - F1-score clase 1: 0.80
    - Accuracy: 1.00
2. Undersampling:
    - Precisión clase 1: 0.10
    - Recall clase 1: 0.90
    - F1-score clase 1: 0.17
    - Accuracy: 0.99
3. Oversampling:
    - Precisión clase 1: 0.66
    - Recall clase 1: 0.86
    - F1-score clase 1: 0.75
    - Accuracy: 1.00

En el contexto de la detección de fraudes, es crucial detectar la mayor cantidad de fraudes verdaderos sin aumentar demasiado las falsas alarmas. Por lo tanto, las métricas más importantes a considerar son recall y precisión para la clase 1 (fraude).

- El modelo de undersampling tiene el mayor recall (0.90), pero su precisión es muy baja (0.10), lo que significa que identifica correctamente la mayoría de los fraudes, pero también produce muchas falsas alarmas (falsos positivos).

- El modelo de oversampling tiene un buen equilibrio entre recall (0.86) y precisión (0.66), lo que indica que puede detectar adecuadamente los casos de fraude y mantener las falsas alarmas en un nivel aceptable.

- El modelo del dataset original tiene una buena precisión (0.84) pero un recall más bajo (0.76) en comparación con el modelo de oversampling.

Teniendo en cuenta estos factores, el modelo de **oversampling** parece ser el mejor entre los tres, ya que ofrece un buen equilibrio entre recall y precisión para la detección de fraudes. Aunque el modelo del dataset original tiene una precisión ligeramente mejor, su recall es más bajo, lo que podría resultar en la falta de detección de algunos fraudes.

5.5 Random Forest

Se busca el modelo óptimo de los árboles de decisión que maximice la mayor categorización/predicción de fraudes sin aumentar los falsos positivos, dándole importancia, a la métrica de recall, precisión

y matriz de confusión, como fue mencionado con anterioridad.

1. Data Desbalanceada:
    - Precisión clase 1: 0.96
    - Recall clase 1: 0.83
    - F1-score clase 1: 0.89
    - Accuracy. 1: 1.00
2. Data Balanceada + Undersampling:
    - Precisión clase 1: 0.05
    - Recall clase 1: 0.92
    - F1-score clase 1: 0.10
    - Accuracy. 1: 0.97
3. Data Balanceada + Oversampling:
    - Precisión clase 1: 0.94
    - Recall clase 1: 0.83
    - F1-score clase 1: 0.88
    - Accuracy. 1: 1.00

Los 3 modelos presentan una precisión variada del [5% al 96%], un Recall desde [83% al 92%], un F1-score desde [10% al 89%] y un Accuracy desde [97% al 100%].

El mejor recall para identificar más transacciones de la clase fraudulenta es con la "data balanceada y con la técnica de Under-Sampling" que tiene un [92%], luego con la "data balanceada más la técnica de Over-Sampling" [83%] y finalmente con "data desbalanceada"[83%].

El modelo con "data desbalanceada" tiene una excelente precisión [96%] pero tiene un recall aceptables [83%].

El modelo con "Data Balanceada + Undersampling" tiene muy baja precisión [5%] pero tiene el recall más alto [92%] entre los 3 modelos, detectando acertadamente los fraudes, pero generando falsos positivos.

El modelo con "Data Balanceada + Oversampling" tiene una alta precisión [94%] y tiene un recall bueno [83%] entre los 3 modelos, puede detectar adecuadamente los fraudes, y una mediana detección de falsos positivos.

Se concluye que el modelo que más se acerca al objetivo de maximizar la "detección de fraudes" es el modelo con "datos balanceados con la técnica de under-sampling" que tiene un excelente recall [92%] a pesar de tener una baja precision [5%] y F1-score razonablemente alto, se ha dado más importancia a la detección de fraudes que a la precisión con la identificación de falsos positivos o falsas alarmas.

5.6 XGBoost

5.6.1 XGBoost hiperparámetros.

Se usan los siguientes parámetros:

max_depth	Defina la profundidad máxima del árbol de decisión, determina el número máximo de niveles permitidos en el árbol, dependiendo

	del valor si es excesivo (underfitting) o ajuste insuficiente (overfitting).
learning_rate	Define la contribución de cada árbol aporta al modelo final, dependiendo del valor si es excesivo (learning_rate) o ajuste insuficiente (learning_rate).
n_estimators	Define el número de árboles de decisión que se construye en el modelo, si mayor es el número de árboles, dependiendo del valor si es excesivo (sobreajuste) o ajuste insuficiente (ajuste insuficiente).

