Seminario II
MIOTIA
Maestro: Gabriel Alberto García Mireles
Alumno: Eduardo Alonso Martinez Chenoweth

MARCO TEÓRICO

1.1 Inteligencia Artificial

1.1.1 Definiciones Fundamentales

La Inteligencia Artificial (IA) es un campo de la ciencia de la computación cuyo objetivo científico es comprender los principios del comportamiento inteligente. Sin embargo, cuenta con un objetivo tecnológico el cual es diseñar y construir sistemas que tengan este comportamiento (Russell & Norvig, 2004). Debido a su rápido desarrollo y a la dificultad de definir la propia "inteligencia" no existe una definición única aceptada de lo que es IA (Russell & Norvig, 2004).

El contexto utilizado por Russell & Norvig (2004) para definir lo que es IA comienza por centrarse en la racionalidad: la capacidad de un sistema de tomar la decisión correcta para lograr su objetivo. De esta manera, el enfoque que se le da es matemático y optimizable, lo cual permite definir el "éxito" utilizando funciones de coste y métricas de rendimiento, lo cual sienta bases para el desarrollo de algoritmos.

1.1.2 Categorías de la Inteligencia Artificial

Existen varios enfoques para clasificar los sistemas de IA, el enfoque utilizado por Russell & Norvig (2004) será el utilizado. Este define dos dimensiones: la primera, pensamiento y comportamiento, la segunda, diferencia fidelidad a la actuación humana y seguir un modelo ideal de pensamiento perfectamente lógico y coherente. Esto da un total de cuatro categorías las cuales serán descritas a continuación:

A. *Sistemas que piensan como humanos:* Este enfoque, relacionado con la ciencia cognitiva, busca emular el pensamiento humano a través de modelos computacionales que simulen el proceso de razonamiento y resolución de problemas de la mente humana. Este enfoque fue utilizado en los orígenes de las redes neuronales artificiales.

B. *Sistemas que actúan como humanos:* Este enfoque se asocia a la Prueba de Turing. El sistema es considerado inteligente si su comportamiento con el exterior es indistinguible del ser humano, sin tomar en cuenta como realiza sus procesos internos para obtener estos resultados. Robots humanoides y asistentes virtuales son ejemplos de este enfoque.

C. *Sistemas que piensan racionalmente:* Aquí entran las "leyes del pensamiento" utilizadas en la lógica formal las cuales son la base de la computación inteligente. Esto quiere decir, que basados en un conjunto de premisas el sistema sea capaz de realizar inferencias lógicas correctas. Un ejemplo podría ser los sistemas expertos.

D. *Sistemas que actúan racionalmente (El Agente Racional):* Este es el enfoque utilizado en IA moderna. Un agente racional se refiere a la entidad que mediante lo que percibe en su entorno y la información disponible, actúa de manera optima para maximizar las posibilidades de lograr el objetivo propuesto. Para este enfoque se entiende la racionalidad no como un pensamiento consiente, sino como la acción optima.

El proyecto de esta tesis se enmarca firmemente dentro de la categoría de "sistemas que actúan racionalmente". El objetivo no es crear un sistema que piense o actúe como un analista financiero humano, sino desarrollar un agente computacional que, a partir de los datos de una declaración patrimonial (su percepción), tome la decisión óptima (marcarla como 'normal' o 'anómala') para maximizar una medida de rendimiento definida.

1.2 Aprendizaje Automático

El aprendizaje automático (Machine Learning, ML) es una rama de la IA que tiene como objetivo el estudiar algoritmos y sistemas que logran mejorar su rendimiento de manera automática a través de la experiencia (Bishop, 2006). Es decir, en lugar de seguir instrucciones explícitamente programadas, logra aprender de patrones y hacer relaciones desde los datos.

La definición más citada en la literatura académica es la propuesta por Mitchell (1997) en el libro “Machine Learning” (1997), la cual es la siguiente:

> "Se dice que un programa de computadora aprende de la experiencia E con respecto a alguna clase de tareas T y una medida de rendimiento P, si su rendimiento en las tareas en T, medido por P, mejora con la experiencia E."(Mitchell, 1997)

Esta definición formalizo el concepto de “aprender” como un problema de optimización definido y cuantificable, dando como resultado un marco para el diseño y evaluación de algoritmos.

En el contexto de esta tesis, los componentes de la definición de Mitchell se instancian de la siguiente manera:

A. *Tarea (T):* La clasificación de declaraciones patrimoniales como 'normales' o 'anómalas'.

B. *Experiencia (E):* El conjunto de datos de declaraciones patrimoniales históricas utilizado para entrenar el modelo.

C. *Medida de Rendimiento (P):* Métricas de evaluación para clasificación en datos desbalanceados, como precisión, exhaustividad, puntuación F1 y el área bajo la curva ROC (AUC-ROC).

1.2.1 Proceso de Aprendizaje Automático

Para poder desarrollar un modelo de ML, es necesario seguir el siguiente flujo de trabajo de varias etapas clave (Bishop, 2006; Goodfellow et al., 2016):

1. *Recopilación y Preparación de Datos:* Esta etapa consiste en la recolección y preprocesamiento de los datos antes de comenzar con el modelado, aquí se hace limpieza, normalización y codificación de variables.
2. *Diseño de Características (Feature Engineering):* En este apartado se extraen de los datos las características más representantes de los patrones que se quieren identificar. En muchos de los modelos de aprendizaje profundo, este paso es automatizado por el mismo modelo (Goodfellow et al., 2016).
3. *Selección y Entrenamiento del Modelo:* Dependiendo de la naturaleza del problema y del conjunto de datos que se va a trabajar, se selecciona y entrena un modelo, utilizando una muestra del conjunto de datos originales (conjunto de entrenamiento).
4. *Evaluación del Modelo:* En cada iteración de entrenamiento se evalúa el modelo con un conjunto de datos no visto durante el entrenamiento (conjunto de prueba) de esta manera se obtiene una estimación imparcial de su capacidad de predicción.
5. *Ajustar Hiperparametros y Despliegue:* Se optimizan los hiperparametros del modelo (configuraciones que no aprenden automáticamente de los datos) y, finalmente, el modelo validado se implementa para trabajar con datos reales.

1.3 Paradigmas del Aprendizaje Automático

Existen tres paradigmas de algoritmos para el aprendizaje automático, cada uno definido por la naturaleza de los datos de entrenamiento y retroalimentación que recibe el sistema (Bishop, 2006; Goodfellow et al., 2016).

1.3.1 Aprendizaje Supervisado

El aprendizaje supervisado utiliza un conjunto de datos etiquetados, es decir, que a cada renglón del conjunto de datos tenga una categoría o salida correcta (Bishop, 2006). El objetivo es que el modelo sea capaz de aprender una función que categorice de manera correcta de nuevos datos. Esta tarea se puede realizar tanto para clasificación (salida categórica) y regresión (salida continua).

1.3.2 Aprendizaje No Supervisado

El aprendizaje no supervisado utiliza datos no etiquetados, por lo que el objetivo de los modelos es detectar los patrones o estructuras en los datos que no se ven a simple vista (Bishop, 2006; Goodfellow et al., 2016). Esto se utiliza para el agrupamiento (clustering), la reducción de dimensionalidad y la detección de anomalías.

Este paradigma es fundamental para la detección de anomalías en declaraciones patrimoniales. El aprendizaje supervisado requeriría un vasto conjunto de datos con casos

de corrupción previamente etiquetados, los cuales son, por definición, escasos y difíciles de obtener (Aggarwal, 2017). El aprendizaje no supervisado elude este problema al aprender la estructura de la "normalidad" a partir de la gran mayoría de los datos (que se asumen como legítimos), lo que lo convierte en el enfoque natural y más pragmático para este dominio (Chandola et al., 2009)

1.3.3 Aprendizaje por Refuerzo

La definición de Sutton & Barto (2018) es que el aprendizaje por refuerzo es un agente que debe tomar decisiones en un entorno para maximizar una recompensa acumulada. Es decir que el agente aprende a prueba y error, ayudado de un sistema de retroalimentación en forma de recompensas o castigos. Su utilidad es en tomas de decisiones secuenciales como en robótica o juegos.

1.4 Fundamentos de las Redes Neuronales y el Aprendizaje Profundo

Las Redes Neuronales Artificiales (RNA) son modelos basados en la estructura del cerebro humano, los componen una red de nodos de procesamiento interconectados (Bishop, 2006; Goodfellow et al., 2016).

1.4.1 Neurona Artificial (Unidad de Cómputo)

Lo más elemental de una RNA es la neurona artificial. Esta lleva a cabo un proceso de suma ponderada de las diferentes entradas de esta, más un sesgo que pasa a una función de activación no lineal (Goodfellow et al., 2016). La fórmula que representa esto es

$$\boldsymbol{y = \phi(\sum_i w_i x_i + b)} \quad \textbf{(1)}$$

Donde x_i son las entradas, w_i los pesos, b el sesgo y ϕ la función de activación. Al utilizar funciones no lineales en las funciones de activación permite que la red aprenda relaciones complejas.