Tabla 3. Tabla de parámetros usados para XGBoost

Nuevamente, se busca elegir la técnica que maximice la mayor cantidad de fraudes, sin aumentar los falsos positivos.

1. Data Desbalanceada:
   - Precisión clase 1: 0.96
   - Recall clase 1: 0.83
   - F1-score clase 1: 0.89
   - Accuracy. 1: 1.0
2. Data Balanceada + Undersampling:
   - Precisión clase 1: 0.05
   - Recall clase 1: 0.92
   - F1-score clase 1: 0.10
   - Accuracy. 1: 0.97
3. Data Balanceada + Oversampling:
   - Precisión clase 1: 0.94
   - Recall clase 1: 0.83
   - F1-score clase 1: 0.88
   - Accuracy. 1: 1.00

El mejor recall para identificar más transacciones de la clase fraudulenta es con la "data balanceada y con la técnica de Under-Sampling" que tiene un [92%], luego con la "data balanceada más la técnica de Over-Sampling" [83%] y finalmente con "data desbalanceada"[83%].

El modelo con "data desbalanceada" tiene una excelente precisión [96%] pero tiene un recall aceptable [83%].

El modelo con "Data Balanceada + Undersampling" tiene muy baja precisión [5%] pero tiene el recall más alto [92%] entre los 3 modelos, detectando acertadamente los fraudes, pero generando falsos positivos.

El modelo con "Data Balanceada + Oversampling" tiene una alta precisión [94%] y tiene un recall bueno [83%] entre los 3 modelos, puede detectar adecuadamente los fraudes, y una mediana detección de falsos positivos.

Se concluye que el modelo que más se acerca al objetivo de maximizar la "detección de fraudes" es el modelo con "datos balanceados con la técnica de under-sampling" que tiene un excelente recall [92%] a pesar de tener una baja precisión [5%] y F1-score razonablemente alto, se ha dado más importancia a la detección de fraudes que a la precisión con la identificación de falsos positivos o falsas alarmas.

7. Conclusiones

Con los resultados finalmente obtenidos (Véase en el Anexo).

Se concluye que el modelo que más se acerca al objetivo de maximizar la "detección de fraudes" es el modelo con "datos balanceados con la técnica de under-sampling" que tiene un excelente recall [92%] a pesar de tener una baja precisión [4%] y F1-score razonablemente alto, se ha dado más importancia a la detección de fraudes que a la precisión con la identificación de falsos positivos o falsas alarmas, con el supuesto que se pierde más económicamente no detectando fraudes (recall [92%]) que detectando fraudes falsos o falsas alarmas precisión [4%].

Se ha desarrollado un caso de uso, con supuesto económico de costos promedios, si a la entidad financiera le cuesta un promedio de 1,000 € por no detectar un fraude y 10 € por una falsa alarma.

Para el caso del modelo XGBOOST con "datos balanceados con la técnica de undersampling" visualizamos que hay 8 Falso Negativos, es decir fraudes no detectados, por lo tanto, al banco le costaría 8,000 € siendo el monto más bajo y el algoritmo más optimo computacionalmente para ser elegido.

8. Trabajo Futuro

Finalmente, se detallan algunas ideas para mejorar el proyecto en el futuro:

- Elección de dataset con variables no anonimizadas. Con el objetivo de sacar mejores conclusiones como por ejemplo ¿Qué época y horarios se producen los fraudes? ¿En qué lugares? ¿Cuáles son los segmentos de clientes más vulnerables?, etc.
- Incorporación de datos en tiempo real. Podría implementarse el modelo seleccionado y recibir datos en tiempo real.
- Mejora de los hiperparámetros de los modelos (cross-validation, capas de redes neuronales, etc) Debido a la falta de tiempo y recursos computacionales, quizás se pueden programar u optimizar para ser ejecutados en entornos GPUs y probar con diferentes parámetros.
- Prueba con otros modelos (clasificador Naive Bayes, LSTM, Stacking) o ensamblado de modelos

combinando varios modelos individuales para producir una predicción final. En lugar de elegir un solo modelo, se puede combinar varios modelos diferentes para obtener una mayor precisión en las predicciones.

- Disponer del significado de las variables para saber a qué se refiere cada variable.

Referencias

[1] C. S, «Detection of fraudulent financial statements using the hybrid data mining approach,» 2016.

[2] Analytics Vidhya, «10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2023),» 03 2023. [En línea]. Available: https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/.

[3] Imbalanced-learn, «imbalanced-learn documentation,» Diciembre 2022. [En línea]. Available: https://imbalanced-learn.org/stable/.