1.4.2 Arquitectura de Redes Neuronales Artificiales (RNA)

Una RNA es una estructura de capas de neuronas interconectadas que se dividen en: capa de entrada, una o más capas ocultas y una capa de salida. En una red que es de alimentación hacia adelante (feedforward), la información fluye en una sola dirección, desde la entrada a la salida. Además, estas redes cuentan con un proceso de aprendizaje llamado retropropagación, el cual mediante una operación reajusta los pesos y sesgos de la red para lograr el menor error (Goodfellow et al., 2016).

1.4.3 Aprendizaje Profundo (Deep Learning, DL)

Dentro del ML existe el Aprendizaje Profundo (Deep Learning, DL) este utiliza redes neuronales con varias capas ocultas, llamadas redes neuronales profundas (LeCun et al., 2015). Esta “profundidad” permite aprender una jerarquía de características de manera automática desde los datos brutos, a esto se le conoce como aprendizaje de representación

(representation learning) (Goodfellow et al., 2016). El proceso que sigue es, las primeras capas aprenden las características más simples del conjunto de datos, las siguientes capaz logran representaciones más abstractas y complejas.

1.5 Detección de Anomalías

También conocida como detección de *outliers*, la detección de anomalías tiene como objetivo encontrar los datos que no se ajustan al comportamiento esperado (Chandola et al., 2009). Una anomalía según Aggarwal (2017) citando a Hawkins (1980) se conoce como "una observación que se desvía tanto de otras observaciones como para despertar sospechas de que fue generada por un mecanismo diferente". Esto es clave para dominios como la detección de fraudes, ciberseguridad y mantenimiento predictivo (Hodge & Austin, 2004).

1.5.1 Tipología de Anomalías

Las anomalías pueden ser clasificadas en tres tipos (Aggarwal, 2017; Chandola et al., 2009):

A. *Anomalías Puntuales:* Dícese de una instancia de datos individual que es anómala en comparación al resto del conjunto de datos. *Ejemplo:* Un servidor público con un salario modesto que declara un activo de un valor desproporcionado a su salario.
B. *Anomalías Contextuales:* Al igual que la anterior es un valor que sale de la media del conjunto de datos pero que solo lo hace bajo un contexto especifico. *Ejemplo:* Un incremento patrimonial del 200% en el primer año de un cargo público, sin una justificación clara, es anómalo en ese contexto.
C. *Anomalías Colectivas:* Un conjunto de instancias de datos relacionadas es anómalo en su conjunto, aunque las instancias individuales no lo sean. Ejemplo: Un servidor público que recibe un préstamo de un empresario que posteriormente gana múltiples licitaciones supervisadas por dicho servidor.

1.5.2 Enfoques Generales de la Detección de Anomalías

Según Aggarwal, (2017) y Chandola et al., (2009) se pueden categorizar los diferentes enfoques para atacar los problemas de detección de anomalías, los cuales son:

A. *Enfoque Estadístico:* Toma los valores que tienen baja probabilidad dentro de una distribución de probabilidad especifica de los datos y los cataloga como anomalías.
B. *Enfoque Basados en Proximidad:* Considera que los datos normales se concentran en cierta región y los valores que quedan fuera o estén aisladas son considerardas anomalías. Aquí existen algoritmos basados en distancia (k-NN) o densidad (DBSCAN).
C. *Enfoque Basado en Modelos de Aprendizaje:* Utilizando modelos de ML, aprende lo que es un dato “normal”. Las redes neuronales, más específicamente los **autoencoders**, forman parte de esta categoría ya que son especialmente potente para

aprender patrones complejos y no lineales en datos de alta dimensionalidad (Aggarwal, 2017).

1.6 Desafíos Fundaméntales

Durante el desarrollo de detección de anomalías se encuentran diferentes desafíos que hacen que los métodos estadísticos tradicionales y algoritmos de clasificación estándar sean insuficientes. Estos desafíos aumentan en los conjuntos de datos modernos, lo cual requiere de la adopción de arquitecturas de aprendizaje automático más complejas y modelados híbridos.

- *Desbalance Extremo de Clase:* Dentro de los conjuntos de datos que se utilizan para la detección de anomalías, dichas anomalías son la excepción a la regla. Por lo tanto, conduce a que los conjuntos de datos tengan un desequilibrio de clases muy pronunciado, donde la clase a identificar (anómala) puede que sea un porcentaje bajo en comparación con la clase normal, en algunos casos puede ser hasta 1% (Chandola et al., 2009; Thimonier et al., 2023). Esto afecta a los clasificadores convencionales, ya que suelen estar optimizados para maximizar la precisión general, por lo que desarrolla un sesgo importante a la clase mayoritaria. Como resultado, el modelo puede clasificar todas las instancias como "normales" y aun así tener una precisión general alta, fallando por completo en identificar las anomalías, que es el punto de interés (Gupta et al., 2020; Johnson & Khoshgoftaar, 2019).
- *Alta Dimensionalidad y Correlaciones Complejas:* Un problema al que se enfrentan los sistemas modernos es que no solo es una gran cantidad de datos si no que una gran cantidad de características también. Así es, los datos financieros pueden incluir cientos de atributos que representen una sola acción. Esta dimensionalidad crea un espacio de características grande y disperso, a este fenómeno se le conoce como la "maldición de la dimensionalidad" (Chandola et al., 2009). En estos espacios de alta dimensionalidad, los conceptos tradicionales como distancia y densidad se vuelven menos significativos, lo que hace que a los algoritmos basados en proximidad se les dificulte distinguir una acción normal de una anómala. Además, las anomalías se presentan a través de interacciones sutiles, y más importante, no lineales entre múltiples características, dificultando aún más la detección (Aggarwal, 2017).
- *Concept Drift y Naturaleza Adaptativa de las Anomalías:* Los fenómenos que se requieren estudiar sufren de un fenómeno llamado "concept drift" (deriva de concepto), esto es que los patrones anómalos del pasado difieren de los actuales (Gama et al., 2014). Esto requiere que los sistemas encargados de encontrar estas anomalías sean capaces de hacerlo más allá de los patrones anómalos encontrados durante el entrenamiento (Johnson & Khoshgoftaar, 2019; Lu et al., 2018).

Estos tres desafíos dan pie a que se proponga un cambio en el enfoque de la detección. La alta dimensionalidad proporciona un "gran espacio para esconderse" a las anomalías

novedosas, que se manifiestan con combinaciones de características no vistas. El desequilibrio de las clases hace que estos casos nuevos de anomalías sean difíciles de detectar por la poca información que se tiene, además de que posiblemente no este etiquetado como una anomalía, volviendo complicado el proceso de identificar anomalías a partir de ejemplos. Es por esto, que se propone una estrategia más robusta: "aprender el concepto de normalidad" a un nivel tan profundo que cualquier desviación, sin importar si ha sido vista anteriormente, se detecte como una anomalía. Esta necesidad es la que impulsa directamente la adopción de modelos de aprendizaje de representación no supervisados, como los **autoencoders**, que están diseñados precisamente para esta tarea de modelar la distribución de los datos normales.

1.7 Autoencoders

Los autoencoders son un tipo de red neuronal no supervisada utilizada para el aprendizaje de representación. El objetivo de esta arquitectura es aprender de una abstracción (codificación) de los datos de entrada y, a partir de esta, reconstruye la entrada original con la menor perdida posible (Goodfellow et al., 2016).

1.7.1 Principios y Arquitectura Fundamental

A los autoencoders lo componen dos componentes principales, los cuales según Goodfellow et al. (2016) son:

A. *Codificador (Encoder):* Comprime los datos de entrada de alta dimensionalidad a una representación más abstracta y de menor dimensionalidad.
B. *Cuello de Botella (Bottleneck):* Es la capa central que contiene la representación abstracta de los datos.
C. *Decodificador (Decoder):* A partir de la abstracción y de los datos de entrada originales se reconstruyen.

1.7.2 Fundamentos Matemáticos del Proceso de Reconstrucción

El entrenamiento de esta arquitectura es minimizar el error de reconstrucción, el cual es la diferencia de la entrada original x y la salida reconstruida x' (Goodfellow et al., 2016).

Se compararon los trabajos de Baldi (2012a) y Fahmi et al. (2024) y se puede deducir que, para datos de entrada con valores reales continuos, la función de pérdida más comúnmente utilizada es el Error Cuadrático Medio (Mean Squared Error - MSE). Esta función calcula el promedio de las diferencias al cuadrado entre cada elemento del vector de entrada x_i y el vector reconstruido $r_i = g(f(x_i))$. La función objetivo L a minimizar para un lote de N muestras se representa con la siguiente ecuación:

$$\boldsymbol{L(x, g(f(x))) = \frac{1}{N}\sum_{i=1}^{N} ||x_i - g(f(x_i))||^2} \quad \textbf{(2)}$$

Esta función penaliza fuertemente las grandes diferencias entre los valores originales y los reconstruidos (Fahmi et al., 2024).

Para datos de entrada binarios o categóricos (por ejemplo, representados como one-hot encoding), se suele utilizar la entropía cruzada binaria (binary cross-entropy), que es más adecuada para medir la discrepancia entre distribuciones de probabilidad (Baldi, 2012a).