[4] S. Kellerhals, «INTRO TO MACHINE LEARNING IN R (K NEAREST NEIGHBOURS ALGORITHM),» [En línea]. Available: https://ourcodingclub.github.io/tutorials/machine-learning/.

[5] R. Pupale, «Support Vector Machines(SVM) — An Overview,» Junio 2018. [En línea]. Available: https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989#:~:text=SVM%20or%20Support%20Vector%20Machine,separates%20the%20data%20into%20classes..

[6] Scikit-learn - Papers with code, «Support Vector Machine,» [En línea]. Available: https://paperswithcode.com/method/svm.

[7] «geeksforgeeks,» [En línea]. Available: https://www.geeksforgeeks.org/understanding-logistic-regression/.

[8] «¿Qué es una red neuronal?,» [En línea]. Available: https://aws.amazon.com/es/what-is/neural-network/.

[9] A. Dertat, «Applied Deep Learning - Part 1: Artificial Neural Networks,» [En línea]. Available: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6#04e7.

[10] SydneyF, «Seeing the Forest for the Trees: An Introduction to Random Forest,» 2021. [En línea]. Available: https://community.alteryx.com/t5/Alteryx-Designer-Desktop-Knowledge-Base/Seeing-the-Forest-for-the-Trees-An-Introduction-to-Random-Forest/ta-p/158062.

[11] F. Sanz, «Cómo funciona el algoritmo XGBoost en Python,» [En línea]. Available: https://www.themachinelearners.com/xgboost-python/.

[12] M. V. J. Bosco, «Tutorial: XGBoost en Python,» 2020. [En línea]. Available: https://medium.com/@jboscomendoza/tutorial-xgboost-en-python-53e48fc58f73.

[13] Machine Learning Group - ULB (Owner), «Credit Card Fraud Detection,» [En línea]. Available: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud.

[14] scikit-learn, «sklearn.linear_model.SGDClassifier,» [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html.

Anexo

ALGORITMOS		MÉTRICAS CLASE 1				Matríz de confusión				Falsa alarma (*)	Fraude no detectado (*)	
		Recall	Precisión	F1-Score	Accuracy	TN	FP	FN	TP	10	1000	
REGRESIÓN LOGÍSTICA	Data desbalanceada	0,58	0,86	0,7	1	56855	9	41	57	90	41.000	41.090
	Data balanceada - Under-Sampling	**0,93**	0,06	0,11	0,97	55371	1493	7	91	14.930	7.000	21.930
	Data balanceada - Over-Sampling	**0,93**	0,06	0,11	0,98	55458	1406	7	91	14.060	7.000	21.060
ANN	Data desbalanceada	0,76	0,84	0,8	1	56850	14	24	74	140	24.000	24.140
	Data balanceada - Under-Sampling	0,9	0,1	0,17	0,99	56037	827	10	88	8.270	10.000	18.270
	Data balanceada - Over-Sampling	**0,86**	**0,66**	**0,75**	**1**	**56821**	**43**	**14**	**84**	**430**	**14.000**	**14.430**
KNN	Data desbalanceada	0,8	**0,96**	0,87	1	56859	3	20	80	30	20.000	20.030
	Data balanceada - Under-Sampling	0,9	0,08	0,15	0,98	55859	1003	10	90	10.030	10.000	20.030
	Data balanceada - Over-Sampling	0,84	0,85	0,84	1	56847	15	16	84	150	16.000	16.150
SVM	Data desbalanceada	0,51	0,83	0,63	1	56854	10	48	50	100	48.000	48.100
	Data balanceada - Under-Sampling	**0,92**	0,01	0,03	0,88	50133	6731	8	90	67.310	8.000	75.310
	Data balanceada - Over-Sampling	**0,92**	0,07	0,13	0,98	55640	1222	8	92	12.220	8.000	20.220
XGBOOST	Data desbalanceada	0,76	0,95	0,85	0,99	56853	4	25	80	40	25.000	25.040
	Data balanceada - Under-Sampling	**0,92**	0,04	0,08	0,96	54709	2148	8	97	21.480	8.000	29.480
	Data balanceada - Over-Sampling	0,82	0,54	0,65	1	56784	73	19	86	730	19.000	19.730
RANDOM FOREST	Data desbalanceada	0,83	0,96	0,89	1	56849	4	18	91	40	18.000	18.040
	Data balanceada - Under-Sampling	**0,92**	0,05	0,1	0,97	54997	1856	9	100	18.560	9.000	27.560
	Data balanceada - Over-Sampling	0,83	0,94	0,88	0,88	56847	6	18	91	60	18.000	18.060

* Supuesto de costos promedios €