1.7.3 Proceso de Entrenamiento

En el texto de Sakurada & Yairi (2014) define el proceso de minimización de la función de perdida como algoritmos de optimización basados en el gradiente, siendo el más común el descenso de gradiente (gradiente descent) y sus variables (por ejemplo, Adam, SGD). Esto se puede lograr utilizando el algoritmo de propagación (backpropagation) el cual permite calcular los gradientes de la función de perdida con respecto a cada uno de los parámetros de la red (LeCun et al., 1998).

Gracias a los trabajos de LeCun et al. (1998) y Goodfellow et al. (2016) se puede concluir el siguiente proceso:

1. *Pase hacia adelante (Forward Pass):* Una muestra de entrada x se pasa a través del codificador para obtener la representación latente $\boldsymbol{h}$, y después por el decodificador para obtener la reconstrucción $\boldsymbol{r}$.
2. *Cálculo de perdida:* Se calcula el error de reconstrucción $L(x, r)$ utilizando la función de perdida elegida.
3. *Pase hacia atrás (Backward Pass):* El algoritmo de retro propagación calcula el gradiente de la función de pérdida con respecto a cada peso y sesgo de la red, aplicando la regla de la cadena para propagar el error desde la capa de salida hacia atrás, hasta la capa de entrada.
4. *Actualización de Parámetros:* Los parámetros de la red se actualizan en la dirección opuesta al gradiente, multiplicados por una tasa de aprendizaje (), para reducir la pérdida en la siguiente iteración. La fórmula de actualización para un peso es:

$$\boldsymbol{w' = w - \alpha \times \frac{\partial L}{\partial w}} \quad \textbf{(3)}$$

Este proceso se repite iterativamente sobre el conjunto de datos de entrenamiento hasta que la función de pérdida converge a un valor mínimo, lo que indica que el modelo ha aprendido a reconstruir las entradas de la manera más fiel posible dadas las restricciones de su arquitectura.

1.7.4 Detección de Anomalías utilizando el Error de Reconstrucción

El mecanismo utilizado para la detección de anomalías utilizando autoencoders es entrenar el modelo utilizando únicamente datos normales (Aggarwal, 2017).

1. *Fase de Entrenamiento:* Al alimentar al autoenoder únicamente con instancias normales las aprende a reconstruir eficientemente. Al estar forzado a pasar la

información por el cuello de botella, debe aprender a identificar los patrones más importantes de los datos normales (Baldi, 2012b; Zhai et al., 2016).

2. *Fase de Inferencia:*
    a. Cuando se presenta una instancia normal, el modelo la reconstruye con un error de reconstrucción bajo (Zhou & Paffenroth, 2017).
    b. Cuando se presenta una instancia anómala, el modelo, que no ha sido entrenado en tales patrones, tendrá dificultades para reconstruirla, lo que resulta en un error de reconstrucción alto (Aggarwal, 2017).

El error de reconstrucción se convierte en la puntuación de la anomalía. Se establece un umbral y si la puntuación supera este umbral se clasifica como anómala (Aggarwal, 2017).

1.7.5 Limitaciones del Enfoque Básico

La principal limitación es que un Autoencoder con la suficiente capacidad (es decir, una red suficientemente profunda o con un cuello de botella no tan restrictivo) puede llegar a aprender a hacer muy buenas reconstrucciones. Esto puede parecer contradictorio, pero a lo que se refiere es que, en lugar de aprender la distribución subyacente de los datos normales, puede aprender una función de identidad trivial o casi perfecta (Goodfellow et al., 2016; Sakurada & Yairi, 2014). Es decir, un modelo de este tipo puede ser capaz de reconstruir cualquier entrada, incluyendo las anomalías, ya que aprende a copiar la entrada de la salida con mucha precisión, esto provocando que el error de reconstrucción para las anomalías sería bajo, haciéndolo indistinguible del de las instancias normales y, por lo tanto, inútil como métrica de detección (Nolle et al., 2022; Pang et al., 2021).

Es decir, que un Autoencoder sea útil para la detección de anomalías no es una propiedad inherente de la arquitectura, más bien es un efecto secundario resultante de la regularización adecuada.

Por lo que un modelo demasiado simple, no podría reconstruir los datos normales (underfitting). Por otra parte, si es un modelo demasiado complejo, podría reconstruir todo incluyendo anomalías (overfitting). Esto causa que el ajuste de hiperparametros sea crítico para confiar únicamente en el umbral sobre el error de reconstrucción, lo que resulta en una estrategia frágil (Baldi, 2012b; Sakurada & Yairi, 2014).

1.8 Modelos de Ensamblaje: Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting, o XGBoost, es un algoritmo de aprendizaje automático desarrollado por Tianqi Chen y Carlos Guestrin y actualmente está consolidado como una de las herramientas más potentes y populares para resolver problemas de clasificación y regresión con datos tabulares. El éxito de este algoritmo se le atribuye a su eficiencia, capacidad de escalabilidad y rendimiento predictivo (Chen & Guestrin, 2016; Nielsen, 2016; Shwartz-Ziv & Armon, 2022).

1.8.1 Estructura y Funcionamiento

Este modelo este compuesto por varios predictores que construyen un predictor robusto. Es decir, su principio fundamental consiste en mejorar el modelo de manera secuencial y aditiva, donde cada nuevo modelo se entrena para corregir los errores cometidos por los anteriores (Friedman, 2001).

- *Aprendices Débiles (Weak Learners):* Es el bloque fundamental de del XGBoost, típicamente conocido como árbol de clasificación y regresión (CART - *Classification and Regression Tree*). Un solo árbol tiene un poder predictivo limitado y es propenso al sobre ajuste, sin embargo, es capaz de encontrar algunas relaciones no lineales en los datos.
- *Proceso Aditivo y Secuencial:* El proceso que sigue XGBoost es, comenzar con una predicción inicial simple; después, en cada paso, añade un árbol nuevo al ensamblaje. A diferencia de métodos de ensamblaje como *Random Forest*, donde la creación de árboles es independiente, en XGBoost cada árbol nuevo creado es para corregir errores cometidos por los árboles anteriores (Bentéjac et al., 2021; Friedman, 2001).
- *Corrección de Errores (Residuals):* Estos árboles nuevos que se crean no es con el objetivo de predecir la variable original, sino que se utiliza para predecir los "residuos" o errores del modelo actual. Es decir, el residuo para una instancia dada es la diferencia entre el valor real y la predicción acumulada de todos los árboles hasta el momento. Al añadir estos árboles, se reduce el error general del ensamblaje.
- *Predicción Final:* Este proceso continúa hasta que se alcanza el numero predefinido de árboles o que ya no sea necesario agregar nuevos árboles para mejorar el rendimiento del modelo en un conjunto de validación. Se suman todas las predicciones de cada árbol individual que compone el ensamblaje y se obtiene la predicción final del modelo.

1.8.2 Fundamentación Matemática: Función Objetivo Regularizada

Según los trabajos de Chen & Guestrin (2016), Friedman (2001) y Parsa et al. (2020) la innovación clave de XGBoost, que lo distingue del *Gradient Boosting* tradicional, es su función objetivo regularizada. Cada iteración que el algoritmo añade un árbol que minimice la siguiente función objetivo es representada de la siguiente manera:

$$\boldsymbol{Obj^{(t)}} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{l}(\boldsymbol{y_i}, \boldsymbol{\hat{y}_i}^{(\boldsymbol{t-1})} + \boldsymbol{f_t}(\boldsymbol{x_i})) + \sum_{\boldsymbol{k=1}}^{\boldsymbol{t}} \boldsymbol{\Omega}(\boldsymbol{f_k}) \quad \textbf{(4)}$$

Donde:

- $\boldsymbol{l}(\boldsymbol{y_i}, \boldsymbol{\hat{y}_i})$: Es la función de perdida (error cuadrático para regresión, perdida logística para clasificación), que mide la discrepancia entre el valor real $\boldsymbol{y_i}$ y la predicción $\boldsymbol{\hat{y}_i}$.
- $\boldsymbol{\hat{y}_i}^{(\boldsymbol{t-1})}$: Es la predicción acumulada de los árboles anteriores (t – 1).

- $\boldsymbol{f_t(x_i)}$: Es la predicción del nuevo árbol por añadir.
- $\boldsymbol{\Omega(f_k)}$: Es el termino de regularización que penaliza la complejidad del k-ésimo árbol.

Optimización con Aproximación de Taylor

XGBoost utiliza una aproximación de Taylor de segundo orden de la función de perdida alrededor de la predicción actual $\boldsymbol{\hat{y}_i^{(t-1)}}$, esto con el objetivo de optimizar la función objetivo general para cualquier función de perdida diferenciable. Gracias a esto es posible rescribir la fórmula de manera que dependa únicamente de la estructura del nuevo árbol $\boldsymbol{f_t}$:

$$\boldsymbol{Obj^{(t)} \approx \sum_{i=1}^{n} l\left(g_i f_t(x_i) + \frac{1}{2} h_i {f_t}^2(x_i)\right) + \Omega(f_k)} \quad \textbf{(5)}$$

Donde g_i y $\boldsymbol{h_i}$ son la primera y segunda derivadas (gradiente y Hessiano) de la función de pérdida con respecto a la predicción, evaluadas en el paso anterior. Este uso de la información de segundo orden (el Hessiano) proporciona una dirección de optimización más precisa en comparación con el Gradient Boosting estándar, que solo utiliza el gradiente. Esto es análogo a utilizar el método de Newton-Raphson en lugar del descenso de gradiente en el espacio funcional (Chen & Guestrin, 2016).

Termino de Regularización ($\boldsymbol{\Omega}$)

Para prevenir el sobreajuste se utiliza una de las características distintivas de XGBoost, el término de regularización. Para un solo árbol $\boldsymbol{f}$, se define como (Parsa et al., 2020):

$$\boldsymbol{\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2} \quad \textbf{(6)}$$

- T es el número de hojas en el árbol.
- w_j es el peso asignado a la j-ésima hoja.
- Gamma ($\boldsymbol{\gamma}$): Este es un umbral de poda, asegura que una división de nodo se realice solo si la ganancia en la función de perdida es mayor que $\boldsymbol{\gamma}$. Entre mayor sea este valor más simples serán los árboles y por lo tanto se tendrá un modelo más conservador, es decir, esta es una forma de regular la complejidad de la estructura del árbol (Bentéjac et al., 2021).
- Lambda ($\boldsymbol{\lambda}$): Este es un parámetro de regularización L2 sobre el peso de las hojas, el cual se enfoca en penalizar puntuaciones de hojas de magnitudes más grandes. Con esto se consigue "suavizar" la predicciones finales del modelo, es decir, hace que ninguna hoja individual tenga una influencia desproporcionada del resto, esto con el objetivo de hacer que el modelo sea menos sensible a puntos de datos individuales (Parsa et al., 2020).

La potencia de XGBoost no proviene simplemente de la adición secuencial de árboles, sino de su enfoque de regularización por diseño. Los parámetros no son un ajuste posterior, sino que se integran directamente en el proceso de construcción del árbol. La "ganancia" de una posible división de un nodo se calcula teniendo en cuenta la penalización de la regularización. Esto significa que el algoritmo solo realizará una división si esta mejora la capacidad de generalización del modelo (según la función objetivo regularizada), no solo su ajuste a los datos de entrenamiento. Esta construcción de árboles "consciente de la regularización" es lo que hace a XGBoost tan robusto contra el sobreajuste y es una diferencia fundamental con respecto a técnicas como Random Forest, que construyen árboles profundos y luego dependen del promedio para regularizar.

1.8.3 Manejo de Datos Desbalanceados

Para atacar el problema de datos desbalanceados, XGBoost utiliza un parámetro que controla el balance entre las clases positivas y negativas. En caso de trabajar con un problema de clasificación binaria (como es el caso de la detección de anomalías) se asume que la clase "positiva" es la clase minoritaria (por ejemplo, "fraude"). El nombre de este parámetro es *scale_pos_weight*, que al establecer un valor mayor a 1, se aumenta el peso de la clase positiva en la función de perdida (Wang, 2022).

Para determinar el valor de este parámetro se sigue la recomendación de Chen & Guestrin (2016)la cual es utilizar el número de proporción entre instancias negativas sobre las positivas:

$$\boldsymbol{scale\_pos\_weight} = \frac{\boldsymbol{Número\ de\ muestras\ negativas}}{\boldsymbol{Número\ de\ muestras\ positivas}} \quad \textbf{(7)}$$

Este valor es utilizado durante el entrenamiento, según y los hessianos ($\boldsymbol{h_i}$) para actualizar el modelo, el peso de cada entrada que pertenezca a la clase positiva se multiplica por el valor de $\boldsymbol{scale\_pos\_weight.}$ Esto causa que el modelo reciba una penalización mayor cuando clasifica incorrectamente la clase minoritaria (un falso negativo) que por clasificar incorrectamente una entrada perteneciente a la clase mayoritaria (falso positivo). Esto resulta en un algoritmo que presta más atención a la clase minoritaria, aprendiendo a clasificarla con mayor precisión, pese a no tener tantos ejemplos.

1.9 Arquitectura Híbrida Autoencoder-XGBoost

Con lo mencionado anteriormente se puede decidir elegir esta combinación de Autoencoder (AE) con XGBoost como una estrategia que capitaliza las fortalezas complementarias de ambos modelos. Gracias a esta combinación, se crea un sistema con sinergias que atacan los problemas de la detección de anomalías en datos tabulares complejos, dícese, desequilibrio de clases, alta dimensionalidad y la necesidad de detectar patrones novedosos.

1.9.1 Aprendizaje de Representación y Clasificación de Alto Rendimiento

La estrategia es utilizar el modelo híbrido AE-XGBoost para la división de responsabilidades, donde cada componente se especializa en la tarea para la que es más adecuado:

- *Autoencoder como Ingeniero de Características No Supervisado:* El AE al ser entrenado con mayormente datos normales, aprende a modelar la variedad de los datos normales. Su función principal no se centra en la clasificación, más bien en la representación compacta y significativa de la "normalidad" (Aggarwal, 2017). Como consecuencia, se tiene un potente extractor de características no lineales y un detector de desviaciones, capturando la estructura de los datos.
- *XGBoost como Clasificador Supervisado de Alto Rendimiento:* Por otra parte, XGBoost funcionara como un clasificador supervisado. Tomará un conjunto de características (enriquecidas por el AE) y encontrar de forma precisa las diferencias entre las clases, aprovechando los mecanismos internos para manejar el desbalance de datos.

En esencia, el AE transforma el problema de uno de detección en un espacio de características crudo a uno de detección en un espacio de características semánticamente más rico, y XGBoost resuelve este último problema con una eficiencia y precisión superiores (Obushnyi et al., 2025).

REFERENCIAS

Aggarwal, C. C. (2017). *Outlier Analysis* (2nd ed.). Springer.

Baldi, P. (2012a). Autoencoders, Unsupervised Learning, and Deep Architectures. *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, *27*, 37–49.

Baldi, P. (2012b). Autoencoders, Unsupervised Learning, and Deep Architectures. *Proceedings of the Unsupervised and Transfer Learning Workshop, JMLR Workshop and Conference Proceedings*, *27*, 37–50.

Bentéjac, C., Csörgő, A., & Martinez-Munoz, G. (2021). A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, *54*, 1937–1967.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, *41*(3), 1–58.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Fahmi, A.-T., Kashyzadeh, K. R., Mazali, A., & Khorasani, K. (2024). Advancements in Gas Turbine Fault Detection: A Comprehensive Review on Deep Learning and Explainable AI. *Aerospace*, *11*(1), 5. https://doi.org/10.3390/aerospace11010005

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 1189–1232.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, *46*(4), 1–37. https://doi.org/10.1145/2523813

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Gupta, A., Tatbul, N., & Gottschlich, J. (2020). Class-weighted evaluation metrics for imbalanced data. *ArXiv Preprint ArXiv:2010.05787*.

Hodge, V., & Austin, J. (2004). A survey of outlier detection methodologies. *Artificial Intelligence Review*, *22*(2), 85–126.

Johnson, J. M., & Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, *6*(1), 27.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2018). Learning from non-stationary data: A survey on concept drift. *ACM Computing Surveys (CSUR)*, *51*(2), 1–37. https://doi.org/10.1145/3190503

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Nielsen, D. (2016). *Tree boosting with XGBoost-why does XGBoost win" every" machine learning competition?*

Nolle, T., Lu, T., Meisen, T., & Seel, T. (2022). Are autoencoders robust to outliers? *2022 International Joint Conference on Neural Networks (IJCNN)*.

Obushnyi, S., Virovets, D., Ramskyi, A., Zhytar, M., & Skladannyi, P. (2025). Variational Autoencoders for Detecting Anomalous and Fraudulent Transactions in Financial Systems. *Proceedings of the DECaT'2025: Digital Economy Concepts and Technologies*, 110–118. https://ceur-ws.org/Vol-3188/

Pang, G., Shen, C., van den Hengel, A., & Cao, L. (2021). Deep learning for anomaly detection: A review. *ACM Computing Surveys (CSUR)*, *54*(2), 1–38.

Parsa, A. B., Movahedi, A., Taghipour, H., Derrible, S., & Mohammadian, A. (Kouros). (2020). Toward safer highways, application of XGBoost and SHAP for real-time accident detection and feature analysis. *Accident Analysis & Prevention*, *136*, 105405.

Russell, S. J., & Norvig, P. (2004). *Inteligencia artificial: Un enfoque moderno* (2nd ed.). Pearson Educación.

Sakurada, M., & Yairi, T. (2014). Anomaly detection using autoencoders with nonlinear dimensionality reduction. *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*.

Shwartz-Ziv, R., & Armon, A. (2022). Tabular data: Deep learning is not all you need. *Information Fusion*, *81*, 84–90.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.

Thimonier, H., Popineau, F., Rimmel, A., & Doan, B. L. (2023). Deep Anomaly Detection on Tabular Data. *International Conference on Machine Learning*.

Wang, H. (2022). XGBoost-Based Fault Diagnosis for Photovoltaic Arrays. *Journal of Physics: Conference Series*, *2171*(1), 12020.

Zhai, S., Cheng, Y., & Lu, W. (2016). Deep structured energy based models for anomaly detection. *ArXiv Preprint ArXiv:1605.07717*.

Zhou, C., & Paffenroth, R. C. (2017). Anomaly detection with robust deep autoencoders. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.